**MASTER EN SCIENCES INFORMATIQUES**

# Animer l'exécution d'un programme
## une solution pour un outil d'aide à l'apprentissage de la programmation plus didactique

MOREELS, Martin

*Award date:*
2022

*Awarding institution:*
Universite de Namur

Université de Namur
Faculté d'informatique
Année académique 2021–2022

# Animer l'exécution d'un programme : une solution pour un outil d'aide à l'apprentissage de la programmation plus didactique

Martin Moreels


Promoteur : _____ (Signature pour approbation du dépôt - REE art. 40)
Benoît Vanderose

Co-promoteur : Julie Henry

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

# Table des matières

# Table des figures

# Remerciements

Je voudrais remercier dans un premier temps mon promoteur de mémoire Mr Benoît Vanderose, professeur à l'université de Namur, qui a cadré et validé le travail.

Je remercie également ma co-promotrice Mme Julie Henry, assistante à l'université de Namur, pour sa disponibilité, son expérience et ses conseils.

Enfin, je témoigne ma reconnaissance aux étudiants de l'université de Namur ayant été volontaires pour réaliser des entretiens.

# Résumé

De par leurs observations et expériences réalisées lors des cours de programmation à l'université de Namur, les professeurs et assistants remarquent une certaine difficulté des étudiants à comprendre et maîtriser des concepts tels que les fonctions, les conditionnelles et l'impact des instructions sur l'espace mémoire. A cela s'ajoutent des difficultés concernant les types des variables ainsi qu'au fait que le langage Python utilise le typage de manière dynamique.

Une piste déjà bien connue de la science pour résoudre certains de ces problèmes est la visualisation. Elle permet de réduire le niveau d'abstraction requis par les étudiants afin de comprendre le fonctionnement des concepts ciblés. Elle aide également les débutants à construire des modèles mentaux robustes et corrects de ces concepts.

L'objectif dans ce mémoire a été d'imaginer et tester un prototype d'outil, qui, couplé à un éditeur de code, permet de suivre étape par étape l'exécution d'un programme en assistant l'utilisateur avec des visualisations. Ces dernières se sont montrées efficaces lors de deux tests réalisés avec des étudiants de l'université de Namur. Les exercices utilisés durant ces tests ont permis de mettre en évidence certaines incompréhensions récurrentes et ont montré que les étudiants pouvaient anticiper certaines erreurs grâce aux visualisations.

# Introduction

La demande d'emplois dans le domaine de l'informatique et particulièrement dans celui du développement n'a cessé de croître depuis son invention. Selon Evans Data Corporation[3], le nombre de développeurs informatiques dans le monde était de 26,9 millions à la fin de l'année 2021. Ce nombre devrait évoluer à 45 millions d'ici 2030. En effet, l'âge moyen des développeurs est encore assez bas (moins de 30 ans en 2016 selon Stack Overflow[15]) et l'informatique continue de se populariser.

Le nombre croissant de ressources utilisables pour apprendre la programmation est un facteur clé de son développement. Les novices disposent de multiples plateformes sur internet telles que des vidéos, des tutoriels, des cours en ligne, des forums de discussion, etc. Il ne faut plus dépenser une grosse somme d'argent afin de pouvoir se procurer un ordinateur capable de programmer. Cependant, même si cet apprentissage est plus accessible à tous que jamais, les compétences à développer afin de devenir bon programmeur ne sont pas pour autant facile à acquérir.

Pour commencer, la programmation tend à effrayer les débutants et il est possible de vite se décourager. Pour un novice, une longue ligne de code peut paraître complètement insensée. En effet, le développement est un domaine très abstrait pour une personne ne s'y connaissant pas. Comment à partir de lignes de code va-t-on construire un programme interactif qui va stocker des données et les analyser ? L'apprentissage de la programmation requiert la construction de modèles mentaux corrects chez les novices. Ces modèles vont permettre aux apprenants de comprendre ce que certains patterns dans le code vont avoir comme effet sur le produit final. Créer ces modèles de manière correcte va demander une réflexion intense de la part de l'apprenant ainsi que de l'entraînement afin d'obtenir de l'expérience et mémoriser à long terme les différents concepts. Dépendamment du langage de programmation utilisé, le novice va également devoir apprendre à maîtriser une syntaxe plus ou moins complexe. L'ordinateur ne laissant aucun droit à l'erreur lorsqu'il compile ou exécute un programme, l'étudiant doit faire preuve de beaucoup de rigueur.

Ces challenges ne sont pas faciles à relever, c'est pourquoi des recherches sont effectuées afin de trouver des solutions à ces problèmes. Ces solutions peuvent être des méthodologies à suivre, des types d'exercices ou encore des logiciels pédagogiques à utiliser lors des premières activités de programmation. Une piste qui est déjà bien connue de la science est la visualisation.

La visualisation en programmation peut permettre aux novices de décomplexifier certains concepts liés au développement en rendant le code moins abstrait. La visualisation peut prendre de nombreuses formes afin d'atteindre différents objectifs. Dans le cadre de ce travail, l'objectif est d'imaginer un prototype d'outil, qui, couplé à un éditeur de code, permet de suivre en temps réel l'exécution d'un programme en assistant l'utilisateur avec des visualisations. Ces dernières permettraient de mettre des images sur les concepts suivants (du prioritaire au moins prioritaire) :

- L'allocation de la mémoire
- Les types des différentes variables et expressions

- Les fonctions
- Les conditionnelles

Cet outil doit être pensé afin de pouvoir être intégré dans le cadre des cours d'introduction à la programmation en BAC1 de l'Unamur. Les visualisations imaginées seront testées avec des étudiants de l'université de Namur afin d'analyser leur efficacité et se forger un avis concernant la pertinence d'utiliser un tel outil avec des étudiants débutants en programmation. Leur avis et leur motivation concernant cet outil sera également capturé. Dans un premier temps, seule la partie IHM (interaction homme-machine) sera travaillée.

Le travail se divisera en différentes parties. Pour commencer, un état de l'art recensera les différentes recherches ayant été effectuées sur le thème de la visualisation en programmation. Cela a pour objectif de découvrir ce qui existe déjà sur le sujet afin d'orienter correctement les recherches par la suite. Les différents types de visualisations ainsi que leur efficacité seront passés en revue.

La problématique du travail sera ensuite décrite ainsi que la/les question(s) de recherche servant de base et de fil conducteur pour la suite du travail.

La méthodologie adoptée pour effectuer l'ensemble des recherches sera ensuite décrite. Cela comprendra le processus suivi afin de créer, tester et améliorer un prototype d'outil à l'aide des deux tests utilisateur réalisés à l'école.

Les visualisations créées seront ensuite expliquées et affichées. Une description des tests utilisateurs ainsi que des données récoltées prendra place.

Les résultats de ces analyses seront finalement interprétés lors de la discussion.

# Chapitre 1

# Etat de l'art

Cet état de l'art a pour objectif de recenser les recherches scientifiques effectuées sur le thème de la visualisation en programmation. Son fonctionnement ainsi que les différents types de visualisations seront expliqués, ainsi que les problèmes qu'ils peuvent résoudre. Différents types d'outil de visualisation existants seront passés en revue afin de se faire une idée globale des fonctionnalités existantes dans ce domaine.

## 1.1 L'apprentissage de la programmation

### 1.1.1 Difficultés des novices en informatique

La programmation fait partie des compétences les plus importantes et demandées en informatique. Quelle que soit la branche de l'informatique choisie, les étudiants feront toujours face à la programmation à un moment ou un autre. Ce n'est cependant pas la compétence la plus facile à acquérir. En effet, l'informatique est la combinaison de plusieurs branches : les mathématiques, l'ingénierie et la science. Il n'est pas possible de correctement comprendre la programmation en étant concentré sur une seule de ces branches [13]. Plusieurs concepts seront rencontrés par les novices en programmation tels que l'utilisation des variables, les itérations, les conditions, la modularité d'un programme avec des fonctions, les opérations sur les tableaux ou listes, etc. Ces concepts peuvent poser des difficultés qui sont causées par différentes raisons :

- La difficulté des novices à comprendre la sémantique des concepts en programmation
- La difficulté à avoir un regard abstrait sur un programme
- Le fait que les débutants aient du mal à comprendre ce que les changements dans le code vont avoir comme effets sur l'exécution du programme
- La difficulté à exprimer des solutions via l'utilisation d'un langage de programmation spécifique

Des solutions à ces différentes difficultés seront recherchées dans la suite de ce travail à l'aide de la visualisation.

### 1.1.2 L'abstraction : un concept clé de l'informatique

L'apprentissage de la programmation requiert de savoir manipuler toutes sortes d'entités abstraites, qui n'ont souvent pas grand chose en commun avec ce qu'on voit dans le monde réel. Ces

entités peuvent aussi bien concerner le langage de programmation utilisé que le fonctionnement général de la programmation[18]. L'abstraction est donc un concept très important dans l'informatique que les étudiants doivent apprendre à maîtriser rapidement.

Pour apprendre le développement, il n'est pas forcément nécessaire de comprendre comment les ordinateurs exécutent les programmes à un très bas niveau, en utilisant par exemple les jeux d'instructions du processeur de l'ordinateur[7]. Les opérations de l'exécution d'un programme peuvent être abstraites en utilisant des concepts de plus haut niveau. Pour comprendre cela, B. Du Boulay[5] a introduit le concept de *Machines notionnelles*.

Une machine notionnelle est formée par l'ensemble de concepts de plus haut niveau. Il s'agit d'un ordinateur idéalisé "whose properties are implied by the constructs in the programming language employed"[du Boulay[5]]. Cette machine offre donc aux étudiants une couche d'abstraction supplémentaire concernant l'ordinateur. Son but est de seulement garder les caractéristiques importantes de l'ordinateur dans son seul rôle d'exécuteur de code dans un langage spécifique ou un groupe de langages spécifiques[24]. Une machine notionnelle englobe toutes les capacités et comportements abstraits aussi bien hardware que software, assez bien détaillés, dans un certain contexte, que pour expliquer comment les programmes sont exécutés et quelles sont les relations des commandes du langage utilisé avec l'exécution du code[24].

Une machine notionnelle est spécifique à un langage de programmation et parfois au contexte d'exécution du programme. Une machine utilisée pour un langage orienté objet sera différente d'une machine pour langage fonctionnel. De plus, un seul et même langage de programmation peut être associé à différentes machines notionnelles, en fonction de pourquoi et comment ce langage est utilisé[24].

Pour résumer, une machine notionnelle [24] :

- Est une abstraction idéalisée de l'ordinateur et des autres aspects de l'environnement d'exécution d'un programme
- Sert à comprendre ce qu'il se passe lors de l'exécution d'un programme
- Est associée avec un ou plusieurs paradigmes de programmation ou langages, et possiblement avec un environnement de programmation spécifique
- Permet de décrire la sémantique d'un programme écrit dans un certain langage
- Donne une perspective particulière de l'exécution des programmes
- Reflète correctement ce que font les programmes quand ils sont exécutés

La visualisation va permettre aux novices de comprendre comment les programmes sont exécutés par ces machines notionnelles.

### 1.1.3 Les misconceptions des novices en programmation et les modèles mentaux

Il existe plusieurs misconceptions connues par rapport à l'utilisation des concepts de programmation. Dans le cadre de ce travail, il sera utile de les comprendre afin de pouvoir utiliser la visualisation. Cela aidera les étudiants à former des modèles mentaux corrects de ces différents concepts.

Quand ils commencent à programmer, les étudiants ont des préconceptions qui proviennent de leurs connaissances antérieures. Ces connaissances peuvent provenir de plusieurs branches comme les mathématiques, l'informatique ou le langage naturel. Prenons ce petit programme comme exemple[30] :

```
a = 100
a = a + 1
print(a)
```

Beaucoup de novices, au lieu d'additionner 100 avec 1 pour trouver la nouvelle valeur de la variable 'a' vont comprendre la deuxième instruction comme une équation mathématique qu'il est impossible de résoudre. Certains vont donc penser qu'après la troisième instruction, la variable 'a' contiendra l'équation complète sous la forme d'une String[30]. Il s'agit d'une misconception concernant le signe d'égalité '=' qui est utilisé comme assignation en programmation.

Voici un deuxième exemple :

```
a = 100
a = 20
print(a)
```

Ici, beaucoup de débutants peuvent penser que la variable 'a' contient la valeur 100 ou 120 après la troisième instruction. Il a deux misconceptions possibles :

- Une variable ne peut pas changer de valeur et la deuxième instruction ne fait donc rien. 'a' vaut donc 100.
- La variable va contenir la somme de toutes les valeurs qu'on lui attribue au cours du programme. 'a' vaut 120.

Les préconceptions, elles, ne sont au final que des connaissances naïves que se fait l'étudiant quand il fait face à du code pour les premières fois. Elles sont généralement facile à corriger ou à supprimer. Les misconceptions sont plus problématiques car elles apparaissent quand les étudiants se construisent des modèles mentaux incorrects de certains concepts. Les étudiants vont assimiler de la nouvelle matière qu'ils vont potentiellement lier avec des modèles existants incorrects[30].

Pour résumer, un modèle mental est une représentation personnelle d'un concept en particulier. Pour qu'un modèle mental soit robuste, il doit être totalement indépendant d'un contexte. Par exemple, un étudiant va se construire un modèle mental d'une boucle for en Python. Le fonctionnement de ce modèle ne doit pas dépendre du code qu'il y autour de cette boucle for. L'étudiant pourra ainsi utiliser ses différents modèles quand il en aura besoin et de manière efficace.

Les concepts de modèles mentaux et machines notionnelles sont fortement liés. L'objectif pour chacun est de se former un modèle mental robuste d'une machine notionnelle spécifique. Ce modèle mental sera divisé en sous modèles pour chaque concept d'une machine notionnelle spécifique. Pour une machine de Python par exemple, ces concepts pourraient être les variables, les conditionnelles, l'espace mémoire ou encore les instructions (avec leurs spécificités propres à Python).

## 1.2 La visualisation en programmation

### 1.2.1 Objectifs

La visualisation permet de représenter avec des images des concepts abstraits. Cela aide les gens à se créer des représentations mentales de ces concepts afin de les retenir et de mieux les comprendre.

N'ayant souvent pas des modèles mentaux corrects et stables, les débutants voient régulièrement les programmes comme des "boites noires". C'est à dire qu'ils ne comprennent pas bien comment le programme arrive à un certain résultat. L'utilisation de la visualisation va leur permettre de baisser le niveau d'abstraction requis et de voir leurs programmes comme des "boites en verre". Ils pourront comprendre les étapes par lesquelles le programme passe afin d'arriver au résultat final.

La visualisation peut être utilisée par différents acteurs lors de l'apprentissage et de différentes manières :

- Les étudiants peuvent l'utiliser lorsqu'ils programment afin de voir comment leur code est exécuté par la machine. Ils peuvent s'en servir pour trouver les différents bugs qu'ils ont écrit. Certains software proposent une visualisation instantanée lorsque le code est rédigé, ce qui permet une exploration interactive et immédiate de la sémantique du langage utilisé.

- Les enseignants peuvent utiliser la visualisation pour les assister dans l'explication de certains programmes ou concepts de programmation.

- La visualisation peut aussi être utilisée comme plate-forme d'apprentissage. Certains logiciels proposent des exercices intégrés. Le software UUhistle[26] permet aussi à l'étudiant de jouer le rôle de l'ordinateur lors de l'exécution de code. Il devra lire le code, suivre les instructions, allouer la mémoire aux différentes variables et objets, évaluer des expressions, assigner des valeurs, passer des arguments, etc. Ce type de visualisation s'appelle VPS (Visual Program Simulation).

Il existe différentes sortes de visualisation software :

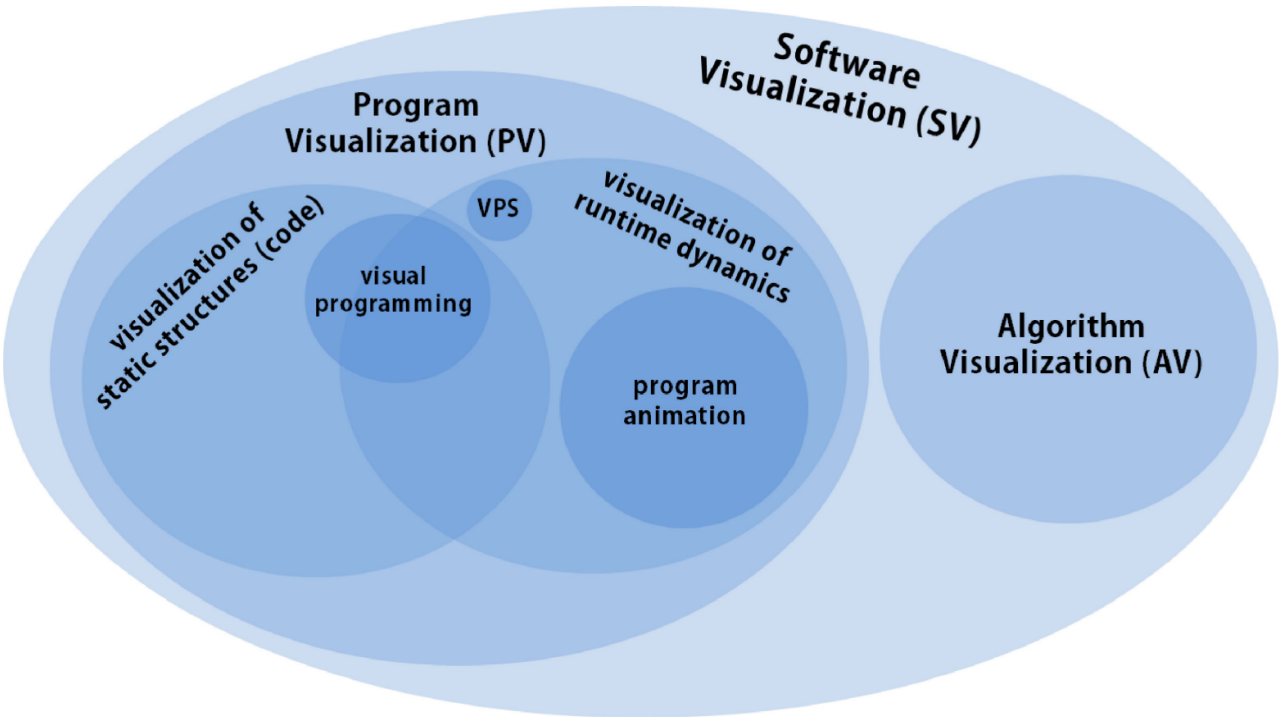



FIGURE 1.1 – Les différents types de visualisation software[25]

### 1.2.2 La visualisation de programmes

La visualisation de programmes permet entre autres de mettre en évidence le flux d'exécution du programme ainsi que les structures de données utilisées par le programmeur. Elle apporte une aide au novice pour qui l'exécution d'un programme est extrêmement abstraite. Ces représentations peuvent être très différentes en fonction du langage de programmation utilisé. C'est ce type de visualisation qui sera principalement exploré dans ce travail. Plusieurs autres types de visualisation la composent.

#### La visualisation de code

Permet d'illustrer la structure du code d'un programme, comme par exemple son évolution ou les dépendances d'un projet.

#### La simulation visuelle de programmes

Permet de faire jouer un rôle à l'utilisateur dans l'exécution d'un programme. Cet utilisateur va prendre le rôle de l'ordinateur et lire le code, suivre les instructions dans le bon ordre, emprunter correctement les structures de contrôle et garder une trace de l'état du programme [23]. Cela sera assisté par la visualisation afin de garder une trace des différents événements.

#### L'animation de programmes

Permet d'exécuter un programme étape par étape (comme avec un debugger traditionnel) en assistant l'utilisateur avec des visualisations.

### 1.2.3 La visualisation d'algorithme

Représente un algorithme informatique avec un très haut niveau d'abstraction. Ce type de visualisation n'est généralement pas lié à un langage de programmation en particulier. De plus, cela ne se trouve pas dans la partie "Visualisation de programmes" car on s'intéresse purement aux algorithmes et non à leur intégration dans un quelconque programme.

### 1.2.4 Langages de programmation visuels et textuels

Il existe deux groupes de langages de programmation ; les VPL's (Visual Programming Language) et les TPL's (Textual programming Language). Les VPL fournissent au développeur des éléments graphiques qu'il peut manipuler afin de rédiger son programme. Cette méthode est également appelée *Visual Programming*. Aucun code à proprement parler ne doit être rédigé afin d'arriver à un programme fonctionnel. Un exemple de langage visuel est scratch. Scratch est destiné à l'apprentissage de la programmation chez les enfants. Il est utilisable via l'application ou même via le site web. L'utilisateur peut créer des animations, des jeux ou encore des histoires avec les éléments qu'il dessine. Afin de rendre le tout interactif, l'utilisateur peut utiliser des blocs de code en les faisant glisser. Ces blocs peuvent contenir différentes instructions ainsi que des conditions, boucles etc.

A l'inverse, les langages de programmation textuels(TPL's) sont les langages utilisés dans la vie de tous les jours, tels que Python ou Java.

Les VPL's, bien qu'intéressants lors de l'apprentissage du développement chez les enfants comportent certains défauts[13].

Premièrement, les étudiants ne font face à aucuns problèmes de syntaxe en utilisant un VPL. Savoir maîtriser la syntaxe d'un langage est une compétence clé en informatique.

Deuxièmement, les VPL's favorisent la programmation dite *spaghetti*. C'est à dire une programmation peu claire et lisible dans laquelle il y a trop de sauts inconditionnels. L'utilisateur programme avec les éléments qu'il a à disposition sans penser à rendre la structure de son code pertinente. Les VPL's s'utilisent également avec une approche ascendante et non descendante. Cela veut dire que le développeur construit son programme en partant des éléments primitifs (qu'il a dessiné) pour créer de nouveaux éléments qui seront à leur tour employés ensuite. Dans la pratique, il est nécessaire de programmer avec une approche descendante. Le développeur doit prendre du recul et doit partir des exigences du programme final afin de définir des détails d'implémentation pertinents.

Troisièmement, l'abstraction et la modularisation ne sont pas entraînés lors du développement via VPL. En effet, le niveau d'abstraction requis est moindre car les éléments de base du programme sont dessinés à un niveau bien plus haut que si on les définissait via un langage textuel.

Finalement, les novices utilisant des VPL's pour apprendre la programmation ont des problèmes de compréhension concernant l'utilisation des variables, des opérateurs et des expressions qui sont pourtant des concepts clés[13].

Ces compétences que l'utilisation des VPL's ne travaille pas sont pourtant primordiales dans le monde du développement. Étant donné que l'objectif final est la maîtrise de la programmation via TPL, qui est la seule forme de langage utilisée dans le monde du travail et du développement, il est judicieux d'apprendre à les utiliser directement.

### 1.2.5 L'efficacité de la visualisation dans l'apprentissage de la programmation

J. Sorva et Al[25] en 2013 ont analysé 46 systèmes de visualisation software créés entre 1979 et 2012. Leur étude à montré que leur efficacité n'était pas stable et qu'il n'était pas encore certain que ces systèmes soient réellement efficaces. Les auteurs de l'article [13] ont également analysés l'efficacité de la visualisation en donnant cours à des étudiants. Une partie des étudiants ont suivi un cours classique sans visualisation, et l'autre s'est vu utiliser Online Python Tutor[6]. Aucune différence notable de performance après le stade d'apprentissage n'a été observée.

Plusieurs travaux [13][7] ont démontré que différentes conditions étaient requises afin de rendre l'utilisation de la visualisation efficace avec les étudiants. En effet, cela dépend fortement du niveau d'implication des novices dans les différents exercices. Naps et Al. [14] ont introduit une taxonomie d'engagement qui décrit les différents taux d'interaction possible avec la visualisation. 6 niveaux différents ont été identifiés : no viewing, viewing, responding, changing, constructing et presenting.

Ces 6 niveaux sont listés dans un ordre croissant, du moins attentif et impliqué (no viewing) au plus impliqué (presenting). Contempler les mêmes animations encore et encore sans aucune interaction n'est pas optimal du tout. Les étudiants finissent par utiliser la visualisation comme debugger et n'assimilent pas correctement sa sémantique[13].

Une étude[16] réalisée par eemu Rajala, Mikko-Jussi Laakso, Erkki Kaila, and Tapio Salakoski analysant les performances de Ville, un outil de visualisation, a démontré que son efficacité était significativement améliorée une fois que le niveau d'engagement des participants était au dessus de "viewing'. Il est donc essentiel de garder le taux d'engagement des étudiants au niveau "responding"

au minimum.

Une autre caractéristique à prendre en compte lors de l'enseignement avec visualisation est la théorie cognitive de l'apprentissage du multimédia. Cette théorie affirme que le traitement de l'information de l'être humain est accomplie via deux canaux distincts : visuel et audio. Chaque canal a une capacité limitée[11]. Cela veut dire qu'il ne suffit pas juste d'ajouter des images ou des visualisations pour que l'apprentissage devienne optimal chez les débutants. De plus, se limiter à la visualisation lors de l'apprentissage peut potentiellement causer la surcharge cognitive du canal visuel de l'étudiant, ce qui serait contre productif. Idéalement, la visualisation doit être accompagnée d'explications orales afin d'exploiter le canal audio du novice.

Comme indiqué ci-dessus, l'efficacité des outils de visualisation développés entre 1979 et 2012 est encore assez floue. L'importance du niveau d'engagement de l'apprenant afin de rendre l'apprentissage plus efficace a été démontrée mais ce n'est pas la seule caractéristique à prendre en compte lors du développement d'une plate-forme d'apprentissage.

Les auteurs de l'article[7] ont identifié 16 principes pouvant impacter l'efficacité d'un outil d'apprentissage. Pour ce faire, ils se sont basés sur la théorie d'apprentissage de Vygotsky's. Ces principes sont des étapes par lesquelles il faut faire passer l'apprenant lors des activités d'apprentissage.

Dans cet article ont été analysés 6 nouveaux systèmes entre 2012 et 2015. Plutôt que regarder la mise en forme de la visualisation en elle-même, ils se sont intéressés à la pédagogie et aux méthodes proposées par chaque plate-forme d'apprentissage. Leur but était donc de découvrir comment les auteurs des différentes solutions proposaient d'utiliser la visualisation avec des novices. Il a été découvert que seulement 2 des 16 principes étaient utilisés par ces nouveaux systèmes.

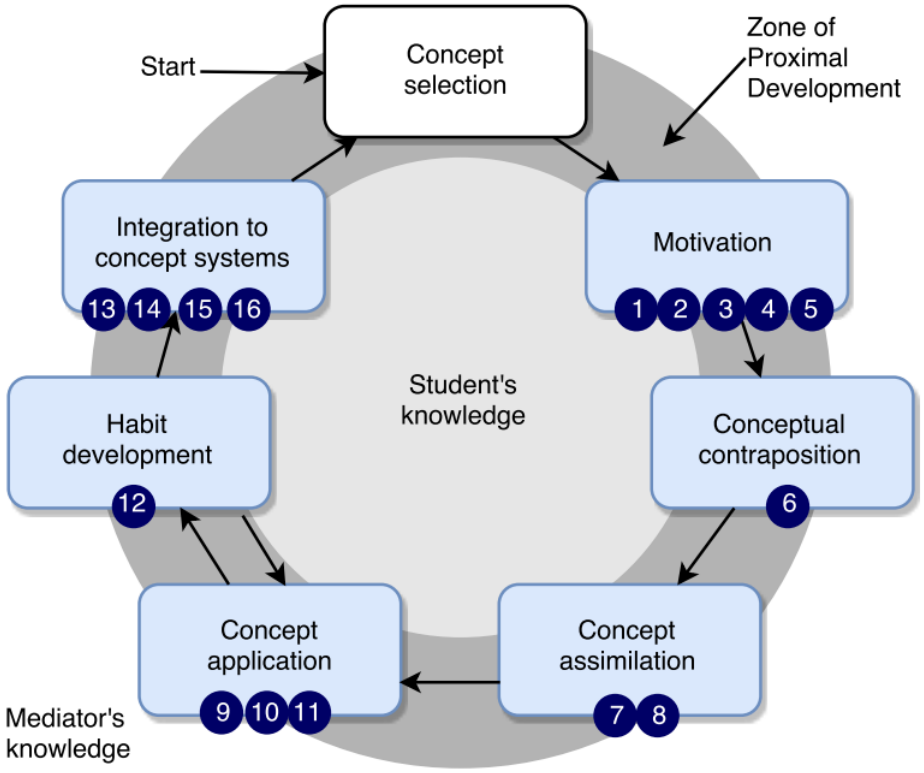



FIGURE 1.2 – Les stades d'un processus d'apprentissage et leurs principes respectifs[7]

Comme le montre la figure1.2, le premier stade d'apprentissage est le stade de motivation. Le but ici va être de placer l'apprenant d'un état d'esprit actif et positif. Ce dernier va devoir se sentir

impliqué et motivé d'apprendre. Pour cela, il va falloir rendre l'exercice interactif pour que l'étudiant ne soit pas qu'un simple observateur. Les 5 principes de ce stade sont :

1. Promouvoir un état d'esprit actif

2. Trouver ce qui motive l'étudiant à apprendre (réussir son année, devenir programmeur...)

3. Organiser les nouveaux concepts avec ceux connus

4. Evaluer les notions/émotions précédentes liées au concepts à apprendre. Le but est de pouvoir les positiver si elles sont négatives

5. Améliorer l'intérêt de l'étudiant dans la tâche académique. Par exemple, le professeur peut expliquer en quoi le nouveau concept est très important à maîtriser.

Les nouveaux concepts s'apprennent en les associant aux concepts déjà connus. Il se peut que ces derniers ne suffisent pas pour créer des liens. Dans ce cas, le stade de "Conceptual contraposition"(principe 6) peut être efficace. Son but va être de faire réaliser à l'étudiant que ce qu'il connaît déjà ne suffit pas à remplir l'objectif qu'il est motivé à atteindre. Un moyen d'y arriver est de demander à l'étudiant de résoudre un problème qui requiert les nouvelles notions (avant de les avoir apprises). Lorsque l'apprenant réalisera par lui-même qu'il ne lui est pas encore possible de réaliser la tâche car il n'a pas les compétences requises, il sera forcé de réorganiser son savoir et acquérir les nouvelles notions.

Le troisième stade de l'apprentissage est le stade d'assimilation des concepts. Il est divisé en deux principes :

7. Visualiser les nouveaux concepts et les associer avec les notions précédentes. Le professeur peut aider l'étudiant à trouver des informations par lui-même ou bien simplement lui les donner directement.

8. Encourager l'étudiant à comparer activement les nouveaux concepts avec les anciens (trouver des différences et des points communs, afin de supprimer les associations erronées)

Afin de mémoriser et comprendre correctement les nouveautés, il faut passer par la phase d'application :

9. Le professeur donne une série d'exercices à l'étudiant. Ces exercices aident les étudiants à identifier les propriétés du nouveaux concept ainsi que ses liens avec les précédents.

10. Les novices ont besoin d'un feedback immédiatement après les premiers exercices. Ce feedback peut venir du professeur ou bien être implicite dans les exercices. Il va permettre à l'apprenant de valider ou non sa compréhension.

11. Les nouveaux concepts doivent être appliqués à des situations pratiques de la vie réelle. Si ce n'est pas le cas, ils n'apporteront rien à l'expérience de l'étudiant.

Après avoir réalisé ses premiers exercices, l'étudiant va devoir développer une habitude concernant les nouveaux concepts. L'application de ces derniers doit devenir naturelle. L'étape 12 comprend

donc des nouveaux exercices qui vont permettre à l'apprenant d'utiliser ses nouveaux acquis dans plusieurs cas de figure différents.

Le dernier stade de l'apprentissage "Integration to concept systems" est divisé en 4 points :

13. Des problèmes complexes demandant l'application des nouveaux concepts en lien avec d'autres concepts doivent être rencontrés par l'étudiant.

14. La performance de l'étudiant lors de résolution de problèmes doit être évaluée.

15. Le fait que la compréhension des différents concepts est correcte et stable sur le temps doit être évalué.

16. Le fait que les étudiants soient capables d'appliquer les nouveaux concepts à des nouvelles situations est évalué.

## 1.3 Outils de visualisation existants

Plusieurs outils de visualisation de programmes existent déjà, et cela pour plusieurs langages. Certains se limitent seulement à l'affichage de visualisations alors que d'autres offrent plus de fonctionnalités, telles que des outils de collaboration, l'animation des différentes visualisations, des exercices interactifs, etc. Les outils abordés lors de cette section seront (liste non-exhaustive de tous les outils existants) : Thonny, PlanAni, Online Python Tutor, Collece-2.0, UUhistle, BlueJ, Jgrasp et Ville.

Les fonctionnalités intéressantes de ces différents outils ainsi que leurs avantages/inconvénients vont être discutées dans cette section.

### 1.3.1 Contenu des variables - Exécution du code step by step

Cette fonctionnalité est celle qui revient le plus dans les outils de visualisation. L'utilisateur peut exécuter le programme étape par étape en voyant directement quels effets chaque ligne a sur le contenu des variables, comme il pourrait également le faire avec un debugger classique. La forme avec laquelle le contenu des variables est affiché dépend fortement de l'outil utilisé. Par exemple, l'outil *Thonny*[2] se contente d'afficher le contenu de chaque variable sans aucune animation ni mise en forme spécifique, comme le montre la figure 1.3.

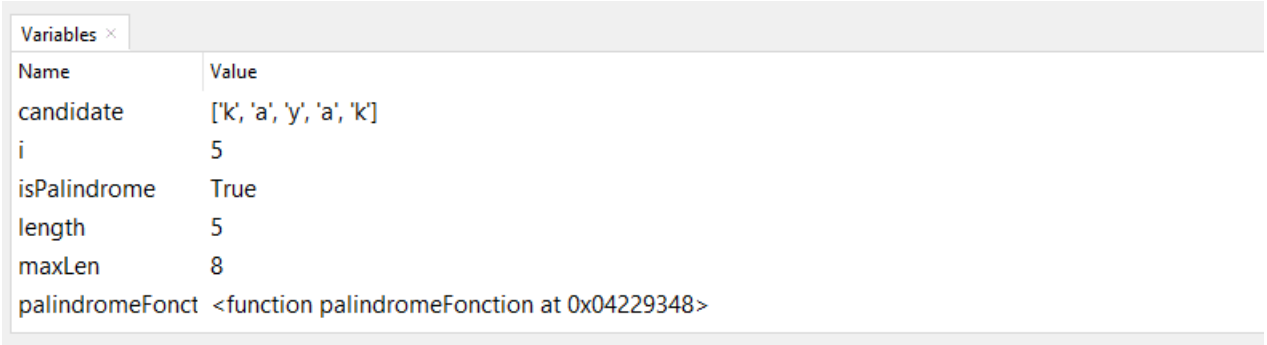



FIGURE 1.3 – L'affichage du contenu des variables dans l'outil Thonny

### 1.3.2 Ouvertures de nouveaux contextes pour les fonctions

Lors de l'appel à une fonction, un nouveau contexte (ou frame) portant le nom de la fonction apparaît et affiche le code qui s'y trouve ainsi que le contenu des variables locales à cette fonction. Cela met en évidence qu'une fonction est indépendante du reste du code et peut être appelée à tout moment dans le programme principal. Les variables déclarées dans cette fonction ne pourront pas être utilisées à l'extérieur.

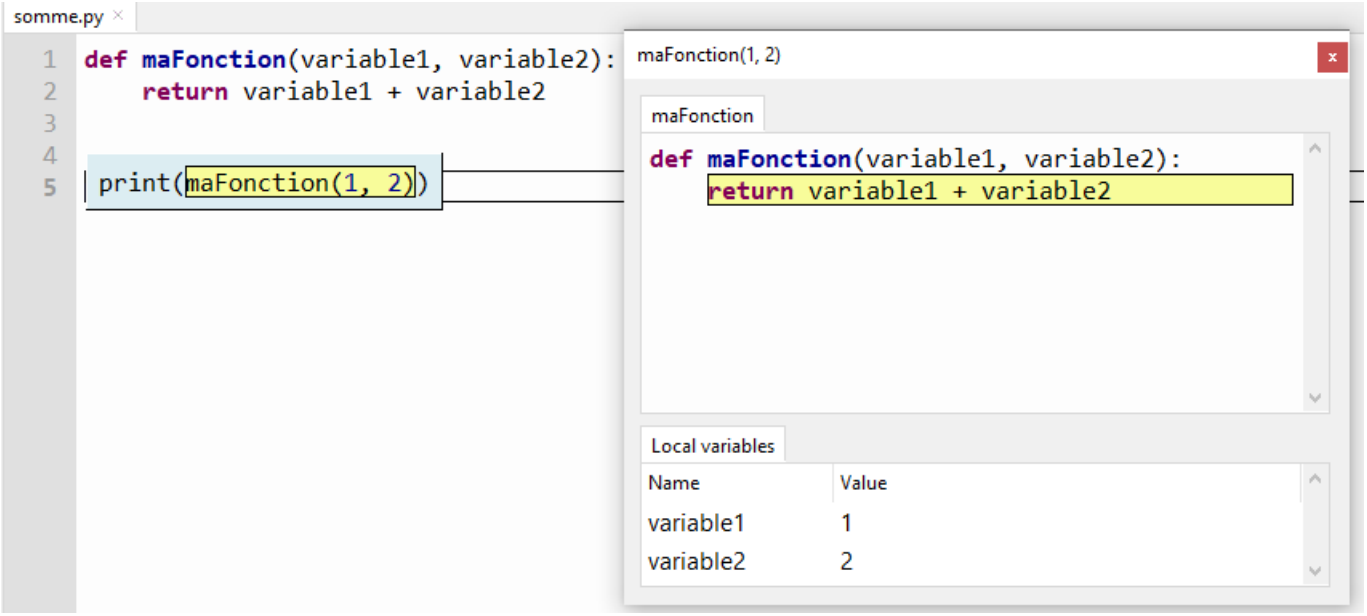



FIGURE 1.4 – Un nouveau contexte ouvert pour la fonction maFonction dans Thonny

### 1.3.3 Affichage des stack frames et heap

#### Online Python Tutor

Online Python Tutor[6](voir figure 1.5) affiche les variables dans leur contexte. La partie visuelle de l'application est divisée en deux colonnes : Frame et Objects. Dans la partie Frame se trouvent les différents contextes liés aux fonctions. Les variables déclarées dans le programme principal se trouvent dans le contexte "Global frame". Un nouvel espace est créé à chaque appel de fonction. Chaque valeur de variable est représentée par une flèche qui pointe à un objet se trouvant dans la partie "Objects". Même si techniquement, en Python chaque objet se trouve dans la Heap (partie "objects"), les créateurs de Online Python Tutor ont décidé de rendre immuable les objets primitifs, ce pourquoi leurs valeurs se trouvent dans la partie "Frame" (correspondant à la stack). Ce choix a été fait de manière à rendre la visualisation plus facilement compréhensible. Le grand avantage de la séparation stack/heap est qu'il est aisé de remarquer les partages de références, qu'on ne voit pas quand seul le contenu des structures de données est affiché.

#### UUhistle

UUhistle[26] (voir figure 1.6), un software de visualisation de programmes Python, divise encore plus l'affichage des différents éléments d'un programme avec ces différents menus :

- *Data in heap* : Affiche les différents objets présents dans la heap.

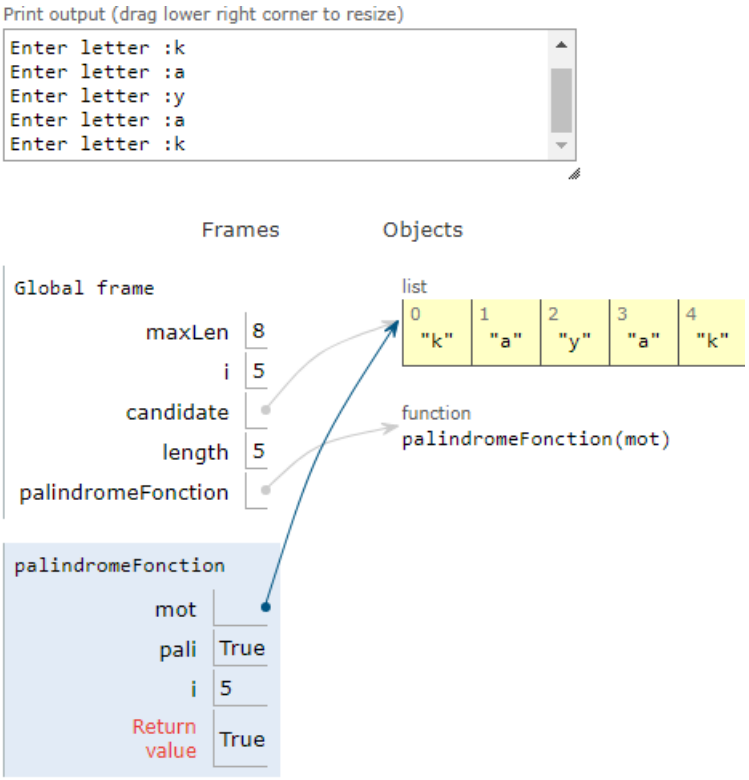



FIGURE 1.5 – Online Python Tutor exécutant un simple programme

- *Call stack* : Affiche les différentes frames du programme. Une première frame est affichée et contient les expressions et variables globales du programme. Une frame supplémentaire est créée pour chaque fonction présente dans le programme.
- *Classes* : Pour les programmes orientés objets, chaque classe utilisée est affichée ainsi que les différentes fonctions qu'elle contient.
- *Functions* : Affiche les différentes fonctions *built-in* utilisées et non les fonctions déclarées par l'utilisateur ! Cela pourrait prêter à confusion.
- *Operators* : Affiche les opérateurs qui sont utilisés par le code étant exécuté au moment même.

### 1.3.4 Visualisation des diagrammes de classe (pour les langages OO)

#### BlueJ[28]

BlueJ (voir figure 1.7) est une plateforme d'apprentissage du langage Java. L'outil affiche les différentes classes d'un projet et leurs relations sous forme de diagrammes de classes UML. Il est possible pour les étudiants de voir les différents objets qui ont été créés à partir des classes. La manipulation des classes et objets peut être effectuée directement depuis l'interface, sans passer par l'écriture de code. Les novices peuvent ainsi travailler les concepts de la programmation orientée objet sans devoir maîtriser le langage.

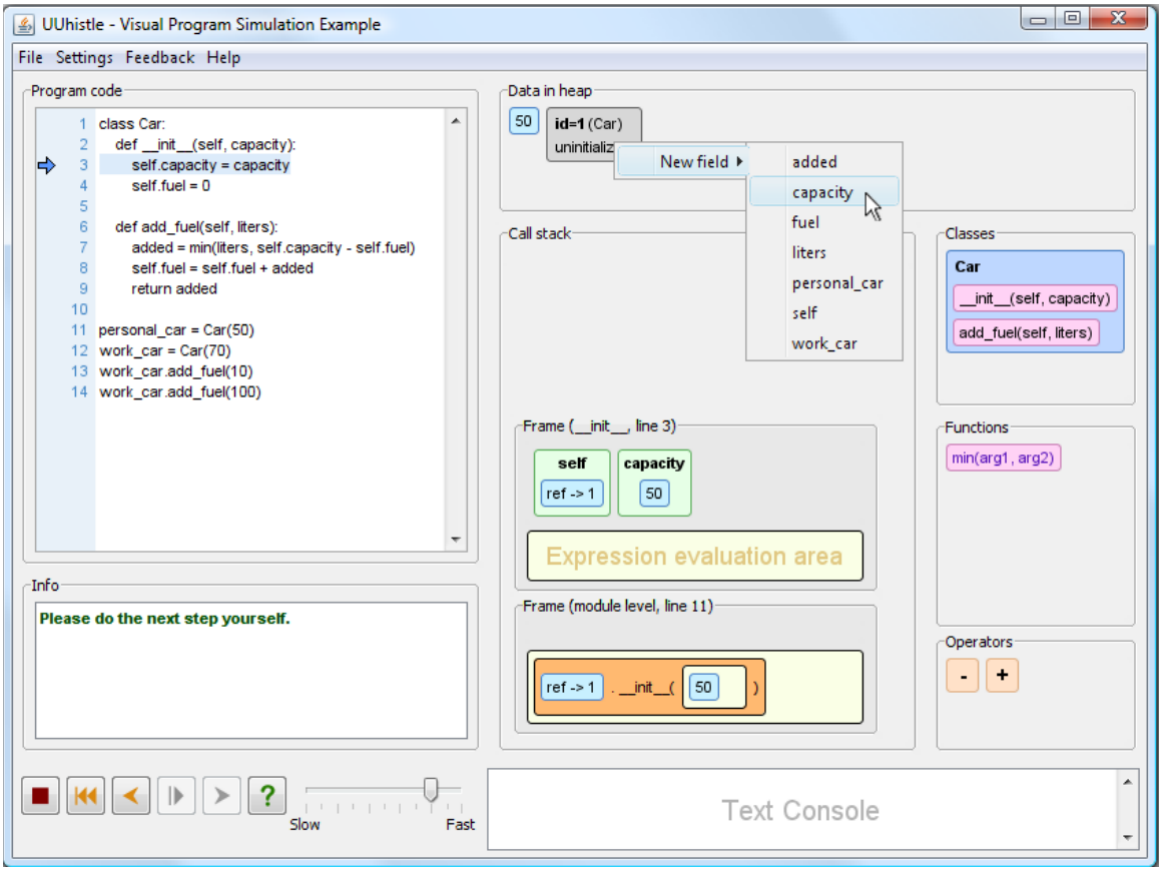



FIGURE 1.6 – Interface de UUhistle[26]

### 1.3.5 Diagramme de structures de contrôle

#### Jgrasp[12]

Jgraps propose plusieurs fonctionnalités intéressantes dans le cadre de la visualisation de programme. L'une d'entre-elles est l'affichage de diagrammes d'activité directement attaché aux lignes de code. Ces diagrammes permettent de repérer les différentes possibilités lorsque l'on rencontre des conditions. Ils affichent également les boucles, classes et fonctions (voir figure 1.8).

### 1.3.6 Vue dynamique des structures de données

#### Jgrasp[12]

Jgrasp (voir figure 1.9) propose des visualisations des différentes structures de données complexes comme les listes, tableaux ou encore arbres binaires. Cette fonctionnalité accessible depuis la fenêtre "Object viewers" s'utilise de pair avec le debugger, grâce auquel l'utilisateur va pouvoir constater les effets de son code sur les structures de données au fur et à mesure que le programme s'exécute.

### 1.3.7 Mise en lumière du flux d'exécution du programme

#### Collece-2.0[20]

Collece-2.0, un outil de collaboration et de visualisation de programme, propose une approche assez différente des softwares cités précédemment : son objectif est centré sur le flux d'exécution d'un programme. Les visualisations reprennent la métaphore de la route avec tous ses composants (rond-point, carrefour, panneaux) afin de représenter les programmes. Un programme prend la forme

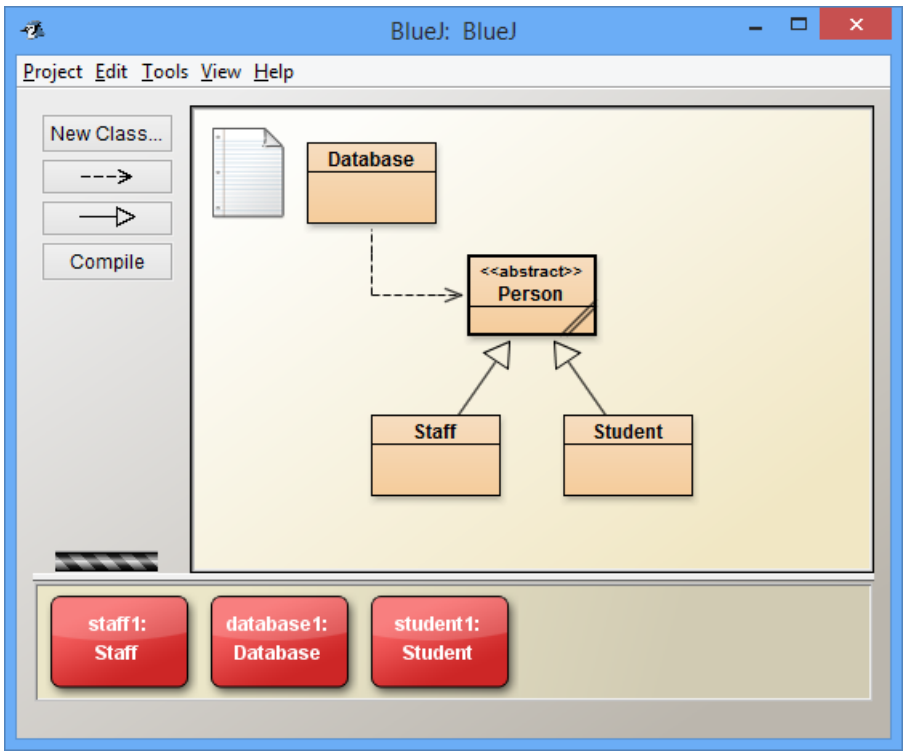



FIGURE 1.7 – Un diagramme de classe ainsi des objets dans BlueJ[28]

```
while (child <= size) {
    if (child < size
            && heap[child].compareTo(heap[child + 1]) < 0) {
        child++;
    }
    if (last.compareTo(heap[child]) >= 0) {
        break;
    }
    // Move child up.
    heap[node] = heap[child];
    node = child;
    child *= 2;
}
```

FIGURE 1.8 – Les structures de contrôles mises en lumière par Jgrasp[12]

d'un parcours par lequel l'utilisateur doit passer. Des ronds points sont utilisés lors des passages dans des boucles et des embranchements sont rencontrés lors des passages par des conditionnelles. Des panneaux permettent d'afficher le code utile comme les conditions des structures de contrôles. Comme il est possible de le constater dans la figure 1.10, l'outil prévoit également des visualisations pour les définitions de fonction, retours de fonction et les expressions. Les étudiants ayant pu tester l'outil lors d'une étude ont en grande partie validé les métaphores utilisées pour les conditionnelles et boucles. Les représentations utilisées pour les fonctions et expressions ont par contre eu moins de succès[20]. Bien que la métaphore routière s'est avérée adaptée pour représenter les structures de contrôle, il semble que ça ne soit pas le cas pour représenter la modularité d'un programme, les évaluations des expressions, etc.

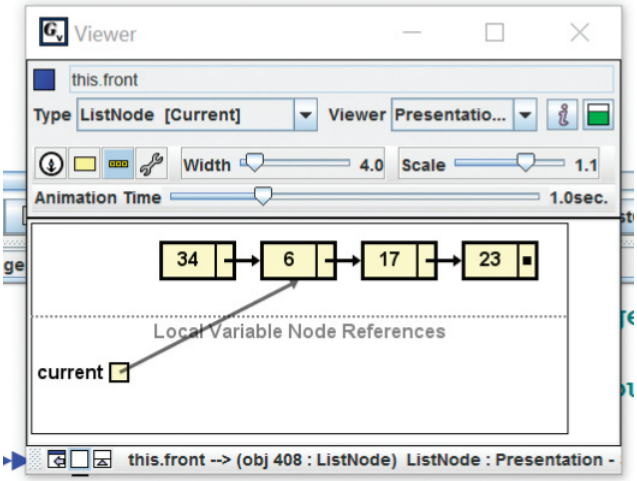



FIGURE 1.9 – Visualisation d'une linkedList en java dans Jgrasp[12]


(a) (b) (c) (d) (e)

FIGURE 1.10 – Les visualisations 3D de l'outil Collece-2.0[5]

### 1.3.8 Les rôles de variables et les métaphores visuelles (PlanAni)

#### Objectifs des rôles

La visualisation peut être utilisée pour rendre certains langages de programmation ainsi que la structure de programmes plus facilement compréhensibles. Plusieurs outils de visualisation de programmes se concentrent sur les concepts et constructions liés aux langages de programmation. Jorma Sajaniemi et Marja Kuittinen[18] pensent que ce type de visualisation n'est pas efficace et qu'il faudrait plutôt utiliser des concepts de construction de programmes de plus haut niveau.

Prenons comme exemple le programme Pascal de la figure 1.11. Dans ce programme, on peut remarquer qu'on utilise deux fois la comparaison :

```
uneVariable > 0
```

À la première occurrence, le programmeur teste si la valeur entrée par l'utilisateur est valide, alors que le deuxième occurrence est utilisée pour voir s'il reste des valeurs à traiter. Si l'on compare ces deux éléments au niveau du langage de programmation, les deux instructions sont équivalentes ; elles retournent toutes les deux "true" si la valeur de la variable testée est supérieure à 0, et "false" dans le cas contraire. Pourtant, quand on regarde à la sémantique de ces deux instructions au niveau du programme, on remarque qu'elles sont toutes les deux utilisées pour une raison radicalement différente. La première est utilisée comme garde qui vérifie que la valeur entrée par l'utilisateur est supérieure à 0 et qui la refuse si ce n'est pas le cas. La deuxième est utilisée pour vérifier une succession descendante de valeurs en attendant le moment ou une valeur sera égale à 0, ce qui arrêtera l'itération.

```
program doubles (input, output);
    var data, count, value: integer;
    begin
        repeat
            write('Enter count: '); readln(data)
    until data > 0;
    count := data;
    while count > 0 do begin
        write('Enter value: '); readln(value);
        writeln('Two times ', value, ' is ',
            2*value);
        count := count - 1
    end
end.
```

FIGURE 1.11 – Un programme pascal[18]

Si les visualisations utilisées pour représenter ces deux instructions sont égales, elles ne donnent aucune information concernant le programme, ce qui n'est pas utile. Les auteurs de l'article [18] ont donc cherché un moyen pour différencier visuellement des instructions en fonction de leur sémantique dans le programme et ont introduit le concept des rôles de variable. Dans l'exemple de la figure 1.11, ce qui permet de différencier la sémantique des deux instructions est la nature de la variable utilisée. La première contient la valeur entrée par l'utilisateur et la deuxième est utilisée comme un compteur qui décroît au fur et mesure.

Le rôle d'une variable caractérise la nature dynamique d'une variable en fonction de la séquence de valeurs successives qu'elle contient et de sa relation avec les autres variables et évènements externes[18]. Un rôle est un concept général qui est souvent rencontré dans les programmes.

Sajaniemi [2002] a trouvé un total de 9 rôles différents qui permet de couvrir 99% des variables utilisées dans les programmes de niveau débutant, en programmation procédurale. Voici la liste :

- *Fixed value* : Une variable dont la valeur ne change pas après son initialisation ou bien une seule fois juste après son initialisation. Par exemple une variable booléenne contenant "true" si le programme est exécuté durant une année bissextile

- *Stepper* : Une variable passant par une succession de valeurs dépendamment de ses valeurs précédentes et potentiellement des valeurs précédentes d'autres variables *steppers, stepper followers et fixed values*. Peut être par exemple un index qui passe par toutes les cases d'un tableau en s'incrémentant ou bien une variable qui alterne entre deux valeurs.

- *Follower* : Une variable qui passe par une succession de valeurs en fonction de la valeur d'une seule variable qui est mise à jour directement après avoir été utilisée pour mettre à jour le *follower*, ou en fonction d'une *fixed value*. Par exemple, le "low index" dans une recherche binaire.

- *Most recent holder* : Une variable contenant la dernière valeur rencontrée lors d'une succession de valeurs, et qui peut être mise à jour directement après avoir rencontré une nouvelle valeur. Par exemple, la dernière lecture en input qui ait eu lieu.

- *Most-Wanted holder* : Une variable contenant la meilleure valeur rencontrée après être passée par une succession de valeurs (quelle que soit la manière de déterminer la meilleure valeur). Par exemple un index vers la plus petite valeur d'un tableau.

- *Gatherer* : Une variable qui accumule des valeurs individuelles, comme par exemple la somme totale des km's parcourus.

- *One way flag* : Une variable pouvant contenir deux valeurs et qui ne peut être modifiée qu'une seule fois, comme par exemple une variable initialisée à "false" et qui passe à "true" à chaque fois qu'une erreur est rencontrée lors de l'exécution du programme.

- *Temporary* : Une variable contenant la valeur d'une autre variable ou input pour une période courte.

Compte tenu de cette classification, un tableau est considéré comme *fixed value* (ou *stepper*, *follower*, *most-recent holder*, *most-wanted holder*, *gatherer*, *one way flag*) si tous ses éléments sont des *fixed values*. Par exemple, un tableau est *gatherer* s'il contient 12 *gatherer* pour calculer les ventes totales de chaque mois d'une année.

Il existe cependant un rôle spécial pour les tableaux :

- *Organizer* : Un tableau qui est seulement utilisé pour réarranger ses propres éléments après avoir été initialisé.

### Les changements de rôles

Dans un programme, il est possible que certaines variables changent de rôle. Cela arrive souvent entre deux boucles. Reprenons la variable "count" du programme en figure 1.11. Cette dernière est un *most recent holder* lors de la première boucle et un *stepper* dans la deuxième.

Il y a deux types de changement de rôle :

1. *Proper* : La valeur finale de la variable dans son premier rôle est utilisée comme valeur initiale pour le rôle suivant.

2. *Sporadic* : La variable est réinitialisée avec une valeur nouvelle lors du changement de rôle.

Selon les analyses de Sajaniemi[2002], seulement 5% des variables sont affectées par un changement de rôle, et 85% de ces changements sont *sporadic*.

### PlanAni : La visualisation de programmes basée sur les rôles de variable

PlanAni[18] est un logiciel d'animation de programmes créé et utilisé à des fins pédagogiques. Dans PlanAni, une métaphore est utilisée pour représenter chacun des neufs rôles cités plus haut, l'objectif final étant que le novice puisse comprendre plus facilement le flux d'exécution du programme en visualisant le rôle des différentes variables via les métaphores et leur animation.

Voici la liste des métaphores choisies pour chaque rôle de variable. Quelques exemples sont repris à la figure 1.13.

- *Fixed value* : Une pierre tombale qui représente le fait que la valeur contenue dans la variable ne changera plus.

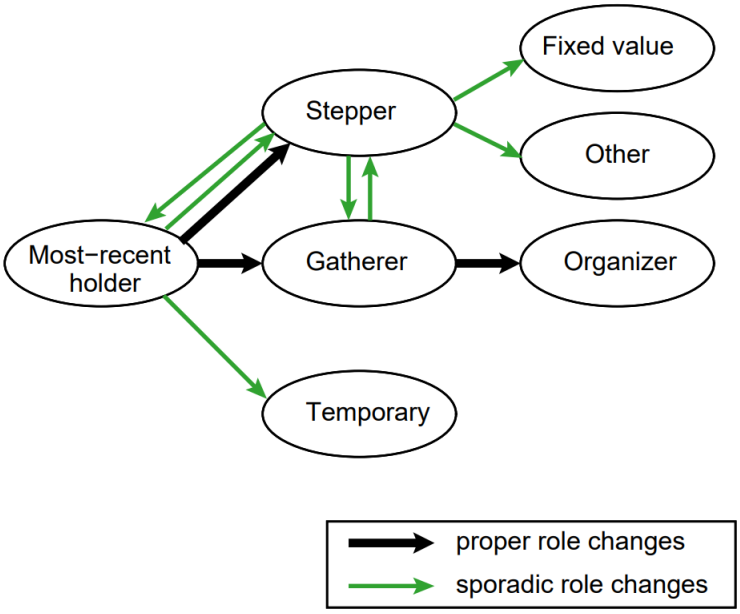



FIGURE 1.12 – Les types de changement de rôle de variable[18]

- *Follower* : Un chien positionné à coté de la variable qu'il suit. Cela rappelle l'idée d'un chien qui suit constamment son maitre.
- *Most-wanted holder* : Deux fleurs de différentes couleurs : Une claire pour la meilleure valeur trouvée à un moment T et une grise pour les valeurs précédentes.
- *Most-recent holder* : Affiche aussi plusieurs valeurs : la valeur actuelle et les précédentes. Cette fois-ci simplement dans des carrés de couleur unique (qui font penser à un tableau).
- *Stepper* : Des empreintes de pas. Montre la valeur actuelle ainsi que les précédentes et futures.
- *Gatherer* : Une boite contenant la valeur courante et les valeurs précédentes.
- *One-way flag* : Une ampoule qui s'éteint quand elle est à "false".
- *Temporary* : Une lampe torche qui éclaire la valeur tant qu'elle est utilisée.
- *Les tableaux* : Contiennent plusieurs fois les métaphores des rôles des cellules. Les éléments d'un *organizer* sont similaires à des *fixed value*(pierre tombale) mais avec des roues pour exprimer une potentielle mobilité.

### Animation des images

PlanAni ne se limite pas à l'affichage des métaphores lorsqu'un rôle de variable est rencontré. Au fur et à mesure que l'exécution du programme avance et que les instructions sont exécutées, PlanAni affiche des animations. Ces animations dépendent des rôles des variables qui sont utilisées dans l'instruction qui est exécutée. L'exemple de la figure 1.14 montre la visualisation utilisée dans le cas des deux comparaisons "uneVariable" $>$ 0" rencontrée dans le programme Pascal de la figure 1.11. Dans le premier cas, la variable "data" est un *Most-recent holder* et l'animation affiche les valeurs que l'utilisateur est en train de rentrer au clavier. Chaque nouvelle valeur entrée est comparée à 0 via une flèche verte ou rouge.

Dans le deuxième cas, la variable "count" est un *stepper* et est donc représentée par des traces


FIGURE 1.13 – Métaphores utilisées par PlanAni

de pas. Les valeurs défilent au fur et à mesure que la variable est décrémentée à chaque tour de boucle. Les valeurs positives sont inscrites en vert et l'on voit les valeurs négatives en rouge arriver.

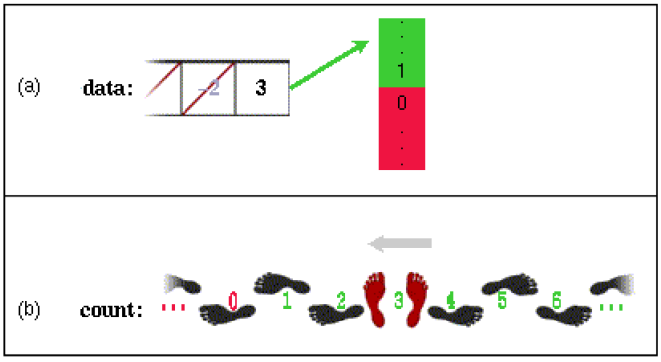



FIGURE 1.14 – L'animation de la comparaison "variable > 0" en fonction du rôle de la variable

### Design de l'application et autres fonctionnalités

La figure 1.15 est une capture d'écran du logiciel PlanAni en fonctionnement. Le code est affiché dans la partie de gauche de l'interface. La ligne qui est en train d'être exécutée est mise en couleur. La visualisation des variables se trouve dans la partie de droite. Un cadre se trouve en bas à droite dans lequel se trouve la partie "input - output". La partie input est représentée avec un morceau de papier et la partie output est un plat. La partie du code à gauche qui est exécutée est reliée à la variable utilisée dans la partie de droite grâce à une flèche.

De manière à éviter que l'utilisateur ne doive trop sauter entre la partie code et la partie visualisation, des pop-ups apparaissent fréquemment pour expliquer ce qu'il se passe dans le programme comme par exemple la création de variables, un changement de rôle, des opérations ou encore des constructions de contrôle telles que des boucles ou conditions.

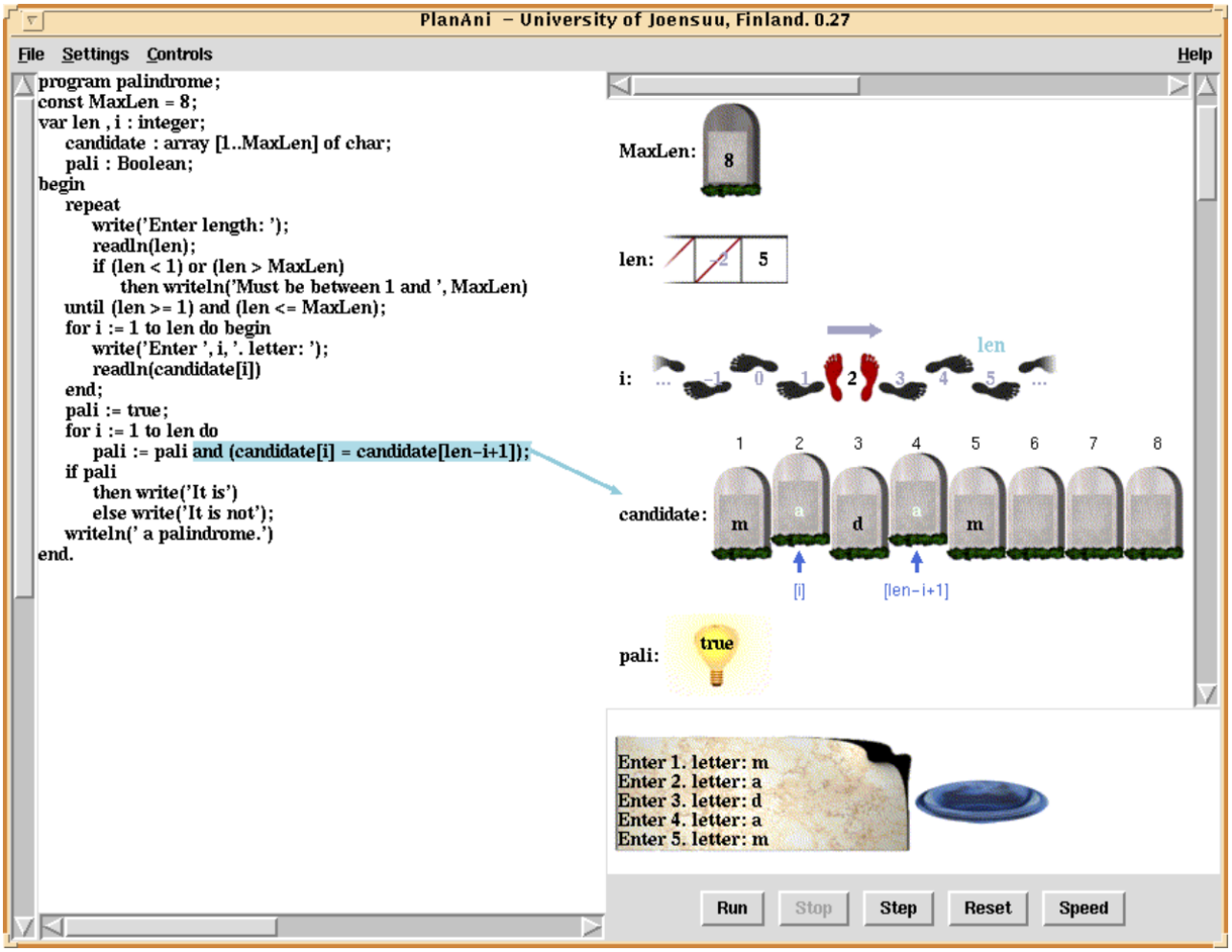



FIGURE 1.15 – Visualisation d'un programme avec PlanAni[18]

## 1.4 Etat de l'art - Conclusion

De nombreuses recherches sur le thème de la visualisation en programmation ont été effectuées et beaucoup sont encore en cours. L'objectif de la plupart des outils cités plus haut est d'aider les novices à apprendre le développement plus facilement, souvent en fournissant à l'étudiant une couche d'abstraction plus haute concernant le code. Les méthodes utilisées pour fournir cette couche d'abstraction sont diverses et tous les systèmes ne visent pas à améliorer les mêmes compétences dans la programmation. Certains permettent à l'étudiant de comprendre plus facilement le flux d'exécution d'un programme tandis que d'autres permettent de visualiser le contenu des variables. Certains ciblent les compétences liées à la programmation orientée objet, d'autres la programmation procédurale, sans parler des différents langages avec lesquels ces systèmes permettent de travailler.

Malgré les avancées sur le sujet, les tests et expériences concernant l'apprentissage de la programmation avec de tels outils ne démontrent pas encore une efficacité notable. Les résultats semblent fortement dépendre de l'implication des étudiants dans les différents exercices et surtout de la ma-

nière dont les outils de visualisation sont utilisés lors des activités d'apprentissage.

Certains outils comme PlanAni nécessitent d'être utilisés dès le début des activités d'apprentissage afin de se montrer utiles. En effet, le concept des rôles de variables doit être intégré directement dans les cours pour que l'étudiant puisse réellement se référer à ces métaphores à chaque fois qu'il programme ou qu'il doit comprendre du code.

Dans le cas de ce travail pour l'Unamur, l'objectif serait de s'approcher d'un outil qui pourrait être utilisable dans le cadre des cours qui se donnent actuellement, et ce sans devoir modifier toute la méthodologie utilisée. Se baser sur des concepts tels que les rôles de variables ne serait pas pertinent étant donné que ces idées ne sont pas exploitées lors du cursus des étudiants.

Le chapitre suivant de ce travail reprendra la démarche et les étapes utilisées afin d'imaginer des visualisations palliant à ces problèmes de compréhension des types avec le langage Python. Pour aller plus loin, d'autres visualisations seront aussi proposées pour supporter les étudiants dans la compréhension des fonctions et conditions.

# Chapitre 2

# Problématique

Dans cette section seront décrits les différents problèmes auxquels font face les étudiants qui débutent dans la programmation avec Python. Les objectifs ainsi que l'étendue des recherches seront ensuite fixés. Les questions de recherche qui guideront la suite du projet seront finalement définies.

## 2.1 Problèmes

De par leurs observations et expériences réalisées lors des cours de programmation à l'université de Namur, les professeurs et assistants remarquent une certaine difficulté des étudiants à comprendre l'impact des séquences d'instructions sur l'espace mémoire. A cela s'ajoutent des difficultés concernant les types des variables ainsi que le typage dynamique utilisé par le langage Python.

En effet, il s'avère que les étudiants puissent éprouver des difficultés à identifier les types de certaines variables ou expressions. Python étant un langage typé dynamiquement, les étudiants peuvent vouloir réaliser des opérations qui sont en fait incorrectes. En effet, un langage de programmation typé dynamiquement réalise ses vérifications de typage lors de l'exécution du programme, et non lors de sa compilation. Cela veut également dire que les variables ne sont pas explicitement liées à un type lors de l'écriture du code. L'étudiant, par manque de connaissance ou d'expérience peut penser connaître le bon type de certains éléments alors que ce n'est pas le cas. Dans le meilleur des cas, le programme enverra une erreur expliquant à l'étudiant sa faute. Dans le pire des cas, le programme fonctionnera mais renverra un résultat différent que celui espéré par le novice et il ne comprendra pas forcément son erreur.

Une autre difficulté des étudiants que le projet devrait aider à supporter est le concept de fonction. Ce dernier est effectivement très souvent mal compris par les novices. Beaucoup de misconceptions y sont liées. L'objectif de la visualisation sera de faire comprendre aux étudiants que le code d'une fonction qui vient d'être définie n'est pas exécuté directement. Une fois cette fonction définie, elle est disponible et peut être appelée dès que le développeur le souhaite (jamais, une ou plusieurs fois). Lors de l'appel d'une fonction, il faut respecter le nombre de paramètres qu'elle prend en entrée. Si la fonction se termine par un return, il faut aussi pouvoir récupérer la sortie de la fonction et la stocker quelque part (ou bien l'utiliser directement). Comme pour le reste, il faudra faire attention aux types des paramètres et du return. Python étant un langage dynamique, ce n'est pas défini de manière statique mais il faudra savoir ce que l'on fait afin que la fonction soit utilisée correctement et fasse ce que l'on attend d'elle. Lors de la création d'une fonction, on créé une scope(étendue) dont chaque variable créée fera partie. Il s'agit d'un contexte isolé du reste du programme. Il sera impossible d'utiliser une variable créée dans une certaine fonction depuis l'extérieur de celle-ci. Pour récupérer les résultats d'une fonction, il faudra utiliser un "return". Cependant, l'inverse est possible : on peut utiliser dans une fonction des variables globales déclarées dans le programme principal.

Lors de discussions avec les promoteurs de ce travail, il a aussi été dit que les conditionnelles posaient des problèmes aux débutants. En effet, lors d'une relecture d'un code contenant des conditionnelles, certains étudiants semblent avoir tendance à vouloir exécuter tout le code qu'elle comprend. Lors d'un if/else par exemple, l'étudiant voudrait exécuter le contenu du if mais également du else. La condition booléenne devient alors inutile. La visualisation permettra à l'utilisateur de voir si la garde d'une condition est évaluée à True ou à False. Il verra également quel bloc de code doit être exécuté et quel bloc de code doit être passé.

Au final, pour un débutant, la programmation reste un tas de commandes et de chiffres avec lequel il est difficile d'être à l'aise. Lors des recherches sur les différents outils de visualisation existants, aucun de ceux-ci vise à améliorer la visibilité des types, fonctions et conditionnelles dans le code n'a été trouvée, ce pourquoi ces thèmes seront explorés dans ce travail.

## 2.2 Objectif et contraintes du travail

Les recherches effectuées dans le cadre de ce mémoire viseront à imaginer et tester un prototype de visualisation de programme qui pourra être utilisé comme debugger. Ce dernier se placera donc dans la partie gauche de la figure1.1, partie dédiée à la visualisation de programmes. Il se placera plus précisément dans la partie *program animation*.

L'objectif de l'école est de former les étudiants au monde du travail et du développement. L'outil devant être développé doit donc supporter les activités d'apprentissage dans cette tâche. Pour cette raison, utiliser les concepts de *visual programming* ne semble pas être adapté. En effet, les étudiants sont directement confrontés au langage Python qui est un langage textuel. Cela est une volonté de l'école car Python est un langage considéré comme très adapté à l'apprentissage de la programmation. Il est également largement utilisé dans le monde professionnel, comme outil de développement back-end ou dans le domaine de la data-science, pour citer deux exemples.

Le but recherché concernant l'outil à développer est qu'il puisse s'utiliser en parallèle avec un éditeur de code, de manière à soutenir l'étudiant dans ses activités de programmation en ajoutant des informations visuelles. L'étudiant pourrait l'utiliser comme un debugger classique, auquel s'ajouterait les visualisations. Il aurait donc plus facile à comprendre le fonctionnement de son programme. L'affichage de certaines visualisations en temps réel lors de l'écriture du code (et non lors de l'exécution comme pour le debugger) est également une possibilité qui sera envisagée lors de discussions avec le public cible ; les étudiants.

Ce prototype doit être adapté à l'enseignement actuel de la programmation à l'Unamur afin de potentiellement y être utilisé plus tard. Les compétences que devra renforcer cet outil sont les compétences visées au premier quadrimestre de la première année d'apprentissage de la programmation. A l'Unamur, l'apprentissage de la programmation a lieu dans différentes sections :

- Le bachelier en informatique
- Le bachelier en ingénieur de gestion option "technologies et management de l'information"
- Le bachelier en mathématique

Les compétences devant être acquises dans le cours d'introduction à la programmation de la première année du bachelier d'informatique sont les suivantes :

- manipuler des variables et des valeurs
- abstraire du code à l'aide de fonctions
- écrire une spécification de fonction et documenter son code
- concevoir des structures conditionnelles et les conditions associées
- utiliser des boucles for/while et choisir entre ces deux constructions
- choisir et manipuler une structure de données adaptée au problème
- créer, utiliser et manipuler des fichiers, notamment avec Pickle
- écrire un algorithme simple (minimum, tri, etc.) et caractériser sa complexité
- concevoir un algorithme récursif

Les concepts ciblés dans la problématique correspondent bien aux compétences devant être acquises durant ce cours. Le public cible de cette étude sera donc les étudiants de première année.

Premièrement, la visualisation devra permettre aux étudiants de remarquer les types des différentes variables et expressions et cela n'importe ou dans le code. Les étudiants pourront ainsi constater s'ils se trompent ou non dans les différentes instructions qu'ils utilisent. Ils pourront par exemple vérifier si les opérateurs utilisés correspondent bien aux types des expressions et si les types des paramètres passés à une fonction correspondent bien avec le code de cette dernière

L'objectif de la visualisation concernant les fonctions sera de faire comprendre aux étudiants que le code d'une fonction qui vient d'être définie n'est pas exécuté directement. Une fois cette fonction définie, elle est disponible et peut être appelée dès que le développeur le souhaite (jamais, une ou plusieurs fois). Lors de l'appel d'une fonction, il faut respecter le nombre de paramètres qu'elle prend en entrée. Si la fonction se termine par un return, il faut aussi pouvoir récupérer la sortie de la fonction et la stocker quelque part (ou bien l'utiliser directement). Comme pour le reste, il faudra faire attention aux types des paramètres et du return. Python étant un langage dynamique, ce n'est pas défini de manière statique mais il faudra savoir ce que l'on fait afin que la fonction soit utilisée correctement et fasse ce que l'on attend d'elle. Lors de la création d'une fonction, on créé une scope(étendue) dont chaque variable créée fera partie. Il s'agit d'un contexte isolé du reste du programme. Il sera impossible d'utiliser une variable créée dans une certaine fonction depuis l'extérieur de celle-ci. Pour récupérer les résultats d'une fonction, il faudra utiliser un "return". Cependant, l'inverse est possible : on peut utiliser dans une fonction des variables globales déclarées dans le programme principal.

Concernant les conditionnelles, la visualisation permettra de montrer aux étudiants si une garde est évaluée à True ou à False et de montrer quel morceau de code doit être exécuté ou non.

Il est également important de noter que les concepts de la programmation orientée objet ne sont pas vus au cours. Les visualisations seront donc adaptées à de la programmation procédurale.

## 2.3 Questions de recherche

Ces problématiques et difficultés des étudiants nous amènent vers différentes questions de recherche, dont la plus large est :

1. **Comment la visualisation de programme peut t-elle assister les étudiants de première année dans la compréhension des différents concepts du développement avec Python ?**

   Cette question est divisée en différentes parties en fonction des thèmes qui seront abordés :

   (a) **Comment rendre visibles les types de variables et expressions avec la visualisation ?**

      i. **Les visualisations et couleurs choisies permettent t-elle aux étudiants de reconnaître les types des différents éléments d'un programme ?**

   (b) **Comment rendre visuels les concepts liés aux fonctions et conditionnelles ?**

      i. **Les visualisations de fonctions et conditionnelles choisies permettent t-elles aux étudiants de comprendre le fonctionnement de ces concepts ?**

# Chapitre 3

# Méthodologie

Dans ce chapitre seront décrites les différentes phases de la recherche, ainsi que le déroulement des deux tests utilisateur réalisés à l'université de Namur.

## 3.1 Étapes suivies

Le travail de recherche va se diviser en différentes étapes. Une fois avoir trouvé certaines problématiques à l'aide de questions de recherche et défini l'étendue du travail, une première version des visualisations va être proposée. Ces visualisations concernent les concepts de programmation discutés dans la problématique, c'est à dire les types, fonctions et conditionnelles. Seule la partie IHM de l'outil va être imaginé dans le cadre du travail.

Lorsque la première version des visualisations est établie, un test utilisateur va être mis sur pied. L'objectif étant que l'outil serve dans les cours de l'Unamur, le public visé lors du test sera les étudiants de BAC1. Afin de maximiser les chances d'avoir des volontaires, une visite lors du cours de projet de programmation sera effectuée. Le projet sera expliqué afin d'avoir des étudiants motivés à réaliser l'interview. Durant un laboratoire de 1h30, 3 ou 4 étudiants peuvent potentiellement être interrogés.

Ce test sera réalisé avant les vacances de Pâques afin de ne pas prendre de retard. Les étudiants ne seront ainsi pas encore trop avancés dans leur projet. L'objectif du test sera d'évaluer le niveau des étudiants concernant les thèmes abordés et de tester l'efficacité des visualisations sur des exemples de programmes concrets. Les étudiants devront réaliser des exercices et utiliser l'outil en expliquant leurs expériences. Ce test est décrit dans les détails dans la section 3.2.

Les résultats du test seront analysés selon une étude qualitative. Les entretiens seront retranscrits et le processus de codage sera effectué afin de catégoriser les fragments de paroles jugés intéressants. Le codage se fera selon les méthodes des codages ouverts, axiaux et sélectifs. Les résultats peuvent être retrouvés à la section 4.2.

Ces analyses permettront d'améliorer le prototype et ses visualisations. Une nouvelle version des visualisations sera créée. Certaines nouvelles fonctionnalités seront également intégrées.

Un deuxième et dernier test utilisateur va ensuite être créé. Son objectif sera de valider ou non les changements apportés au premier test. Les visualisations s'appliqueront sur des programmes un petit peu plus grands et complexes afin de mieux devoir répartir et penser les différents éléments faisant partie de l'interface. Le test sera raccourci afin d'avoir la possibilité de faire passer plus

d'étudiants. La description détaillée de ce test se trouve à la section 3.3.

Les résultats seront analysés de la même manière que pour le premier test. Les résultats de ce test sont décrits à la section 4.4. Si nécessaire, d'autres modifications seront apportées au prototype. Les résultats seront ensuite discutés au chapitre 5.

## 3.2 Premier test utilisateur

Afin de tester le prototype d'outil avec le public cible concerné et pouvoir l'améliorer par la suite, un test utilisateur a été réalisé avec trois étudiants de l'Unamur. Des étudiants de première année, un à un, ont pu faire face à différents exercices/programmes utilisant les visualisations réalisées dans le cadre de ce travail. L'objectif était d'observer les étudiants pendant qu'ils utilisaient l'outil afin de tester son efficacité. L'avis de chaque étudiant a également été enregistré. Le type d'entretien choisi pour cette tâche a été l'entretien semi-directif. Les étudiants ont pu donner leurs impressions en étant guidés dans cette démarche. L'entretien dure entre 20 et 30 minutes et se fait de manière individuelle. Chaque entretien a été enregistré en utilisant une application dictaphone sur smartphone.

Le test a été réalisé au milieu du deuxième quadrimestre de la première année ou les étudiants ont des cours de programmation. Les visualisations étant plutôt destinées à des étudiants du premier quadrimestre, ceux du deuxième risquent d'être à un stade plus avancé et donc de moins avoir besoin de ce genre d'outils pour comprendre les exercices donnés.

La suite de cette section comprendra des explications concernant le scénario de test ainsi que des différents exercices utilisés.

### 3.2.1 Scénario de test

Le test utilisateur est divisé en différentes parties :

1. Un exercice est donné à l'étudiant. Celui ci doit le résoudre en inscrivant les réponses dans un tableau. Il s'agit de plusieurs assignations basiques et l'objectif est de donner le type et le contenu de chaque variable. Cet exercice vise à évaluer les compétences de l'étudiant et identifier certains concepts qu'il n'aurait pas bien compris. L'étudiant répond aux questions oralement et par écrit. Le superviseur ne lui donne aucunes informations concernant l'exercice et le laisse répondre. Si l'étudiant ne connaît pas la réponse, il laisse la case du tableau vide.

2. Un exercice très semblable est donné à l'étudiant. Cette fois-ci, les visualisations des types de données se trouvent directement sur le code. Une légende est bien sur aussi fournie. L'étudiant répond aux questions seul comme pour le premier exercice. Une fois que l'étudiant a terminé, le superviseur lui demande si les visualisations l'ont aidé et s'il peut aussi corriger le premier exercice en les utilisant. L'étudiant vérifie ses réponses. Une correction avec le superviseur a lieu et ce dernier pose des questions pour évaluer la compréhension des visualisations par l'étudiant. L'objectif de cet exercice est de familiariser le participant aux visualisations et à la légende. Toutes les visualisations se trouvaient directement sur toutes les lignes de code. L'outil ne s'utilise donc pas encore comme un debugger.

3. Le superviseur donne à l'étudiant un paquet de feuilles sur lesquelles se trouve un petit programme. Le superviseur explique qu'il s'agit d'un programme qu'il va pouvoir exécuter ligne par ligne (en mode debugger) en faisant défiler les feuilles une à une. Chaque feuille représente donc les visualisations lors de l'exécution d'une ligne de code en particulier. L'étudiant

commence par regarder la première feuille (sur laquelle se trouve juste le programme qui n'a pas encore commencé à être exécuté, il n'y a donc pas de visualisations dessus) et essaie de comprendre ce que fait le programme. Il l'explique au superviseur qui note s'il y a déjà des incompréhensions. Ensuite, l'étudiant fait défiler les feuilles une à une et explique les visualisations qu'il voit. Il suit l'exécution du programme jusqu'à la dernière ligne et explique si les visualisations lui ont permis de repérer des potentielles erreurs de compréhension. Le superviseur lui demande ensuite ce qu'il a pensé des visualisations pour vérifier si elles sont claires aux yeux du participant.

4. Le superviseur répète l'étape 3 avec deux nouveaux programmes à exécuter ligne par ligne. Ces programmes contiennent à chaque fois des nouveaux concepts et des nouvelles visualisations par rapport aux précédents.

5. Une fois tous les programmes parcourus, le superviseur demande à l'étudiant ce qu'il a pensé globalement de toutes les visualisations qu'il a rencontrées. L'étudiant est libre de dire ce qui l'a perturbé et ce qu'il a trouvé positif. Il parle aussi des potentielles améliorations qu'il aurait concernant le prototype. Pour terminer, le superviseur demande si l'étudiant trouverait intéressant d'avoir ces visualisations directement lors de l'écriture de code dans un IDE (et donc avant exécution).

### 3.2.2 Exercice 1

Le premier exercice(figure 3.1) comprend une série d'assignations diverses. Il ne s'agit pas encore d'un programme à proprement parler. Il est demandé à l'étudiant de remplir le tableau avec le type et le contenu de chaque variable. Aucune visualisation se trouve sur les lignes de code, l'objectif étant de tester la compréhension de l'étudiant pour ces différents exercices. La difficulté de l'exercice est de trouver le type des différentes variables. En effet, pas mal de cas de figure sont explorés :

- Utilisation de la fonction input() à plusieurs reprises avec plusieurs entrées différentes de la part de l'utilisateur.
- Le résultat de la fonction input() casté en Int
- Expression booléenne
- Expression booléenne entre guillemets
- Opération arithmétique entre Float et Int
- Opération incorrecte entre types qui ne correspondent pas (Int et Str)
- Opération arithmétique entre expression booléenne et Int
- Création d'un Tuple avec un seul élément
- Utilisation de la fonction __contains__ sur le Tuple
- Concaténation de Str

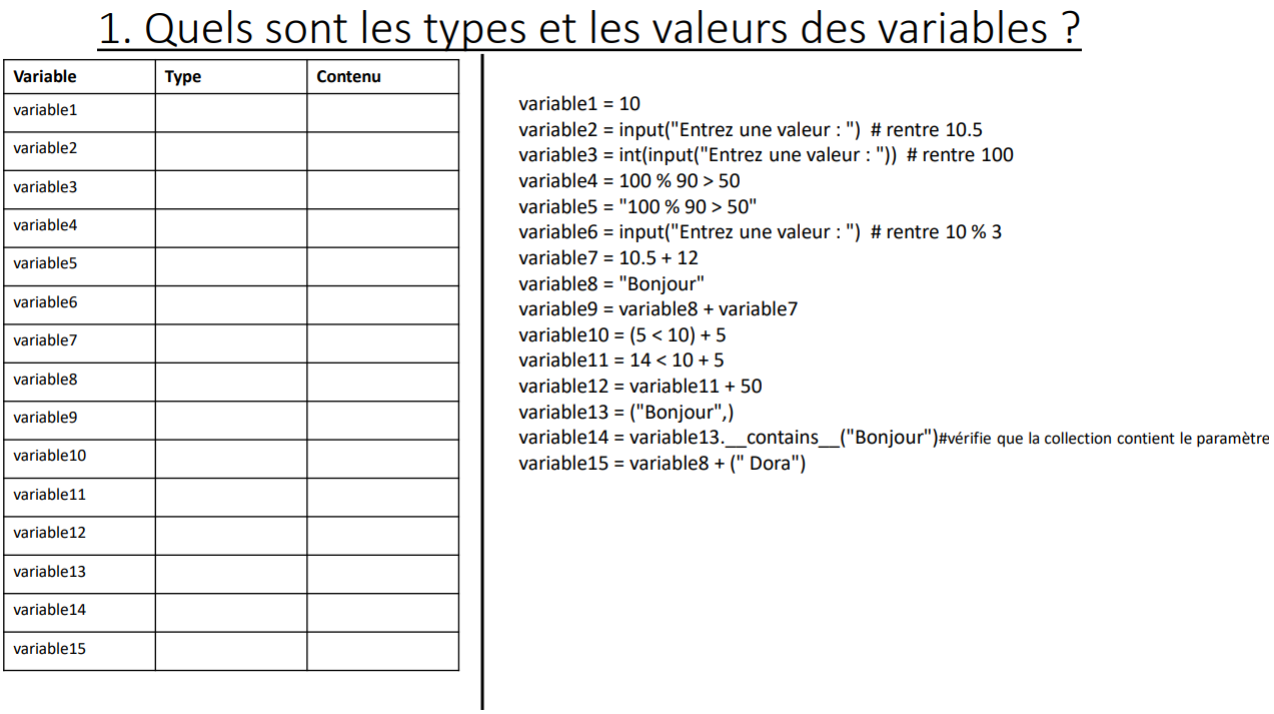



FIGURE 3.1 – Le premier exercice du test utilisateur

### 3.2.3 Exercice 2

Le deuxième exercice(figure 3.2) est très semblable au premier (suite d'assignations testant plusieurs cas de figure). Cette fois-ci, les visualisations légendées se trouvent directement sur toutes les lignes de code. L'étudiant peut donc s'y référer pour remplir le tableau.

### 3.2.4 Exercice 3

Cet exercice(figure 3.3) comprend le premier programme que l'étudiant va pouvoir exécuter ligne par ligne en faisant défiler les étapes. Le programme sert à déterminer le maximum entre trois nombres rentrés par l'utilisateur via la fonction input(). Le design de l'outil comprend ici toutes les parties décrites à la section 4.1.1. L'exercice teste donc une fois de plus la compréhension de l'outil concernant les types. Il y a en effet plusieurs assignations. La variable nommée num3, contrairement aux deux autres, n'est pas castée en Int une fois récupérée via la fonction input(). Il s'agit d'un piège qui sera mis en lumière avec les couleurs liées au type.

Le programme comprend également plusieurs conditionnelles. L'étudiant va donc faire face pour la première fois aux visualisations créées pour ces dernières.

Le programme se termine avec une erreur lors de la comparaison d'un Str (num3), avec un Int(max). Si l'étudiant a bien suivi les visualisations, il devrait pouvoir remarquer qu'on essaie de comparer deux types non compatibles.

### 3.2.5 Exercice 4

Le quatrième exercice(figure 3.4) et deuxième programme sert à additionner deux nombres via une fonction. Le premier terme est une variable globale et le deuxième terme sera passé en entrée à la fonction directement. L'exercice est donc basé sur les visualisations propres aux fonctions.

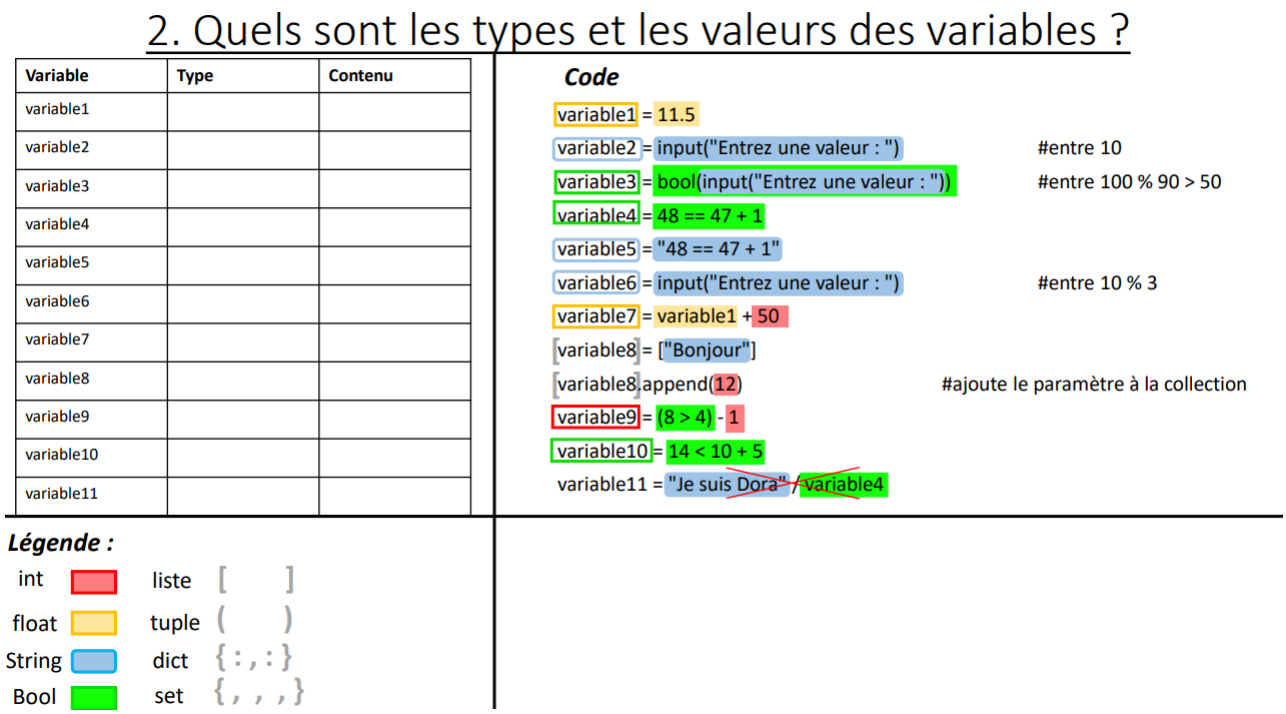



FIGURE 3.2 – Le deuxième exercice du test utilisateur

Un piège se trouve à l'avant dernière ligne du programme ou l'on essaie d'additionner(pour concaténer) un string avec le résultat de la fonction, qui est un Int. En mettant le tout entre parenthèses, et en remplaçant le '+' par une virgule, cela fonctionnerait. Cela peut donc prêter à confusion pour les étudiants.

### 3.2.6 Exercice 5

Le dernier exercice(figure3.5) est un autre programme reprenant une fois toutes les visualisations. Le programme comprend en effet des assignations et opérations, une fonction et une conditionnelle. Il s'agit également du seul programme qui est totalement fonctionnel. Cela est volontaire afin que les étudiants essayent de chercher une erreur qui n'existe pas.

Les informations recueillies pendant les tests utilisateur étant majoritairement orales et basées sur des observations, l'interprétation des résultats se fera principalement de manière qualitative. Les expériences des étudiants lors de la réalisation des différents exercices vont pouvoir être décortiquées afin de comprendre :

- Quelles sont les difficultés de chaque étudiant face à la matière rencontrée
- Quelles sont les réactions (positives, négatives, incompréhension...) des étudiants face aux visualisations
- A quel moment l'étudiant a été aidé/induit en erreur par les différentes visualisations lors de la lecture de programmes
- Quel est l'avis des étudiants concernant les différentes visualisations (type, conditionnelle, fonction...) et sur le prototype d'outil en général et sa potentielle utilité
- Les idées que les étudiants auraient concernant les fonctionnalités de l'outil.

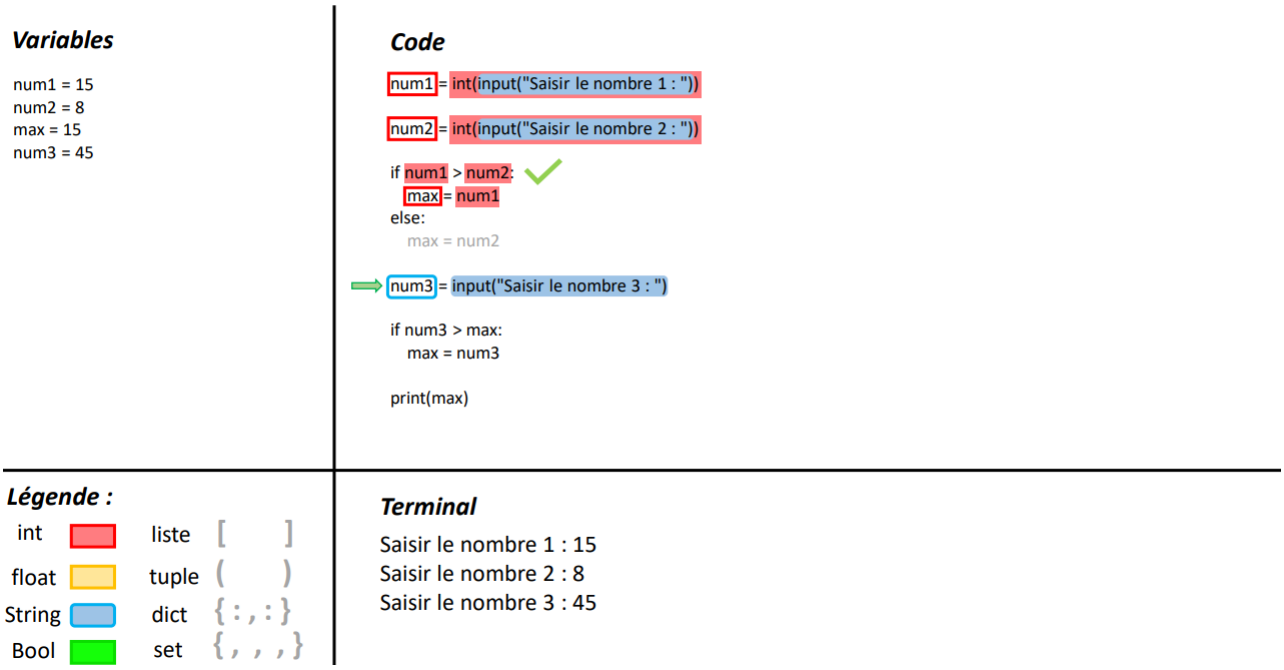



FIGURE 3.3 – Le troisième exercice du test utilisateur

Les réponses des étudiants aux deux premiers exercices (type et contenu des variables) ont été récoltées afin de pouvoir appuyer les analyses. Utilisées en parallèle avec l'enregistrement des étudiants, elles permettront de comprendre pour chaque exercice la maîtrise de la matière par l'étudiant. Il sera possible de trouver les potentielles misconceptions qu'auraient les étudiants par rapport à certains concepts.

### 3.2.7 Codage

Afin de classer et organiser les données recueillies oralement lors des entretiens semi-directifs réalisés avec les étudiants, une étape de codage a été réalisée. Chaque fragment de texte intéressant capturé va être classé selon un thème précis. Ce processus se fera en suivant deux étapes de codage : le codage ouvert et le codage axial.

Les différents thèmes qui ressortent des entretiens et qui seront utilisés pour l'analyse sont les suivants :

- *Types et opérations* : Tous les avis ou expériences liés aux visualisations des types ainsi qu'aux différentes opérations entre expressions/variables
- *Fonction* : Tous les avis ou expériences liés aux visualisations des fonctions
- *Visualisation globale* : Avis généraux des étudiants sur l'outil ou sur le fait de pouvoir utiliser des visualisations
- *Conditionnelle* : Tous les avis ou expériences liés aux visualisations des conditionnelles
- *Temps réel* : L'avis des étudiants sur la question des visualisations qui apparaîtraient en temps réel lors de l'écriture de code.

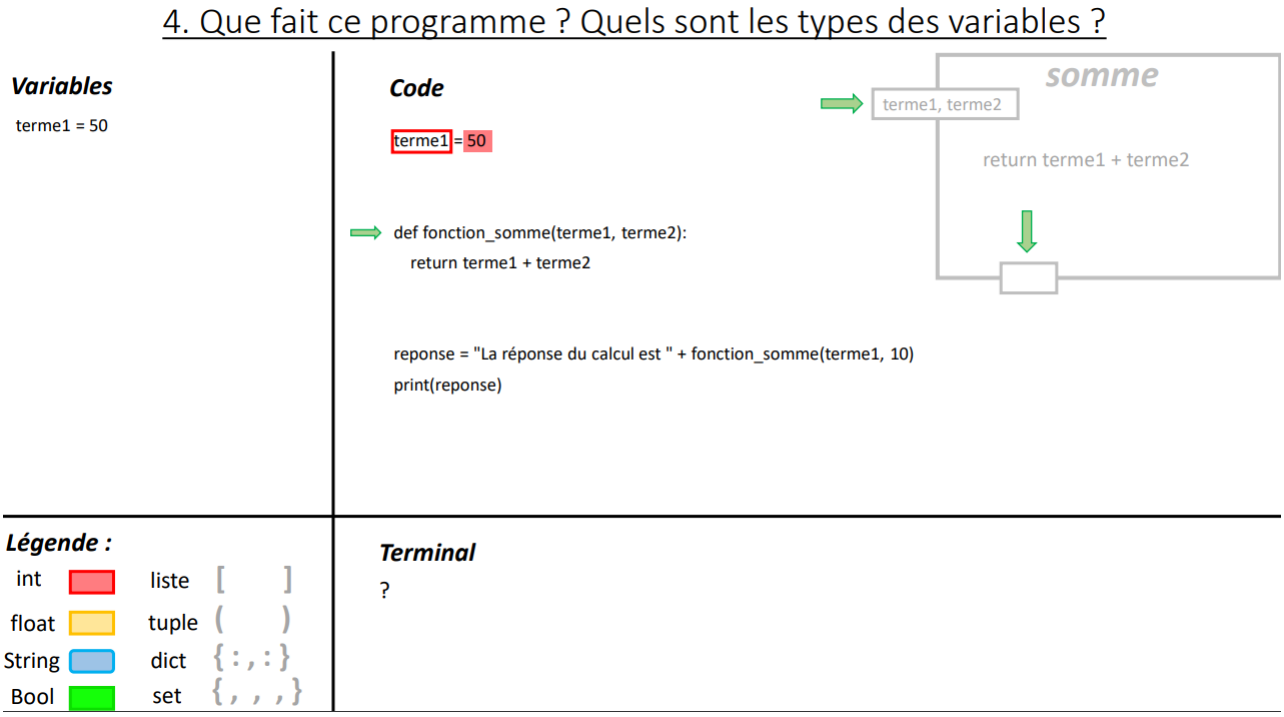



FIGURE 3.4 – Le quatrième exercice du test utilisateur

Les différents thèmes vont être passés en revue dans la section 4.2. Les analyses se feront dans un premier temps pour chaque étudiant séparément.

## 3.3 Deuxième test utilisateur

Afin de pouvoir tester les modifications ajoutées au prototype suite au premier test utilisateur, et compte tenu du fait que seulement trois étudiants aient pu tester l'outil, un deuxième test utilisateur semblait pertinent. Ce test est différent du premier car réalisé plus d'un mois plus tard. Plus court, il sert essentiellement à tester l'outil globalement sur des programmes un peu plus longs que ceux utilisés lors du premier test. Une grande partie du premier test était focalisée sur la question des types alors que le deuxième concerne toutes les visualisations utilisées en parallèle.

Les deux participants au deuxième test sont des étudiants de BAC1 en ingénieur de gestion spécialité management de l'information. Selon les professeurs de l'Unamur, leur intérêt pour la programmation est donc généralement moindre que celui des bacheliers en informatique. Cependant, le test ayant été réalisé à la toute fin du deuxième quadrimestre et après les vacances de Pâques, le niveau des étudiants était nettement supérieur que lors du premier test. En effet, les étudiants ont pu avancer, et pour certains terminer, le projet de programmation (le même qu'en bachelier d'informatique). Ce projet consiste au développement d'un jeu de plateau nécessitant la maîtrise de tous les concepts de programmation en Python appris au premier quadrimestre.

La suite de ce chapitre comprendra des explications concernant le scénario de test ainsi que des différents exercices utilisés.

### 3.3.1 Scénario de test

Le deuxième test utilisateur se divise en ces parties :

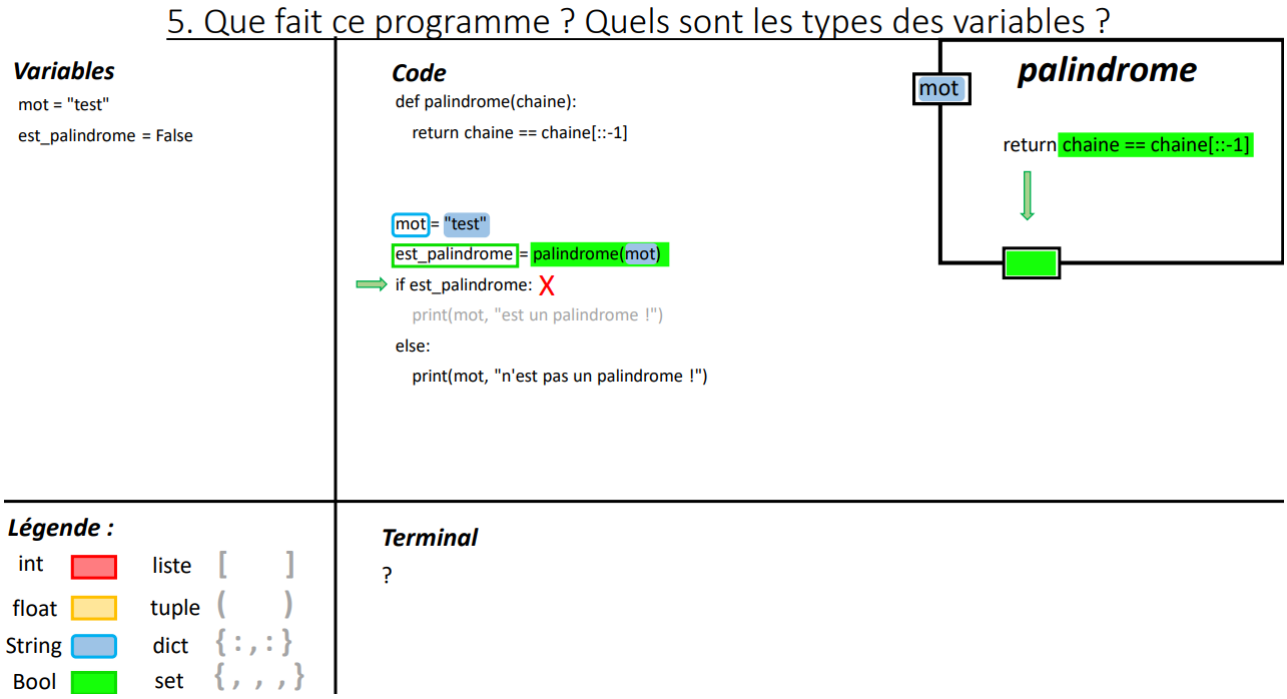



FIGURE 3.5 – Le cinquième exercice du test utilisateur

1. Un programme affiché dans l'outil de visualisation est donné à l'étudiant. Ce programme reprend plusieurs assignations les unes à la suite de l'autre. L'étudiant doit compléter pour chaque variable son type ainsi que sa valeur. Cela permet de vérifier les compétences de l'étudiant quand il fait face à du code brut. Une fois cela terminé, l'étudiant peut faire défiler des pages une à une afin d'exécuter le programme ligne par ligne, en mode debugger. L'affichage des visualisations lui permet de se corriger s'il a fait des erreurs lors de la première étape. Il découvre également les visualisations sur un programme très basique.

2. Un deuxième programme est donné à l'étudiant sous forme d'un paquet de feuilles, comme pour l'exercice précédent. Ce programme est plus complexe et plus long que ceux utilisés auparavant et reprend toutes les visualisations ayant été discutées dans ce travail. L'étudiant, avant de commencer à faire défiler les feuilles, doit essayer de comprendre ce que fait le programme. Une fois qu'il pense comprendre (ou pas), il commence à l'exécuter en expliquant ce qu'il voit.

3. Une petite discussion sur l'avis global de l'étudiant par rapport à l'outil clôture le test.

### 3.3.2 Premier programme

Le premier "programme" est illustré à la figure 3.6. Il s'agit de 8 variables auxquelles on assigne des valeurs de différentes manières. Il reprend les cas de figure intéressants du premier test ainsi que quelques assignations supplémentaires.

Voici les concepts que l'exercice reprend :

- Opérations arithmétiques reprenant des entiers et des Float
- Expression booléenne (avec visualisation détaillée des parties de l'expression)

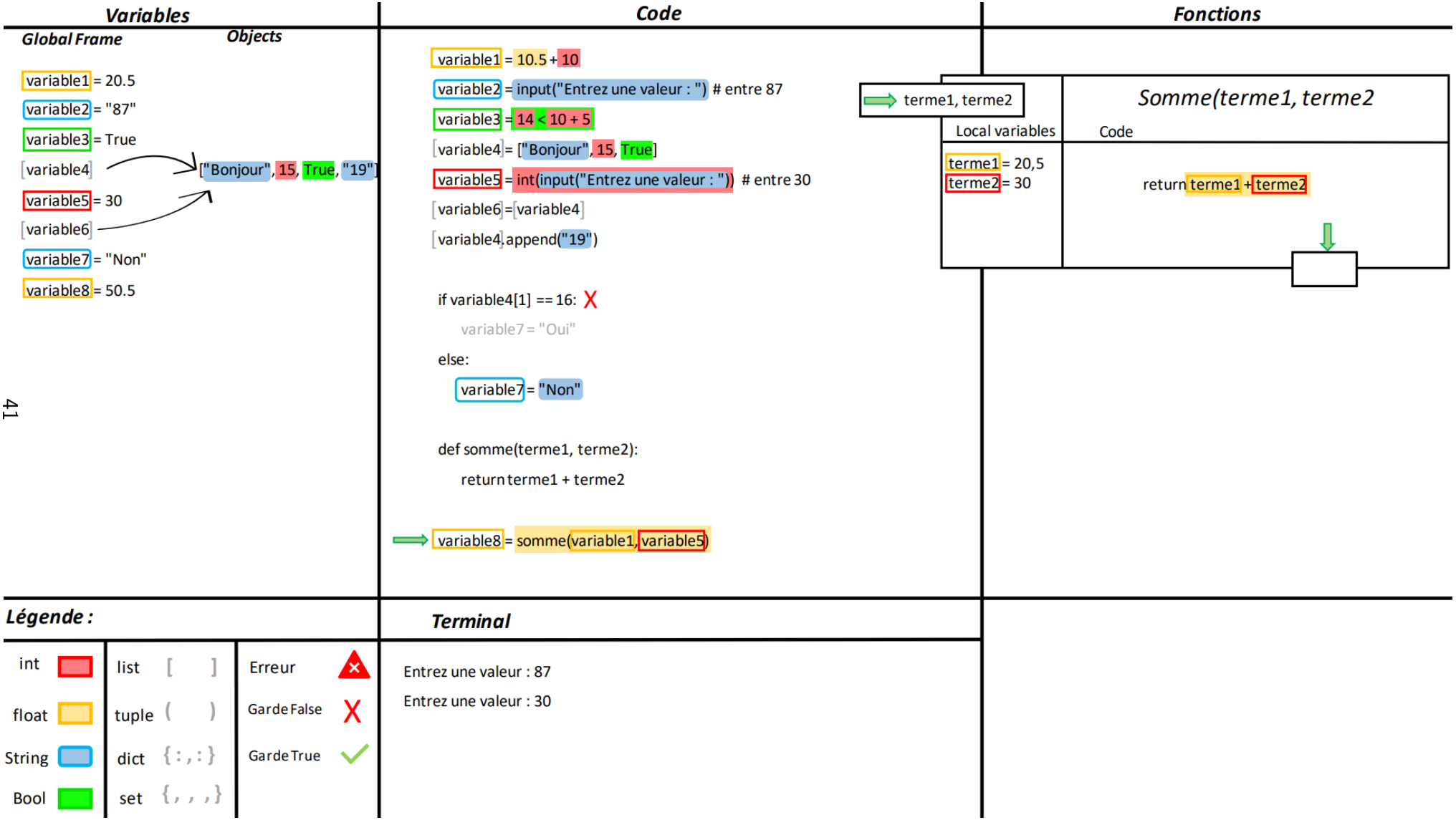



FIGURE 3.6 – Premier programme du deuxième test utilisateur

- Liste avec partage de références
- Cast en Int de la fonction input()
- Condition if/else
- Fonction basique et appel

### 3.3.3 Deuxième programme

Le deuxième programme permet de compter le nombre d'années bissextiles dans une liste contenant des entiers. Pour cela, deux fonctions sont créées :

- *bissextile :* Pour chaque année passée en paramètre, retourne True si elle est bissextile
- *print_number_bissextile_years :* Affiche le nombre d'années bissextiles se trouvant dans une liste passée en paramètre

La deuxième fonction appelle la première pour chaque élément de la liste, ce qui est un élément intéressant à montrer avec la visualisation. Pour chaque élément de la liste sur laquelle on itère, une nouvelle instance de la fonction Bissextile apparaît avec comme variable locale l'année correspondante.

Avant l'appel de la deuxième fonction, deux listes sont créées. On ajoute la deuxième à la première via la fonction append(). Le but ici est de piéger les étudiants qui pourraient croire que la première liste serait juste étendue avec les éléments de la deuxième, ce qui n'est pas le cas. Le dernier élément de la première liste sera de type list et pointera donc vers la deuxième liste. Ce comportement est visuellement mis en lumière par la partie Objects dans les contenus de variables où l'on peut clairement constater grâce à une flèche le pointeur reliant les deux listes.

Ce piège dans le code conduit à une erreur. En effet, la fonction bissextile ne pourra pas fonctionner avec une variable de type List et renverra dès lors une erreur. Cette erreur ne se produira qu'une fois arrivé au dernier élément de la première liste dans l'itération. L'objectif était de voir si l'étudiant comprenait le fonctionnement de la fonction append() et constatait son effet avec la visualisation.

Les entretiens étant plus courts et plus généraux que les premiers, l'analyse ne sera pas divisée en thèmes. Les éléments intéressants de chaque entretien seront donnés dans l'ordre chronologique, quel que soit le thème en question (types, fonctions, conditionnelle...). Afin de ne pas répéter plusieurs fois ce qui a déjà été dit lors du premier entretien, seul l'essentiel sera conservé. Pour chaque exercice, la réaction de l'étudiant avant d'avoir accès à la visualisation sera donnée. A cela suivra sa réaction face à la visualisation.

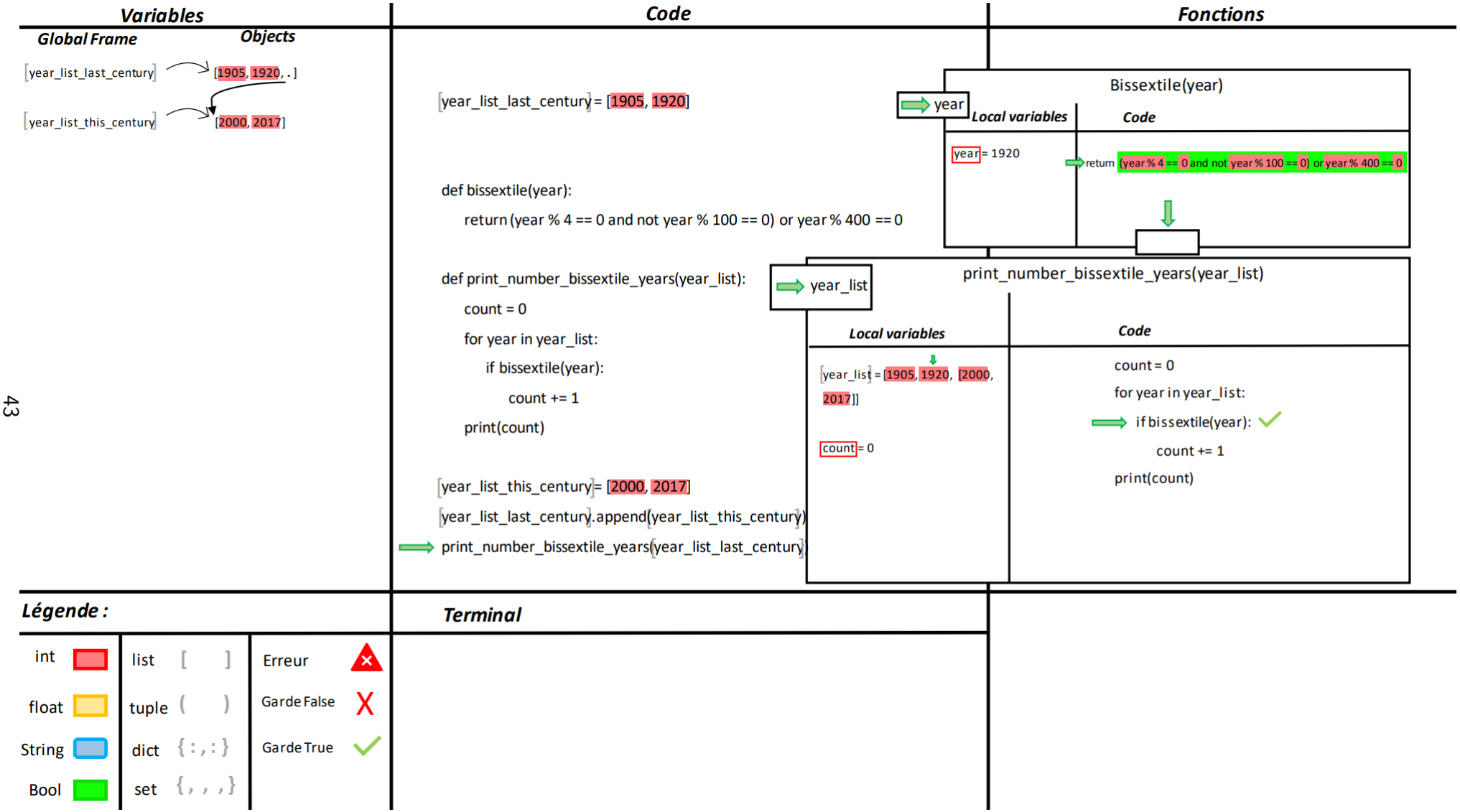



FIGURE 3.7 – Deuxième programme du deuxième test utilisateur

# Chapitre 4

# Résultats

Dans ce chapitre prendront place les différents résultats obtenus au cours du travail. Le premier prototype d'outil va tout d'abord être décrit. Viendront ensuite les résultats du premier test utilisateur auxquels suivra une amélioration du prototype initial. Les résultats du deuxième test utilisateur seront enfin détaillés.

## 4.1 Prototype

La première version des visualisations va être décrite dans les chapitres qui suivent. Il est important de comprendre qu'il ne s'agit que d'une première version qui sera améliorée par la suite grâce aux tests et analyses qui seront effectués. Ces améliorations seront mentionnées plus tard dans ce travail.

### 4.1.1 Aspect général

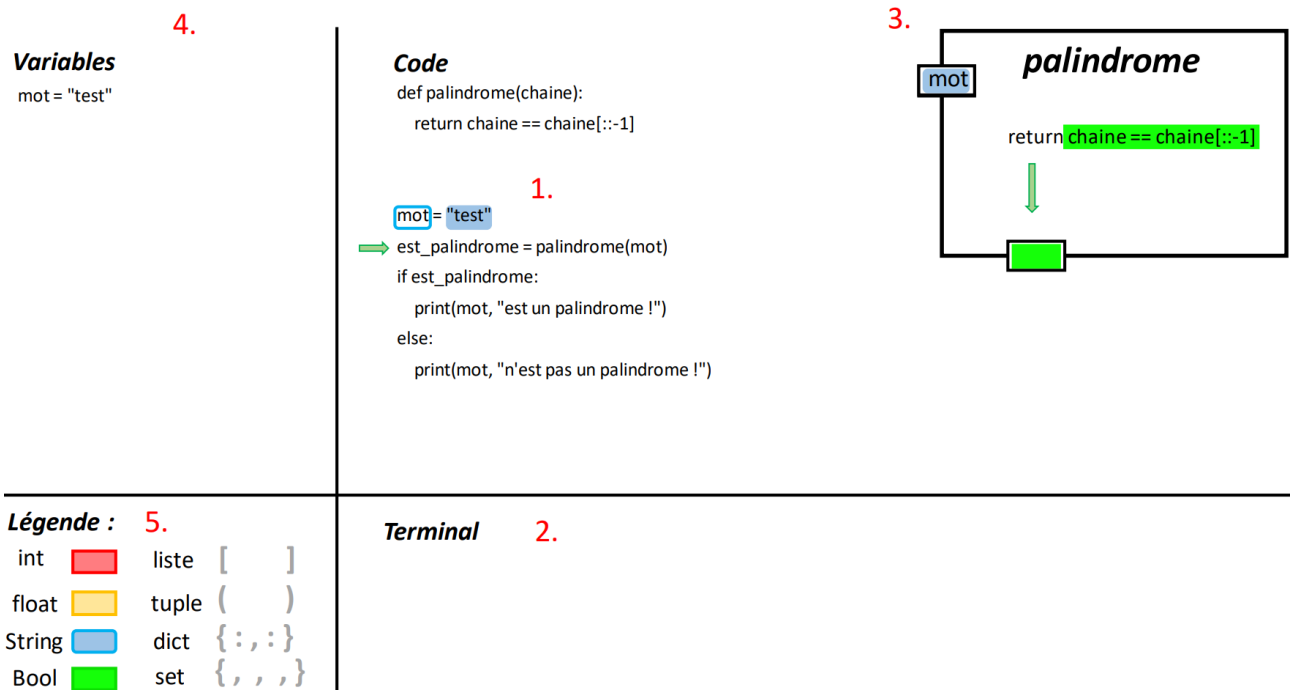



FIGURE 4.1 – Aspect général du prototype

L'objectif final étant que l'outil puisse être intégré directement à un éditeur de code, la démo reprend également la structure basique d'un IDE. La fenêtre est divisée en 4 parties distinctes (numérotées sur la figure4.1) :

1. La partie code ou le développeur écrit son programme

2. Le terminal que l'utilisateur peut utiliser pour interagir avec le programme et lire des potentielles erreurs

3. La partie fonction. Cette partie a été ajoutée dans le cadre des visualisations de manière à voir les différentes fonctions définies à un moment T de l'exécution du programme (plus de détails ci-dessous)

4. La partie variable. Cette partie permet de voir le contenu des variables à chaque étape de l'exécution du programme.

5. La légende à laquelle l'utilisateur de l'outil pourra se référer afin de se rappeler les codes couleurs s'affichant sur le programme

### 4.1.2 Visualisations des types

Les visualisations concernant les types de données se devaient d'être simples et facilement reconnaissables, comme tout type de visualisation de programmes. Il ne s'agit pas de surcharger la mémoire cognitive avec beaucoup d'informations qui pourraient perdre l'utilisateur car cela serait contre productif. Une légende a été établie en utilisant des couleurs et des formes :

#### Types primitifs

Chaque type primitif se voit attribuer une couleur ainsi qu'une forme :

- *Int* : Forme rectangulaire rouge
- *Float* : Forme rectangulaire orange
- *Str* : Forme rectangulaire bleue avec bords arrondis
- *Bool* : Forme rectangulaire verte

Le choix des couleurs n'a pas été fait au hasard. Le Float et sa couleur orange est proche du Int rouge de manière à montrer que les deux types sont "proches" sans pour autant être les mêmes. En effet, les opérations mathématiques effectuées entre ces deux types fonctionnent. Il faudra seulement savoir que le résultat sera une valeur Float.

La couleur bleue a été choisie pour le type Str. L'objectif ici était principalement d'avoir une couleur éloignée des deux premières, à savoir rouge et orange.

Les variables booléennes sont colorées en vert fluo. De nouveau, les booléens n'étant pas régulièrement utilisés dans des expressions contenant d'autres types, la couleur devait à nouveau être différente des autres.

Un détail subtil mais qui a également son importante : Les entiers, Float et booléens ont des bords carrés alors que les Str ont des bords arrondis. Ce choix a été fait pour creuser encore plus la différence entre les types de données. En effet, même s'il n'est peut-être pas courant de rencontrer des opérations entre Bool et entier/Float, ces opérations fonctionnent étant donné qu'un True vaut 1 et un False vaut 0 une fois utilisés dans une opération arithmétique.

Pour conclure, les entiers et Float ont une couleur relativement proche pour montrer que les opérations liant les deux sont possibles. Ils ont, comme les booléens, des bords carrés car il est possible de mélanger les trois dans des expressions mathématiques.

#### Structures de données

Les structures de données sont elles aussi représentées via la visualisation. La variable sera entourée du caractère spécial propre à cette structure de données. Par exemple une variable de type list sera entre crochets et un tuple sera entre parenthèses. Les dict et set seront tous les deux entourés d'entretoises. Pour les différencier, il faudra aller voir le contenu de la variable dans la partie de gauche de l'outil dédié aux valeurs. Par exemple :

```
variable = {1 : "bonjour", 2 : "au revoir"} sera de type dict
variable = {"bonjour", "ça va ?", "au revoir"} sera de type set
```

#### Application des couleurs au code

Les visualisations dans le code respectent une certaine forme. Lors d'assignations (voir figure4.2), le nom de la variable est entourée d'un cadre de couleur non rempli. La partie droite d'une assignation, elle, sera colorée. Qu'il s'agisse de la partie gauche ou de la partie droite, les couleurs et types de formes respectent la légende. Cette manière de faire donne l'idée qu'une variable est un contenant, une sorte de boite dans laquelle on peut stocker des données. La partie droite d'une assignation, une expression, sera le contenu qu'on peut potentiellement stocker dans une variable (une fois cette expression évaluée).

Quand des noms de variable et expressions se trouvent ailleurs que dans des assignations, ils sont colorés et non encadrés (cette idée sera corrigée par la suite pour avoir quelque chose de plus logique). Cela peut comprendre les paramètres qu'on passerait à une fonction, les conditions de structures de contrôle, etc.

Quand il n'est pas encore possible de déterminer le type d'une variable avant son exécution les couleurs ne sont pas appliquées. Prenons l'exemple d'une définition de fonction qui prend plusieurs paramètres et les additionne pour retourner la somme. Les paramètres entrés par l'utilisateur peuvent être de tout type. Cela influencera aussi le type de la valeur de retour de la fonction. Les couleurs seront donc appliquées dans la partie fonction une fois cette dernière appelée par l'utilisateur.

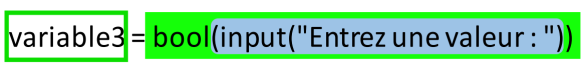



FIGURE 4.2 – Exemple de visualisation sur une assignation

### 4.1.3 Visualisation des fonctions

Comme le montre la figure4.1, les fonctions sont affichées de manière graphique. Au moment de la définition d'une fonction, la visualisation de cette dernière apparaît dans le volet de droite. La

boite est grisée et affiche le nom de la fonction, les paramètres et le retour ainsi que le code de la fonction(figure4.3). Cela montre que la fonction est "créée" et prête à être appelée.

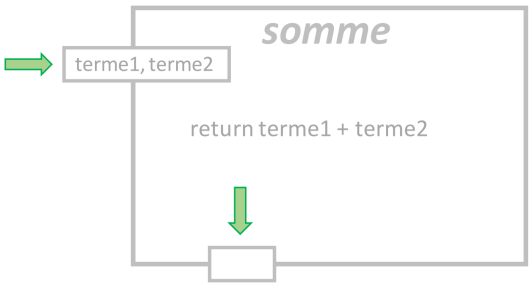



FIGURE 4.3 – La visualisation d'une fonction qui vient d'être définie

Lors de l'appel à une fonction, celle-ci change de couleur et devient noire (comme le reste des éléments). Cela veut dire que la fonction est utilisée et que son code est exécuté. Le type des différents paramètres est affiché grâce aux couleurs de la légende et les visualisations liées aux types apparaissent sur le code de la fonction. Une fois toutes les lignes de code de la fonction exécutées, on voit le type de retour de la valeur retournée (s'il y en a une, voir figure4.4). Quand la ligne de code dans laquelle se trouve l'appel a été exécutée, la fonction perd ses visualisations de couleurs et redevient grisée. Elle est donc prête à être ré-appelée.

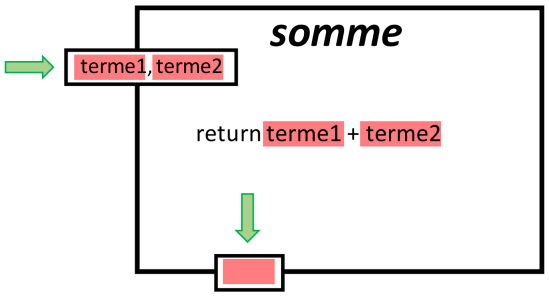



FIGURE 4.4 – La visualisation d'une fonction qui est appelée

### 4.1.4 Visualisation des conditionnelles

La quantité d'informations que la visualisation des conditionnelles doit montrer est limitée. Premièrement, il faut afficher clairement si une garde est évaluée à True ou à False. Colorer la garde en utilisant des couleurs n'est pas une solution car cela prêterait à confusion avec les couleurs des types. Un symbole est donc ajouté à droite de la garde. Lorsque la garde est évaluée à True, un "V" vert apparaît pour indiquer que le "check" est validé. Dans le cas contraire, c'est une croix rouge qui prendra place à coté du if.

Comme il est possible de le constater avec la figure 4.5, les blocs de code devant être exécutés restent en noir foncé alors que les blocs de code ne devant pas l'être deviennent grisés(dans le cas d'un else avec une condition évaluée à True ou directement avec un if évalué à False).

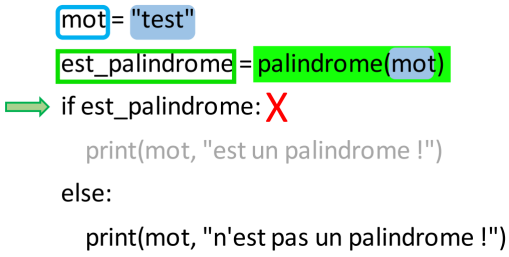



FIGURE 4.5 – Une condition if/else évaluée à False

## 4.2 Première analyse

Dans cette section se trouveront les résultats du premier test utilisateur. Les différentes phrases intéressantes des étudiants prendront place dans l'ordre chronologique des exercices, et seront classés par thèmes de codage. Les différents exercices peuvent être retrouvés à la section 3.2.

### 4.2.1 Types et opérations

#### Étudiant 1

Lors du **premier exercice** (voir section 3.2.2), l'étudiant 1 a montré quelques hésitations concernant les expressions booléennes, les chaînes de caractères, la fonction input() et le casting. En effet, voici ses réactions lors du premier exercice :

**Variable2** : "On met input entrez une valeur, l'utilisateur rentre 10.5, mais comme tout est entre guillemets normalement ça sera ... J'suis pas sûr que si c'est entre guillemets ça reste integer ou bien String." Il ne sait pas si le type de la deuxième variable sera Int ou Str. Cela montre déjà plusieurs incompréhensions. Il pense que le fait que ça soit possiblement Str peut venir du fait que la phrase "Entrez une valeur :" soit entourée de guillemets. Il pense aussi que que le type pourrait être Int car l'utilisateur a rentré une valeur numérique (auquel cas le type serait Float et non integer).

**Variable3**, il dit : "C'est encore input mais cette fois on précise integer. Ça me convint un peu que le précédent était un string". Il comprend ici que comme on cast le retour de la fonction input() en Int, celle-ci renvoie probablement par défaut un Str.

**Variable4**, qui est de type Bool, l'étudiant 1 s'est trompé en mettant le type "int" et le contenu "10". Il ne voit donc pas qu'il s'agit d'une expression booléenne qui vaudra False.

**Variable5**, il dit "C'est la même chose mais c'est entre guillemets donc je mets un string". Il arrive donc à repérer correctement un Str.

**Variable7** : l'étudiant répond incorrectement qu'il s'agit d'un Int avec comme valeur 22,5.

**Variable9** : "Variable 9 c'est var8 + var7, mmmh je ne sais pas si ça va marcher...". Il a correctement repéré l'erreur.

**Variable10** : "Je ne suis pas sûr que ça fonctionne quand tu ajoutes 5 à ça...". Il passe l'exercice car il pense qu'il n'est pas possible de réaliser l'opération. A la fin de l'exercice, après lui avoir expliqué qu'il est possible d'utiliser un Bool dans une telle opération, il dit : "Tu m'as appris quelque chose ! Je ne le savais pas avant de faire celui-la".

Pour le **deuxième exercice** (première rencontre avec les visualisations liées aux types), l'étudiant 1 dit ceci :

**variable1** : "c'est facile à voir du coup ça c'est Float". Il disait lors de l'exercice précédent que 10.5 était un entier alors qu'il voit directement qu'il s'agit d'un Float grâce à la visualisation dans cet exercice ci.

**Variable2** : "Input entre une valeur je vois bien que c'est bleu donc le résultat sera d'office un string en fait". Les visualisations semblent claires pour l'étudiant. Il commence à corriger automatiquement ses réponses du premier exercice.

**Variable3** : "Pour la variable 3 ça sera False car ça vaut 10 et 10 est plus petit que 50 donc ça sera False". Le raisonnement est correct mais il ne sait pas qu'un string vide casté en Bool donnera 0, et 1 sinon.

**Variable4** : "48==47+1 c'est booléen (réfléchit) je me suis trompé, c'est True ou bien false." La visualisation lui a fait remarquer qu'il s'agissait d'une expression booléenne mais il a quand même complété le contenu avec "48 == 47+1" pour la variable3. Il s'est ensuite corrigé. Il a également fait une erreur semblable à la **variable5** pour laquelle il trouvé le bon type (str) mais un contenu erroné. Il a tenté de résoudre l'équation pour stocker la réponse dans la chaîne de caractères.

**Variable9** : "Il est rouge donc je sais que ça sera un int". Il comprend bien les couleurs mais il ne sait pas encore comment réaliser certaines opérations arithmétiques (dont on parlera dans le chapitre suivant).

Après avoir terminé les deux premiers exercices, à la question : "Pour l'instant est-ce que les visualisations t'ont aidé ?", il répond : "Oui avec le type du résultat, par exemple pour la 2 ça m'a aidé j'ai eu bon mais j'étais pas sûr, ça m'a aidé à voir directement". Il dit également "Pour les booléens c'est très utile, aussi pour le premier exo input mais là c'est une erreur de ma part, et les booléens je ne connais pas ceux-là". Les visualisations l'ont donc bien aidé concernant les types mais il reste cependant des incompréhensions concernant le contenu des variables.

Pour **l'exercice 3** qui est le premier programme à exécuter étape par étape, l'étudiant n'a pas montré d'incompréhensions concernant l'objectif du programme. Il n'a cependant pas remarqué l'erreur de type au niveau de la comparaison entre un Int et un Str. A l'assignation de la variable num3, il dit : " : Hein ? pourquoi il est bleu le num3, ha oui car on l'a mis entre guillemets. Ha oui donc c'est un string ce n'est pas un integer. Ha non il n'y a pas le Int devant !". Il n'avait pas vu que contrairement aux deux premières variables, la troisième n'était pas castée en Int. Il l'a donc remarqué grâce aux visualisations.

Lors de **l'exercice 4**, l'étudiant fait la remarque suivante : "Sur le pc j'exécute, ça ne marche pas ? je vais voir je dis ha oui c'est une virgule, ha oui d'accord. Si y'avait ça à l'examen ça serait très bien !(rire)" en parlant d'un problème qu'il a eu dans son projet de programmation. Il pense que les visualisations de type aideraient beaucoup à trouver des erreurs dans le code. Il remarque également l'erreur de l'exercice via la visualisation : "Ha y'a une erreur ! Pourquoi y'a une erreur... (après 10 sec) haaa on additionne encore un string avec un Int !". Il n'avait pas remarqué le problème sans ou avec les visualisations. La notation utilisée pouvait faire penser à celle-ci (qui est correcte) :

```
print("La réponse est", fonction_somme(terme1, 10))
```

Lors de l'appel de fonction dans **l'exercice 5** : "Haa la fonction retourne un booléen en fait !". Je lui demande s'il comprend pourquoi et il répond : "Non pas vraiment". Je lui explique qu'étant donné l'utilisation de l'opérateur de comparaison "==" il s'agit d'une expression booléenne. Il dit : "haaa oké c'est comme si on utilisait un if !". Le fait d'utiliser l'expression booléenne directement dans le return l'a perturbé. Il aurait préféré qu'on passe par une conditionnelle avant de retourner True ou False.

### Étudiant 2

Le deuxième étudiant, tout comme le premier, a montré certaines incompréhensions lors du **premier exercice** :

**Variable2** : "Le type n'est pas précisé, il peut rentrer un Float ou ... etc. ça viendra après en fonction de ce qu'on rentre". L'étudiant pense que le type renvoyé par la fonction input() dépend de ce que rentre l'utilisateur au clavier. Par exemple, si l'utilisateur rentre "250", il s'agira d'un entier. S'il rentre "bonjour", ça sera une chaîne de caractères.

Pour la **variable3** : "Ici donc c'est 100 et integer". L'étudiant semble comprendre que le type sera Int grâce au casting.

**Variable4** : "Ici c'est un entier qui vaut 10 > 50. Juste la variable 4 ça se peut être un entier comme réel ou Float j'ai mis directement réel, car tout dépend il peut donner ... mais en fait ici c'est un seul exemple donc c'est pas l'utilisateur qui va donner une variable. Ici ça deviendrait un entier car c'est un 10". L'étudiant a semblé penser qu'on attendait une entrée de la part de l'utilisateur. Il s'est ensuite rendu compte qu'il n'y avait pas d'input() et que toutes les données nécessaires étaient présentes. Il n'a pas compris qu'il s'agissait d'une expression booléenne et a répondu que la variable était de type Int et contenait 10 > 50. Ce qui est donc incorrect.

Pour la **variable5**, il dit "100%90 > 50, le contenu va être un peu bizarre, ha non c'est bon je prends en considération les guillemets !". Après avoir hésité, il repère les guillemets et indique donc correctement qu'il s'agit d'un Str ayant comme contenu toute la chaîne de caractères.

Pour la **variable6**, nous avons de nouveau une fonction input() sans aucun cast, il continue donc de penser que le type dépendra de l'entrée de l'utilisateur.

Pour les **variables 9, 10 et 12**, il indique qu'il s'agit de trois erreurs directement. Bien qu'il ait raison pour la 9, les 10 et 12 sont correctes.

**Variable14** : "par contre la variable 14 je n'ai pas bien compris, c'est variable 13.__contains__ ...". L'instructeur explique donc que contains() vérifie si la variable contient bien ce qu'on donne en paramètre. Il répond : "Alors c'est un booléen qui vaut vrai".

Pour la **variable13** qui est un tuple, l'étudiant réagit comme tel : "ici ça va afficher une erreur car y'a une virgule après. Y'a une virgule donc on va donner soit un autre commentaire sinon une autre variable qui va prendre par exemple cet entier est égal à, et tu fermes les guillemets, virgule a". Ici, l'étudiant pense qu'il manque d'autres éléments à la structure de données. Il mentionne aussi la notation qu'on peut utiliser lors de l'utilisation de la fonction print() :

```
mot = "Dora"
print("Bonjour", mot)
```

Ce qui affichera "Bonjour Dora" dans le terminal. La notation utilisée dans l'exercice va simplement créer un tuple contenant un seul élément : "Bonjour".

Pour le **deuxième exercice**, l'étudiant dit directement : "Ha ça va être facile ! Franchement oui... Je regarde juste les couleurs, franchement ça aide bien." Il donne ensuite les bons types à chaque variable jusque la 8.

A la **variable8**, il dit : "Append... ça va rien afficher y'a pas l'élément 12, et ici on a bonjour, ça va pas fonctionner". Il confond probablement la fonction append() avec la fonction contains().

Pour la **variable10**, il est étonné : "Tout ça c'est un booléen, ça je savais pas ! Pour moi ça c'est une erreur". Il ne semble pas comprendre comment fonctionne une expression booléenne. Il a d'ailleurs mis que les variables Bool du premier exercice allaient donner des erreurs. Après explication, il se corrige correctement. Il dit aussi pour la variable10 du premier exercice : "Le 10 pour moi c'était une erreur car c'est pas logique de mettre (5<10)+5."

Pour la **variable9** qu'il avait passé dans un premier temps, il comprend que la variable vaudra 0 car on soustrait 1 à True.

En parlant de la **variable3**, il dit : "Ha je me suis trompé ! c'est l'inverse j'aurais dû mettre False". Il pense que quand on cast un string en Bool, on prend l'expression comme elle est pour la résoudre. Des explications lui ont donc été données.

Après avoir parcouru le **troisième exercice**, il donne son avis concernant la visualisation des types : "Juste un point de vue, au lieu de mettre tout en bleu avec le rouge" (il parle des input qui sont castés en int) "Ça serait plus utile de mettre la couleur finale. Par exemple ici, vert et bleu c'est un booléen, je laisserais tout en vert. Par contre si c'est juste un simple input, en bleu directement. Ça peut prêter à confusion, c'est mon point de vue, mais c'est très utile !". Il trouve qu'avoir plusieurs couleurs qui se chevauchent peut prêter à confusion.

Lors du **quatrième exercice**, l'étudiant a repéré l'erreur grâce à la visualisation. Il réagit : "Alors la réponse ... Ici ça va afficher une erreur ! Ha oui c'est ça la remarque que j'ai fait tout avant pour la virgule si tu te souviens. Print c'était normalement la réponse du calcul est, après une virgule, la fonction... la ça aurait fonctionné. Ici avec le + ça fonctionne pas". Il confond à nouveau avec la notation "virgule" qu'on peut utiliser avec la fonction print().

Il rencontre à nouveau certaines difficultés avec le return booléen de la fonction de **l'exercice 5** : "Ta fonction permet d'inverser le mot, ça va renvoyer toujours un string". Une fois face à la visualisation de l'appel de fonction, il comprend que la fonction retourne un Bool : "c'est un booléen c'est donc True, donc ça va pas être exécuté...". Il pense dans un premier temps que la fonction retournera True, ce qui est une erreur. Il regarde encore le return et dit : "Donc c'est == la chaîne, les mots sont égaux, mais ici il manque un truc dans la condition non ? ". Comme le premier étudiant, il a du mal à comprendre qu'on puisse retourner directement une expression booléenne avec un return. Il aimerait utiliser une conditionnelle avant.

### Étudiant 3

Lors du **premier exercice**, cet étudiant a eu globalement les mêmes difficultés :

**Variable2** : "Variable 2 c'est input ou on entre une valeur et on entre 10.5 donc je dirais que c'est un heu... Float et le contenu c'est 10.5". Il pense également que le type que renvoie la fonction input() dépend de l'entrée de l'utilisateur.

Pour la **variable4**, l'étudiant hésite beaucoup avant de trouver la bonne réponse : "Variable 4 c'est 100 divisé par 90 qui doit être plus grand que 50 donc je dirais que c'est un ... bah ça ferait un Float mais le problème c'est que la condition serait pas très bonne car ça serait jamais plus grand que 50 le reste. J'aurais tendance à dire que ça serait un Bool, de dire que si c'était possible oui ou non de ... je dirais Bool et comme le reste de 100%90 c'est plus petit, la réponse c'est no".

L'étudiant hésite aussi à la **variable5** mais finit par trouver qu'il s'agit d'une chaîne de caractères : "La 5 je vois pas du tout ce que c'est, le fait qu'il y ait des guillemets. Ça veut juste dire que ça va renvoyer le contenu total. Je sais plus comment ça s'appelle, c'est du string ? ça va renvoyer '100 % 90 >50'".

Il se trompe également à la **variable6** à cause de la fonction input().

Lors de l'addition d'un Str avec un Int à la **variable9**, il ne dit pas qu'il s'agit d'une erreur et il pense que le résultat sera un Str : "Variable9 ça renverra bonjour et 22.5 et il me semble que ça sera un string parce que si y'a des caractères c'est forcément un string. Je sais pas si je dois écrire comme c'est écrit, bonjour + 22,5"

A la **variable10**, il comprend qu'on additionne une expression booléenne avec un entier. Il ne sait cependant pas comment cela fonctionne : "variable10 c'est 5 < 10 ce qui est vrai, + 5 ça fait 10". A la question : "Vrai additionné à 5 ça fait 10 ?", il répond : "Ha non c'est bizarre .. J'aurais tendance à dire que c'est une erreur. Car un Bool + un Int c'est pas possible." En voyant la **variable11**, il se dit que la précédente était possible : "La 11 ça dépend car je sais pas si c'est 14 <10 ou 14 < 10+5. Je dirais que comme il n'y a pas de parenthèses c'est 14 <10 et puis + 5 après. Roh j'hésite... La variable 10 vu que je savais pas je sais qu'elle est possible(rire). À moins que ça renvoie encore une erreur. Là je dirais que la 11 c'est un Bool et c'est vrai". Il a finalement trouvé la réponse correcte pour la variable 11 qui est effectivement booléenne.

Concernant les opérations arithmétiques contenant un Bool, il dira pour la **variable9 du deuxième exercice** : "La var9 c'est un Int, c'est 8 >4 ce qui est vrai, -1. Ça donnerait 7. On vérifie que 8 est > que 7, et si c'est le cas on fait -1 ! Ça donnerait 7." Il prend donc le premier élément de l'expression booléenne si elle est vraie, et exécute l'opération.

Il ne sait pas ce que veut dire l'assignation de la **variable13** avec le tuple. Il trouve par contre la bonne réponse pour le reste des variables du premier exercice.

Pour les variables du **deuxième exercice** :

**variable2** : "Le deuxième c'est écrit Str entrez une valeur c'est 10, mais ça renvoie un nombre donc je dirais que c'est Int... mais c'est ça le problème...". L'étudiant est perturbé car assez sur de lui pour le fait que la variable soit un entier. Il décide de répondre Int malgré la couleur.

**variable3** : "Comme j'avais dit avant c'est vrai ou faux donc ici c'est faux parce que le reste ça fera 10". La couleur indiquant qu'il s'agit d'un booléen le conforte dans son idée initiale. Sa réponse concernant le contenu n'est pas correcte mais il ne s'agit pas d'une erreur de type.

Pour le restant de l'exercice 2, le troisième étudiant n'a pas fait de nouvelles erreurs concernant les types. Ses erreurs proviennent des opérations dont il n'est pas toujours certain de l'exécution.

Pour le **troisième exercice**, l'étudiant comprend directement les visualisations concernant les types lors des assignations. Les couleurs lui permettent de repérer l'erreur de comparaison avant qu'elle survienne : "C'est un string car y'a pas de Int devant. Ça va renvoyer une chaîne de caractères. Ici une chaîne de caractères qui est plus grande qu'un Int, c'est pas possible, ça va forcément renvoyer une erreur". Il dit également : "J'avais pas vu qu'il n'y avait pas le Int mais la couleur m'a montré". Il n'avait pas vu que le retour de la fonction input() n'était pas casté en Int cette fois-ci.

L'étudiant n'a pas repéré l'erreur pour **l'exercice 4** à l'avance. Comme expliqué plus haut, elle est plus difficile à trouver que la comparaison de l'exercice 3. Il avait sinon correctement compris les notions liées aux types dans la fonction.

Comme les deux premiers étudiants, celui-ci pense (initialement, sans les visualisations) que la fonction de **l'exercice 5** retourne une chaîne de caractères : "ok donc la ça voudrait dire que la chaîne de base est égale à l'inverse, c'est le principe du palindrome. Et puis c'est est_palindrome c'est pour dire si c'est un palindrome ou pas. On établit un égal pour renvoyer le palindrome. Je pense que est_palindrome est un string". Après avoir repéré le '==' grâce à un peu d'aide, il comprend le reste du programme. Il était donc surpris de voir le return avec une couleur verte, ce qui lui a permis de comprendre qu'il n'avait pas bien compris la ligne.

### Types et opérations - Conclusion

L'expérience des utilisateurs avec les visualisations des types a été globalement très positive. Il n'a fallu que quelques secondes face à la légende avant de commencer à résoudre le deuxième exercice pour chacun des trois étudiants. Les exercices ont été pensés afin de passer en revue plusieurs cas de figure concernant les types en Python, chaque étudiant a donc commis un bon nombre d'erreurs. Les problèmes les plus récurrents ont été :

- Connaître le type de retour de la fonction input(). Il s'agit d'un cas plus complexe qu'une simple assignation, cependant les programmeurs sont régulièrement confrontés à de nouvelles fonctions (natives ou non) qu'ils n'ont pas écrit eux-mêmes et il est donc utile de les assister dans cette tâche via la visualisation. Dans ce cas précis, celle-ci pourrait être utilisée comme une certaine documentation.

- Reconnaître une expression booléenne. En effet, celles figurant dans les exercices étant principalement des comparaisons de nombres, les étudiants ont pensé qu'ils faisaient face à des valeurs numériques. Dans la fonction de l'exercice 5, il a fallu leur faire remarquer à tous que l'opérateur utilisé était "==" et non "=". Le fait de retourner directement une expression au lieu de la stocker au préalable dans une variable a aussi dérangé 2 étudiants.

- Comprendre comment créer un tuple contenant un seul élément. Il faut reconnaître qu'il s'agit d'un cas plus spécifique.

- L'ordre des opérations lors des expressions arithmétiques, comme dans les variables 9 et 10 du deuxième exercice. Cet ordre est primordial dans ce cas précis car il permet de savoir si le résultat sera de type entier ou booléen, en fonction de ce par quoi on commence (comparaison ou opérateur de calcul). La visualisation de la variable 9 permettait de comprendre qu'il fallait en premier lieu résoudre le Bool avant d'y soustraire 1. Pour la variable 10, comme toute l'expression était en vert, il était difficile de le voir directement sans connaître la priorité des

opérations. Il est cependant possible d'y ajouter plus de détails de couleurs. Par exemple, on pourrait remplir 10 + 5 en rouge avant de mettre tout en vert. Ce point d'amélioration sera discuté ultérieurement dans ce travail.

- Résoudre certaines opérations arithmétiques contenant des valeurs de différents types. Les opérations contenant seulement des entiers n'ont pas posé de problèmes. Un étudiant a pensé qu'une addition entre un entier et un Float donnait aussi un Int, ce qui est incorrect. Aucun étudiant ne savait qu'il était possible d'utiliser des expressions booléennes dans des opérations. Il s'agit également d'un cas plus spécifique qui n'est pas souvent rencontré dans les programmes. L'opération entre Str et Int a bien été repérée comme une erreur par les étudiants. Ils ont également compris que l'addition de deux chaînes de caractères revenait à les concaténer. Les différents comportements de ces opérations doivent être connus par les étudiants car il n'est pas possible de tous les mettre en lumière rien qu'en utilisant la visualisation sur les types, même si dans certains cas elles peuvent aider. Cette partie ci sort du coup de l'étendue du travail qui se limite à la visualisation des types.

Toutes les remarques faites par les étudiants concernant les couleurs et les visualisations des types ont été positives. Il leur a été demandé d'exprimer leur ressenti par rapport aux couleurs et s'ils n'avaient pas été perturbés par certaines visualisations qu'ils trouveraient illogiques. La seule remarque/proposition d'amélioration qui a été faite est celle du deuxième étudiant qui disait que mélanger les couleurs dans une expression pourrait peut-être "prêter à confusion". Cette fonctionnalité permet cependant de bien comprendre les détails d'une expression et peut même aider à comprendre son fonctionnement. Afin de connaître le type final d'une expression, il suffit de regarder la couleur qui l'englobe totalement.

### 4.2.2 Fonctions

Seuls les exercices 4 et 5 utilisaient une fonction lors du test utilisateur. Les avis des étudiants concernant la compréhension de la visualisation vont être analysés dans ce chapitre. Pour être sur que tout semble clair aux étudiants, une question leur a été posée concernant les différences de visualisation entre la définition et l'appel d'une fonction.

#### Étudiant 1

Pour **l'exercice 4**, quand le premier étudiant voit la visualisation de la définition de fonction pour la première fois : " ok, on rentre les paramètres et le résultat ici. C'est le fonctionnement de la fonction avec son nom". Le mécanisme de base de la boite lui parait donc logique. A la question, il répond : "Elle est activée du coup elle devient grasse. Je trouve ça logique. Ça m'a aidé moins que les couleurs. Si on avait des fonctions qui s'appelaient entre elles même, ça serait utile !". Le concept de fonction avait l'air d'être bien intégré chez cet étudiant. La visualisation ne lui apportait rien sur ces deux exemples. Cependant il comprend directement comment elles ont été représentées et il mentionne le fait que ça serait plus utile si des fonctions pouvaient s'appeler entres-elles (il pensait peut-être également à la récursion en disant cela).

#### Étudiant 2

En voyant la définition de fonction de **l'exercice 4**, il dit : "Ça c'est l'entrée et la sortie de la fonction, avec ce qui rentre ce qui sort". Comme le premier étudiant, la visualisation de base semble clair. A la question sur les différences entre la définition et l'appel de fonction, il répond : "Pour moi ça ici on est dans la fonction, alors qu'ici pas. Ça va prendre terme1 terme2 ça va faire la somme

des 2 et ça va sortir mais ici on rentre dans la fonction parce qu'on fait un appel, voilà c'est ça". Il semble comprendre la différence entre les deux fonctionnalités.

#### Étudiant 3

Face à la définition de fonction de **l'exercice 4**, il dit : "Ici on a un def avec terme 1 et 2 c'est pour pouvoir le réutiliser après". Il associe directement le "def" avec un morceau de code qu'il peut réutiliser à tout moment plus tard. L'instructeur lui demande : "Que penses-tu du petit carré pour la fonction ?". Il répond : "ça peut aider pour bien comprendre le return pour dire qu'on va entrer ces termes la dans le code et qu'après on pourra les réutiliser quand on veut grâce au return".

A la question : "Ici tout est noir, alors que là c'est gris, tu sais pourquoi ? " il répond : "Parce que ça retourne rien pour le moment. Si on lance juste ça, ça fera rien. C'est après qu'on peut le réutiliser et ça devient noir". Il comprend bien que la définition de fonction n'a aucun effet tant qu'il n'y a pas d'appel. Quand une fonction est appelée, le code est exécuté. Après lui avoir demandé s'il trouvait les visualisations logiques, il dit : "Oui les flèches pour dire qu'on entre les termes et qu'après on peut les réutiliser quand elles ressortent c'est pas mal. Les couleurs c'est bien aussi pour dire là c'est grisé donc si on le lance ça fera rien. Quand c'est en noir ça retournera des chiffres".

#### Fonctions - Conclusion

Tous les étudiants ont directement compris la visualisation des fonctions après le quatrième exercice du test. Le premier étudiant disait cependant ne pas vraiment en avoir besoin. Les étudiants du deuxième quadrimestre étant déjà à un stade avancé de leur projet, l'utilisation de fonctions n'était plus un concept nouveau. La visualisation des fonctions a été pensée pour des étudiants qui ne seraient pas encore familiarisés avec ce concept.

Les exemples de fonctions donnés avec les exercices 4 et 5 étaient extrêmement basiques et les programmes étaient très courts. Lors de l'utilisation de programmes plus longs, la visualisation des différentes fonctions déclarées tout au long du programme serait plus utile car il ne faudrait pas spécialement retourner voir le code pour s'en souvenir. Le développeur aurait la liste des fonctions déclarées dans la partie droite de son IDE. Il pourrait potentiellement agrandir la fenêtre d'une fonction pour visualiser les arguments à passer à la fonction avant de l'appeler. La visualisation pourrait être utilisée comme documentation en temps réel lors des activités de programmation.

### 4.2.3 Conditionnelles

Les concepts en rapport avec les conditionnelles que la visualisation permet d'éclaircir sont assez basiques. Il s'agit juste de tester la condition et griser le code qui ne sera pas utilisé. A priori, cette matière a déjà été correctement acquise à ce stade de leur formation. Il faut tout de même s'assurer que les étudiants soient à l'aise avec ces visualisations.

#### Étudiant 1

Pour **l'exercice 3**, le premier étudiant a correctement compris le fonctionnement des if/else lors de la première lecture du programme. Il rentre dans les bons morceaux de code et n'exécute pas ce qui ne doit pas l'être. L'instructeur lui demande s'il a bien compris les différents éléments de visualisation qui sont apparus pour les conditions et il répond : "Oui c'est grisé car on l'utilise pas du coup. Puisque le cas est ici, on est dans le if pas dans le else".

Lors de **l'exercice 5**, l'étudiant comprend que la croix rouge signifie que la condition est évaluée à False, il dit : "est_palindrome vaut vrai ou faux, ici c'est faux donc c'est pas un palindrome, forcément. On passe au else et ça affichera n'est pas un palindrome".

Le premier étudiant a donc l'air d'avoir bien compris tous les concepts et visualisations.

#### Étudiant 2

Le troisième étudiant ne disant pas grand chose lors de l'exercice 3, l'instructeur lui demande d'expliquer les nouvelles visualisations qui sont apparues. L'étudiant répond : "Le petit V qui veut dire que c'est vrai, c'est vérifié donc on a pas besoin de passer sur le Else". A nouveau, tout semble logique. Il dit cependant ensuite : "Ici c'est 45 donc le max va prendre 45 car supérieur à 15. Mmmh attends, ici c'est un string , donc c'est une erreur. Ha oui il n'y avait pas le Int ! Ça va afficher le 15". L'étudiant repère ici l'erreur grâce au panneau. Il pense que le programme va continuer à s'exécuter et que la condition sera du coup évaluée à False.

Lors de **l'exercice 5**, l'étudiant regarde la croix rouge et dit : "J'aime bien le panneau". L'instructeur lui explique donc qu'il ne s'agit pas cette fois-ci d'un panneau signalant une erreur mais bien que la condition est évaluée à False. L'étudiant comprend cependant correctement pourquoi une partie du code est grisée : "Ça va pas être exécuté on va passer directement à la dernière" (en montrant le else).

Après que l'instructeur lui ait demandé son avis sur les visualisations générales des conditionnelles, il répond : "Une idée, pour les croix et le panneau, il y a aurait moyen d'ajouter ce que ça signifie, car personnellement j'ai confondu". Une solution sera apportée afin que les étudiants ne puissent plus confondre certains symboles.

#### Étudiant 3

Le troisième étudiant, face à **l'exercice 3**, dit "Donc ici on voit que la première condition elle est correcte, max sera égale à num1. Le else est forcément oublié". Il comprend directement toutes les visualisations.

Pour le **cinquième exercice**, il dit "Est palindrome vaut false donc forcément c'est pas le premier if, on passe directement au else. Ça affiche, 'n'est pas un palindrome', comme indiqué dans le terminal". Ses explications sont claires et démontrent directement qu'il est à l'aise avec ces nouveaux éléments.

#### Conditionnelles - Conclusion

Les concepts semblent bien compris par tous les étudiants. Le seul souci a été la confusion du panneau d'erreur avec la croix.

### 4.2.4 Visualisation globale

Tout au long du test, certains étudiants ont donné leur avis sur l'outil de visualisation en général. Ces avis ne portaient pas spécialement sur les concepts vus ci-dessus mais sur le fait d'être assisté par de la visualisation et des couleurs lors des activités de programmation.

L'étudiant 1 dit : "En fait c'est une qualité de vie, tu sais te débrouiller sans mais une fois que tu l'as c'est beaucoup mieux !"

L'étudiant 2 dit à plusieurs reprises que les visualisations sont très utiles et que ça l'a bien aidé.

L'étudiant 3 dit que l'outil serait encore mieux pour des étudiants "qui ne s'y connaissent pas". Il dit aussi "En conclusion, ça pourrait être pas mal si quelqu'un s'y connaît pas et veut se lancer la dedans et a peur de coder. Je sais bien qu'au début, quand je voyais des lignes de code ça me faisait un peu peur. Avec des couleurs ça pourrait rendre ça un peu plus ludique". Les couleurs, animations et visualisations peuvent rassurer et permettre de "décomplexifier" une matière qui peut sembler très compliquée au premier abord. Le fait que cet outil puisse aider à motiver les étudiants et susciter leur intérêt concernant le développement est un bénéfice important.

### 4.2.5 Visualisations en temps réel

Les trois étudiants ont porté globalement le même avis sur la question : "Trouverais-tu cela utile d'avoir les visualisations des types en temps réel lors des activités de programmation ?". Le premier étudiant dit : "Oui si on peut le désactiver. Si on a 100 lignes de code c'est trop. Alors je le réactive pour regarder un peu et comparer"

Le deuxième étudiant dit "Oui personnellement oui, mais trop de couleurs ça va gêner un peu. Ça te facilite le codage, mais trop de couleurs ça me gêne un peu. Si on a un bouton activer/désactiver ça serait mieux. Par exemple t'as une erreur, tu peux voir les types, tu sais ce qu'il se passe".

Le troisième : "Je pense que ça pourrait porter à confusion car c'est pas mal aussi dans VScode quand on fait un 'def' c'est pas noir on peut mettre des couleurs. Ça serait dérangeant. Par exemple quand on fait mot est égal à quelque chose, il s'affiche déjà en couleurs donc ça pourrait confondre". L'étudiant parle des couleurs qui sont déjà présentes sur certains IDE. Je lui explique donc que je ne mettrais que mes couleurs et il répond : "A ce moment-là ça pourrait être utile !".

### 4.2.6 Première analyse - Conclusion

Le test utilisateur ainsi que les analyses permettent d'avoir une idée globale sur le niveau des étudiants ciblés, les difficultés qu'ils rencontrent et l'utilité d'un tel outil lors de l'apprentissage de la programmation.

Le retour des étudiants sur leurs expériences est globalement très positif. Leur avis a permis de déceler certains points d'amélioration qui seront travaillés à la section suivante. Ces points concernent principalement l'amélioration de la légende et la mise en forme des couleurs sur des expressions complexes.

Le premier test ayant été réalisé au milieu du deuxième quadrimestre permet de constater que toutes les visualisations ne se concentrent pas sur des concepts de la même complexité. Par exemple, les couleurs des types ont bien assisté les étudiants lors des exercices du test, alors que les visualisations sur les fonctions et conditionnelles mettaient en lumière des concepts déjà généralement bien intégrés. Il ne sera malheureusement pas possible de faire un test utilisateur avec des étudiants du premier quadrimestre, comme cela aurait été optimal.

## 4.3 Amélioration du prototype

Les expériences des étudiants ainsi que la mise en place du test utilisateur ont permis de repérer plusieurs pistes d'améliorations. En effet, le fait de créer des simulations de l'outil et expliquer précisément son fonctionnement lors de la section 4.1.1 a soulevé des interrogations concernant certaines fonctionnalités. Les points discutés dans ce chapitre sont :

- Le niveau de détail à apporter avec les couleurs dans des expressions complexes et comment uniformiser les visualisations
- Les éléments à ajouter à la légende
- Des visualisations de type à ajouter dans la partie "Variables"
- Adaptation de l'interface à des programmes plus longs et complexes
- Ajout du contenu des variables internes à une fonction
- Ajout d'une séparation stack/heap dans la partie "Variables"

### 4.3.1 Visualisation des types

#### Uniformisation des visualisations

Comme le montre la figure 4.1, le fait qu'un élément soit encadré ou rempli de couleur n'est pas très logique. Lors des assignations, le nom de la variable est encadrée et l'expression de droite est colorée. Dans les visualisations des fonctions, les éléments sont toujours colorés. Partout sauf quand il s'agit d'une assignation, les éléments sont remplis de couleur.

Un moyen plus logique de gérer cela est d'encadrer les noms de variables et colorer les expressions, quel que soit l'endroit ou l'on se trouve dans le code.

#### Niveau de détails dans les expressions

Lors du deuxième exercice du test utilisateur (voir figure 4.6), la variable 10 avait posé beaucoup de problèmes aux étudiants car ils ne savaient pas spécialement dans quel ordre appliquer les opérateurs.

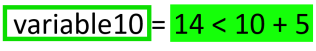



FIGURE 4.6 – Représentation initiale de l'expression

En effet, la visualisation permet juste de constater que l'expression est de type booléenne et ne donne pas plus de détails concernant ses différents composants. En mettant plus de détails sur les expressions, comme cela a été fait pour la figure 4.7, on peut voir que la partie de droite donnera un entier, qu'on comparera ensuite au premier entier. Ces paramètres pourraient être adaptés en

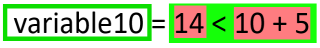



FIGURE 4.7 – Représentation détaillée de l'expression

fonction du niveau des étudiants ainsi qu'avec leurs propres préférences. Un autre exemple d'une expression qui peut sembler compliquée pour un novice se trouve à la figure 4.8. Les couleurs montrent les différentes comparaisons entre valeurs entières.

```
return (annee % 4 == 0 and not annee % 100 == 0) or annee % 400 == 0
```

FIGURE 4.8 – Expression vérifiant si une année est bissextile

### 4.3.2 Visualisations des fonctions

Les fonctions utilisées en exemple lors des tests utilisateurs sont très simples et ne contiennent qu'une ligne de code. Pour des programmes réels dans lesquels des fonctions peuvent avoir une taille conséquente, il faut que la taille des boites soit adaptée. Il manque également la notion de scope avec l'affichage du contenu des variables internes à la fonction en temps réel, comme c'est le cas pour le reste du programme.

Antérieurement, le nom des paramètres d'une fonction étant appelée était le nom de la variable ou expression du programme principal utilisé pour appeler cette fonction (voir figure 4.1 comme exemple). Cela prêtait à confusion car on pouvait penser que la variable "mot" faisait donc partie de la fonction, ce qui n'est pas le cas. Pour utiliser la chaîne de caractères passée en paramètre, il faut utiliser le nom "chaîne", comme mentionné dans la définition de la fonction.

Étant donné que le corps d'une fonction dépasse souvent une seule ligne, il faudrait avoir un moyen pour afficher plus de contenu. Une possibilité serait de, dans la partie fonction, seulement afficher son nom et ses paramètres ainsi qu'indiquer si elle contient un return. Une fois que la fonction est appelée, sa fenêtre s'agrandit et son code apparaît. On pourrait alors continuer l'exécution du programme pas à pas à l'intérieur du corps de la fonction et constater les valeurs des variables en temps réel, comme c'est le cas pour le programme global.

Quand certains types de variables sont connus dans le corps d'une fonction lors de sa définition, les couleurs sont appliquées. Il est par exemple possible de repérer une expression booléenne dans un return. L'étudiant saura alors directement que la fonction retournera une valeur de type Bool, quels que soient les paramètres passés à la fonction lors de l'appel.

### 4.3.3 Légende

Certains éléments de visualisation n'étaient pas repris dans la légende et ont posé problème à un étudiant. Les représentations des erreurs ainsi que celles des gardes évaluées à True ou False ont donc été ajoutées, comme le montre la figure 4.9.

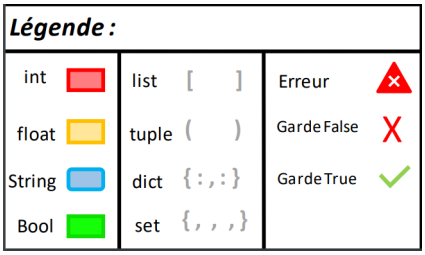



FIGURE 4.9 – Légende complétée

### 4.3.4 Variables

Premièrement, les couleurs de types sont ajoutées à la partie "Variables" de l'outil. Cela permettra d'avoir accès rapidement aux différents types sans devoir naviguer dans le programme.

La partie "Variables" est également découpée en deux parties, comme c'est déjà le cas dans l'outil Online Python Tutor. Dans la partie "Global Frame" vont se trouver les valeurs des différentes variables primitives. Dans la partie "Objects" se trouveront les objets tels que les structures de données et seront pointés via une flèche (pointeur) provenant de la "Global Frame".

Comme mentionné dans l'article [9], Python propose de nombreuses fonctionnalités à la sémantique complexe mais qui sont souvent réalisées à l'aide d'une syntaxe fortement simplifiée. Cela a des avantages pour les développeurs connaissant en profondeur le langage mais il s'agit surtout d'une difficulté pour les novices qui se réfèrent à leurs intuitions concernant le comportement de certaines propriétés du langage. Une de ces propriétés que l'article donne en exemple est l'opérateur "+" qui donne des résultats très différents en fonction du contexte dans lequel il est utilisé. En effet, Python surcharge fortement cet opérateur qui peut additionner des entiers et des Float ainsi que concaténer des chaînes de caractères ou des structures de données.

Il existe aussi en Python l'opérateur "+=" qui n'agit pas spécialement comme l'on pourrait le penser. Il est souvent mentionné que, par exemple :

```
a += b
```

est équivalent à

```
a =  a + b
```

Cela n'est pourtant pas le cas. Par exemple, prenons "a", une liste quelconque :

```
a = a + [4, 5]
```

va recréer une nouvelle liste contenant les valeurs de l'ancienne avec les nouvelles, alors que :

```
a += [4, 5]
```

va simplement ajouter les nouvelles valeurs à l'objet existant. Ne sachant pas cela, un novice peut rapidement se tromper quand il voudra par exemple modifier une variable ne comprenant qu'il fait face à un partage de références.

Ces concepts étant jugés importants lors de l'apprentissage de la programmation avec Python, une assistance visuelle est ajoutée au prototype. La partie "Variables" montre clairement les différents objets ainsi que les potentiels pointeurs les liant. Dans l'exemple de la figure 4.10, la ligne de code :

```
year_list_last_century.append(year_list_this_century)
```

a été exécutée. On peut facilement constater que la première liste ne contient pas simplement tous ses anciens éléments auxquels ceux de la deuxième liste ont été ajoutés. Le dernier élément de la liste contient en fait l'objet de la deuxième liste. Sans savoir cela, des opérations sur la liste pourraient avoir des comportements non voulus ou ne fonctionneraient pas correctement.

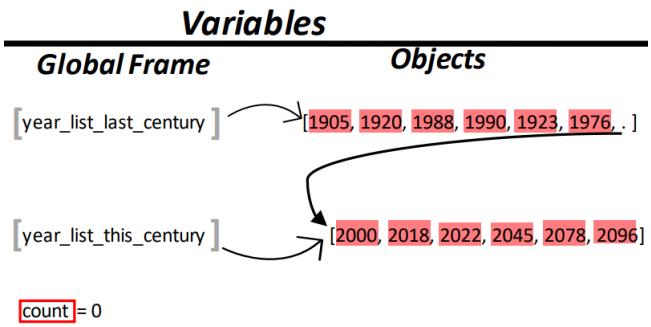



FIGURE 4.10 – Partie "Variables" améliorée

## 4.4 Deuxième analyse

Dans cette section se trouveront les résultats du deuxième test utilisateur. Les différentes phrases intéressantes des étudiants prendront place dans l'ordre chronologique des exercices. Les différents exercices peuvent être retrouvés à la section 3.3.

### 4.4.1 Étudiant 4

Même si le niveau global lors de ces deux nouveaux entretiens était bien plus haut que lors du premier, certaines erreurs ont quand même été commises en faisant face au **premier programme**.

L'étudiant (sans voir de visualisations) répond que la **variable1** est de type entier. La visualisation le corrige ensuite et il comprend qu'il s'agit d'un Float. Il dit : "Du coup Float j'hésitais entre Float et Int, je pensais que Float c'était quand on avait plusieurs nombres à virgule, mais en fait ça le fait avec un seul nombre à virgule". Il pensait que dans une opération entre Int et Float, il faudrait plus de deux Float afin que la variable soit de ce type. Il comprend alors qu'il en suffit que d'un seul pour que la variable soit de type Float.

L'étudiant répond correctement pour les **variables2, 3 et 4**. Pour la 5, il dit : "Variable 5 ça va être un string la valeur, et quand on va rentrer ça va être un int". Il répond sur la feuille que la variable est un Str et contient "30 int". Cette incompréhension est intéressante car il comprend que la valeur sera entière mais il laisse quand même le type de la variable en Str. La visualisation ne lui permet pas de modifier sa réponse car il avait déjà compris l'expression comme cela.

Pour la **variable6**, il dit que les deux listes auront le même contenu, ce qui est vrai. Quand il voit la visualisation dans la partie "variables", il dit : "tu prends la même variable au final que la variable4 du coup ça devient aussi une liste". Il constate que les deux variables pointent vers le même objet. Cependant, quand on ajoute un élément à la variable4 en utilisant la fonction append(), il ne modifie que cette variable-là, sans penser à également modifier le contenu de la variable6. Avec un peu d'aide de l'instructeur, il comprend et dit : "je pense que j'aurais pas pensé à variable6 et je pense que je l'ai pas noté d'ailleurs, j'avais oublié qu'elles étaient égales". La visualisation du partage de référence semble clair à l'étudiant mais comme il ne savait pas qu'il était possible de partager la même valeur entre plusieurs variables, il ne se rendait pas compte qu'en modifiant une des deux, l'autre serait modifiée aussi. Il faut aussi noter qu'il n'y a eu qu'une seule liste créée avant de réaliser le partage de référence, il aurait peut être compris par lui même si une deuxième liste avait été créée dans la partie "Objects".

Il comprend correctement la conditionnelle liée à la **variable7** et donne la bonne réponse. La visualisation confirme sa réponse. Il comprend que la croix veut dire que la condition est évaluée à

False et qu'il doit exécuter le code repris dans le else.

Sans visualisations, il trouve la bonne réponse concernant la fonction et la **variable8**. Il explique : "C'est une fonction ou t'as deux paramètres : terme1 et terme2. Tu retournes terme1 + terme2 du coup la variable8 ça va être 20,5 + 30 et ça va faire 50,50". Face à la visualisation, il dit : "La fonction, c'est les paramètres et ton return au final. Du coup ça va être un Float et un Int ce qui va faire un Float au final, comme ça va être un nombre à virgule. J'ai noté Int car j'avais une autre idée de Float au final". Quand l'instructeur lui demande pourquoi certaines visualisations sont grises et d'autres noires et si c'est logique, il dit : "Parce qu'elle est exécutée et qu'avant elle était juste définie ?".

Lors de la lecture sans visualisations du **deuxième programme**, l'étudiant comprend correctement ce qu'il fait. Il commence à exécuter le code ligne par ligne et l'instructeur demande combien d'années bissextiles le programme affichera. L'étudiant répond : "Je pense qu'il y a aura une erreur". Il constate les visualisations liées aux fonctions sans montrer d'incompréhensions, comme pour l'exercice précédent. Lors de la fonction append() utilisée avec deux listes, il constate la partie "Variables" et réagit : "Ensuite on prend en compte l'autre liste. Ensuite on rajoute en fait celle-ci à celle la. Les deux listes vont se mettre ensemble (regarde variables) ce qui va provoquer qu'il y a qu'une liste au final". Il constate que la première liste pointe vers la deuxième et qu'aucune autre liste n'est modifiée ou créée avec les nouveaux éléments.

Lors de l'exécution de la deuxième fonction, il dit : "Du coup après on appelle les fonctions, on tourne dedans, mais va y avoir un problème car c'est une liste dans une liste. Ça va donner la liste et pas les éléments de la liste. Le count va compter le 1920, mais pas les deux autres". Il a effectivement détecté le problème grâce à la visualisation des variables.

Pour le code la deuxième fonction, il remarque les flèches indiquant sur quel élément de la liste nous sommes dans la boucle for. Il dit : "c'est l'élément sur lequel on est".

Comme remarque finale sur l'outil et les visualisations rencontrées il dit "Non tout m'a semblé logique. Franchement ça pourrait aider car au premier quadri on nous a jamais dit comment utiliser le debugger etc et on est pas vraiment mis au courant de à quoi ça sert. On est plus en mode c'est un peu magique."

### 4.4.2 Étudiant 5

Pour le **premier programme**, le cinquième étudiant ne montre aucune hésitation et répond toujours correctement jusqu'à la fonction append(). Il rajoute le "19" dans la liste de la **variable4** mais pas de la **variable6**.

Face aux visualisations colorées des types, il est agréablement surpris. Pour les **variables1 et 2** il se contente de lire le type final des variables étant donné que les expressions sont assez simples. Pour la **variable3**, il dit : "Variable3 c'est plus utile c'est Int et Int mais le total c'est Bool donc ça renvoie True. C'est pratique ça en vrai !".

Arrivé à la **variable4**, il est de nouveau positif concernant les types de chaque élément de la liste : "Variable4 ici de nouveau on précise chaque valeur c'est pas mal". Il regarde aussi la partie "Variables". A la question : "Tu sais ce qu'est un objet ? ", il répond : "Variable c'est quand on attribue un seul truc, objet c'est ce qui est tupple, dictionnaire et liste". C'est effectivement ce qu'on retient quand on a pas encore fait de programmation orientée objet en Python.

Pour la **variable5**, il dit : "ici c'est intéressant de voir qu'en bleu on sait que c'est un string et qu'au final ça devient du Int globalement. Je trouve ça sympa de préciser ça". L'instructeur lui dit qu'un étudiant précédent disait de ne laisser qu'une seule couleur par expression pour ne pas embrouiller l'utilisateur. Il répond : "Moi je trouve ça intéressant car on sait qu'un input renvoie forcément un str, mais on sait qu'en mettant un Int et si on y prête pas forcément attention l'utilisateur il va se dire mais ça renvoie une chaîne de caractères alors pourquoi ça renvoie un Int ? Avec les couleurs on comprend le comment du pourquoi". Il dira aussi plus tard, sur le même sujet : "je trouve ça surtout intéressant dans les cas on en affiche plusieurs, comme ici avec Float et Int. Moi je me disais à chaque fois oui ça renvoie une chaîne de caractères mais pourquoi on le transforme la ou pourquoi est ce que ça change la ? Alors que la on le voit vraiment bien ! On a la réflexe de se dire celui la il est Int mais comme ça c'est tout du Float, et bien il prend du float".

A la **variable6**, il dit : "Ici c'est sympa on fait la liaison avec les deux listes". L'instructeur demande : "Sans voir la visualisation tu aurais créé une nouvelle liste ou tu aurais pointé vers la même ?". L'étudiant répond : "Je pense intuitivement j'aurais créé une nouvelle liste, la même". Face à l'exécution de la fonction append() : "Ça l'ajoute dans la liste et donc partout" en voulant dire que cette modification affectait les deux variables.

Une fois à la conditionnelle de la **variable7** : "Ici du coup, on dit que c'est faux mais c'est pas spécifié pourquoi c'est faux. La condition est pas respectée donc on va forcément dans le else. Variable7 on lui attribue "non" du coup". Le code grisé ainsi que la croix rouge semblent évidents. Il est cependant étonné qu'il n'y ait pas d'explications supplémentaires concernant la garde évaluée à False.

Face aux visualisations des fonctions, à la **variable8**, il dit : "C'est par ici qu'on rentre et ce qu'on renvoie ici, ça semble logique". Il dit aussi concernant l'expression d'addition des deux termes : "Terme1 et terme2, c'est intéressant d'avoir en rouge le terme2 car c'est Int, et toute l'expression en Float car forcément le Float est au dessus du int". L'instructeur lui demande pourquoi les visualisations de fonctions sont une fois grisées et une fois noires, il répond : "La fonction est grisée car on la définit mais on a pas fait appel à la fonction". Il dit aussi : "C'est bien d'avoir les variables locales à chaque fonction sinon on s'embrouille".

L'étudiant comprend rapidement l'objectif du **deuxième programme** lors d'une première lecture sans visualisations : "Donc ici il permet de définir si une année est bissextile et ici il affiche le nombre d'années bissextiles c'est ça ? Il ajoute à chaque fois 1 dans la boucle".

Il constate et décrit ensuite les visualisations une à une sans embûches et en montrant un intérêt positif. Lors de l'utilisation de la fonction append(), il dit : "Ici year list last century, on reprend.. ha oui il affiche ici qu'il ajoute dans l'autre liste, et que c'est de nouveau toujours la même liste et qu'il en fait pas une nouvelle ! C'est intéressant de savoir ça car on pourrait croire qu'il va créer une autre liste en mettant un élément en plus !". Il est étonné du comportement du append() sur les deux listes et il trouve la visualisation très intéressante et intuitive.

Quand il arrive à l'exécution de la deuxième fonction, il dit : "Donc ici on fait appel à la deuxième fonction, celle qui compte. On définit le compte à 0. Du coup on a year_list, c'est la liste avec les variables. Et la ça va renvoyer quelque chose ici je suppose ?". Il continue de faire défiler les feuilles et il se rend compte que l'exécution pas à pas continue dans les fonctions : "Haaa dans la fonction aussi ! je pensais que ça allait juste renvoyer quelque chose. Donc la boucle, on voit aussi une petite flèche pour montrer par ou il commence. Il va s'arrêter sur chaque, et on appelle l'autre fonction, on vérifie que chaque année est bissextile en fait !". Il constate qu'à chaque itération, la première

fonction est appelée avec l'année sur laquelle on se trouve. Il aime également la petite flèche montrant sur quel élément de la liste nous nous trouvons dans l'itération.

Il pose la question : "Ça va être des cubes qui vont apparaître l'un derrière l'autre ? Ou ça va être un cube interactif ou on va cliquer et ça va se mettre à jour ?". Il s'agit d'une question intéressante. Comme le prototype est limité au papier, les fonctions s'agrandissent simplement lorsqu'elles sont exécutées à l'endroit ou elles se trouvent initialement. Dans la réalité, les fenêtres de fonction apparaîtraient de la même manière que dans l'outil Thonny. L'utilisateur peut déplacer ces différentes fenêtres à la souris.

Avant de tomber sur l'erreur, l'instructeur lui demande s'il pense que le programme va se terminer normalement, afin le faire réfléchir. Il dit : "C'est deux listes différentes, ha non, une grande liste, avec une autre liste dedans, donc va falloir aller chercher l'élément 1 de cette liste la". L'instructeur lui demande si ça sera bel et bien le cas, il répond : "Non, du coup il va prendre la liste [2000, 2017] au lieu de 2000 et 2017. Ici il renvoie une erreur forcément, year correspond à une liste et ça ne convient pas. Il renvoie l'erreur ici dans le terminal". Sans le lui faire remarquer qu'il y avait une erreur, il ne ne serait pas douté que le programme n'allait pas se terminer correctement. Il a tout de même pu la trouver par lui même en constatant le contenu de la liste.

La discussion finale concernant son avis sur l'outil a été très enrichissante. En effet, il explique avoir eu beaucoup de mal avec la programmation lors du premier quadrimestre. Il dit avoir travaillé dur lors du blocus ce qui lui a permis de réussir son examen avec 13/20, en grande partie grâce à ses points concernant les mini-projets. Au début du deuxième quadrimestre, ils ont fait des groupes et ont reçu les consignes du projet. Ils pensaient ne jamais y arriver car leur niveau semblait bien trop faible et ils étaient perdus face aux consignes. Il dit : "J'ai cherché sur Youtube pour trouver quelque chose pour apprendre mais je n'ai rien trouvé". Il explique qu'un tel outil serait intéressant au premier quadrimestre car il était très dur de diagnostiquer les programmes : "Je ne sais jamais dans le long code de notre jeu qu'on a du faire, il fallait rechercher des variables dans le dictionnaire. Je ne savais jamais quelle variable j'allais rechercher donc je les printais à chaque fois en dessous. Je n'ai jamais eu le réflexe d'utiliser le debugger. Donc même pour apprendre ça peut être super bien. Au début c'est compliqué on comprend pas pourquoi il va chercher la, pourquoi il assigne la etc. Je pense qu'au premier quadri ça m'aurait vraiment aidé. On a eu au Q1 un exercice comme ça mais je ne l'ai jamais réussi et je ne savais pas pourquoi". Ce point correspond surtout à l'utilisation du debugger qu'ils n'apprennent apparemment pas à utiliser.

Pour la question des couleurs de types, il dit : "Je trouve ça surtout intéressant dans les cas on en affiche plusieurs, comme ici avec Float et Int. Moi je me disais à chaque fois, oui ça renvoie une chaîne de caractères mais pourquoi on le transforme la ou pourquoi est ce que ça change ici ? Alors que la on le voit vraiment bien ! On a le réflexe de se dire celui-la il est Int mais comme ça c'est tout du Float, et bien il prend du Float au final". L'étudiant est très positif concernant les couleurs quand elles sont utilisées dans des expressions plus complexes résultantes d'opérations entre différents types.

Pour les fonctions, il dit : "Les fonctions c'est top de voir qu'elles s'actualisent. Par exemple pour le count ou ce qui rentre et sort des fonctions en grisé et en noir. C'est vrai qu'au début on faisait une seule fonction et on l'appelait qu'une seule fois dans le fond mais en fait on peut les appeler partout et ça effectue le même morceau de code donc c'est vrai que c'est intéressant de voir pourquoi avec ce morceau de code la ça fonctionne et avec ce morceau de code la ça ne fonctionne pas".

### 4.4.3 Deuxième analyse - Conclusion

Les observations faites durant cette deuxième analyse sont très positives. Les deux participants, ayant déjà terminé leur projet de programmation, avaient un niveau relativement trop élevé par rapport au public cible de ce travail. Cependant, les visualisations ayant déjà été testées lors du premier test et améliorées ensuite ont semblé très claires lors de ce test ci.

Le fonctionnement des visualisations liées aux types, le point le plus important du travail ont toujours été très bien adoptées par les étudiants sans besoin d'explications. Le deuxième étudiant était particulièrement motivé par ce point et dit que cela aurait été très utile au premier quadrimestre afin de comprendre certaines erreurs.

La visualisation liée aux fonctions semble également claire aux yeux des étudiants. Les points ajoutés à cette partie après le premier test (agrandissement de la boite, contenu des variables locales) ont été correctement compris et jugés utiles. Ce concept met en lumière le déroulement de l'exécution des fonctions, surtout quand elles sont appelées récursivement, dans des itérations ou qu'elles s'appellent entres-elles. Le système de debugger fusionné avec ces visualisations en forme de boite est efficace.

La principale fonctionnalité ajoutée pour ce test utilisateur est le découpage de la partie "Variables" en deux parties. Après une petite explication, les étudiants comprenaient son objectif et s'y référaient pour résoudre les exercices.

# Chapitre 5

# Discussion

## 5.1 Résultats

La première question de recherche définie dans la section 2.3 était : "Comment la visualisation de programme peut t-elle assister les étudiants de première année dans la compréhension des différents concepts du développement avec Python ?". Après les différents tests et analyses, il est possible de répondre partiellement à cette question, tout de même assez large, en répondant aux différentes sous-questions.

### 5.1.1 Comment rendre visibles les types de variables et expressions avec la visualisation ?

Le solution utilisée ici a été d'appliquer directement des couleurs et des formes sur le code. Cela permet à l'étudiant de visualiser les types directement sans devoir aller consulter une autre section sur son interface. Cette solution a été adoptée très positivement par les différents étudiants. Elle permet également d'ajouter de manière détaillée tous les types des différentes parties d'une expression. Cela est le point fort de la solution car en plus de connaître le type global d'une expression, cela peut potentiellement permettre à un étudiant de comprendre son fonctionnement plus en détails (voir figure 4.8 en exemple).

### 5.1.2 Les visualisations et couleurs choisies permettent t-elle aux étudiants de reconnaître les types des différents éléments d'un programme ?

La réponse à cette question est donc oui. La visualisation permet effectivement de reconnaître facilement les types des différents éléments dans le code. Cependant, cela ne veut pas spécialement dire que les étudiants en tiennent compte lors de l'exécution pas à pas d'un programme. Certains étudiants lors des tests ont tout de même voulu exécuter certaines opérations alors que les couleurs montraient clairement que les types ne correspondaient pas. Les connaissances de base concernant les opérations et concepts de programmation restent évidemment primordiales à la bonne compréhension/écriture d'un programme. Le fait de visualiser les types réduit donc grandement la marge d'erreur mais pas totalement.

En plus d'aider l'étudiant à comprendre certaines erreurs, la visualisation des types peut être utilisée comme une certaine documentation. En effet, les couleurs peuvent donner le type de retour d'une fonction que les étudiants ne connaîtraient pas par exemple. Lors du premier test utilisateur, cela est arrivé plusieurs fois avec la fonction input(). Les étudiants qui pensaient que le type de retour

de cette fonction dépendait de l'entrée de l'utilisateur au clavier ont pu remarquer d'eux-mêmes qu'il s'agissait d'une erreur.

Il faut aussi, lors de l'enseignement, être sur d'avoir donné les bases théoriques nécessaires aux étudiants afin qu'ils puissent reconnaître les types par eux-mêmes ensuite. Le risque sinon serait d'avoir des étudiants qui ne se réfèrent qu'au debugger avec visualisations pour trouver leurs problèmes de typage.

### 5.1.3 Comment rendre visuels les concepts liés aux fonctions et conditionnelles ?

Pour les fonctions, il a été évident d'utiliser le concept des boites indépendantes qui peuvent avoir des entrées et une sortie. Ce concept a déjà été vu dans certaines descriptions d'algorithmes qui prennent certaines valeurs en entrée et retournent un certain résultat. Ce concept n'est pas étranger aux étudiants et parait simple à comprendre pour ces derniers. Certains outils de visualisation existants, tels que Thonny, ouvrent simplement une nouvelle fenêtre affichant le code de la fonction lors de son appel. Pour un étudiant ayant tout juste découvert le principe de fonction, une telle visualisation n'aiderait pas spécialement sur la sémantique même d'une fonction.

En effet, l'ajout des flèches pour indiquer les entrées et sorties a semblé parlant aux étudiants qui ont tous directement compris le fond de la visualisation. Contrairement à Thonny, les fonctions peuvent déjà être visualisées une fois qu'elles ont été définies. Cela montre à l'étudiant que ce "morceau de code" grisé est disponible sur le coté afin d'être utilisé. Ce changement de couleurs et de taille lors de l'appel d'une fonction a aussi été bien adopté par les étudiants.

Pour bien faire, des tests auraient du être effectués lors du premier quadrimestre car c'est à ce moment la que les étudiants doivent intégrer le principe de la fonction. Les flèches et animations (couleurs et zoom) auraient été plus importantes que pour les étudiants du deuxième quadrimestre, déjà à l'aise avec ces concepts. Cependant, le fait que tous les étudiants du deuxième quadrimestre ayant passé les tests aient eu des avis positifs concernant ces visualisations conforte sur leur pertinence et efficacité.

Pour les conditionnelles, concept plus basique que les fonctions, il a été choisi d'indiquer si la garde est évaluée à True ou à False et de griser les morceaux de code qui ne seront pas exécutés. Cette visualisation était facile à mettre en place et apporte moins d'informations que les visualisations liées aux types et fonctions. Elle peut cependant aider à comprendre le concept à un étudiant qui rencontre pour la première fois les conditionnelles.

Les seules visualisations pré-existantes supportant le concept des conditionnelles sont celles de Collece-2.0 indiquant clairement qu'un if peut être vu comme un embranchement avec deux choix possibles en fonction de la condition. Il y aussi Jgrasp qui affiche un diagramme d'activité à côté du code, ce qui montre qu'un seul chemin sera emprunté. Collece-2.0 offre des visualisations totalement différentes de celles visées dans ce projet ci, qui sont directement affichées dans l'IDE. Les diagrammes d'activité de l'outil Jgrasp ne seront probablement pas compris directement par les étudiants de première année qui n'ont encore jamais vu de langages de modélisation. Les visualisations proposées dans ce travail semblent être le meilleur compromis dans le cas des novices de première année.

### 5.1.4 La séparation heap/stack est t-elle efficace lors du débogage d'un programme ?

Dans un premier temps, aucune question de recherche n'avait été définie concernant la séparation des parties stack et heap dans l'affichage des contenus des variables. La volonté d'ajouter ce concept à l'outil est en effet arrivée plus tard, lors de l'amélioration du prototype à la suite du premier test utilisateur.

Ce concept déjà existant (Online Python Tutor [6]) a révélé être d'une grande utilité dans les exemples de programmes créés pour le test. Il semble très adapté pour appuyer la matière apprise lors du premier quadrimestre à l'Unamur. Les étudiants utilisent tous des structures de données à ce stade mais aucun ne connaissait le concept de partage de référence (ou le fait qu'une liste puisse contenir une autre liste avec la fonction append() par exemple). Assister cet apprentissage avec ce genre de visualisations serait bénéfique. En effet, cette fonctionnalité n'ajoute que du positif par rapport au simple affichage des contenus des variables. Par contre, si les étudiants n'ont jamais entendu parler du concept d'objet, cela risque des les embrouiller. Il faudra introduire ces visualisations avec des explications concernant la notion d'objet. Les concepts d'objets immuables et muables devront également être clairs, ainsi que la possibilité d'effectuer des partages de références. Ces concepts sont de toute façon importants à enseigner au premier quadrimestre étant donné que les étudiants utilisent beaucoup de listes lors de leur projet de programmation.

### 5.1.5 Visualisations affichées en temps réel

Lors du premier test utilisateur, une question avait été posée aux trois étudiants concernant l'affichage des visualisations de types en temps réel (sans avoir besoin de lancer le debugger) dans l'IDE. Il s'est avéré que tous les étudiants trouveraient ça utile. Il faudrait cependant pouvoir le désactiver afin que ça ne pollue pas trop l'interface de l'IDE. Les étudiants pourraient vite être distraits et déconcentrés si les couleurs apparaissaient à tout moment. Les couleurs prennent effectivement beaucoup de place, comme il est possible de le constater avec la figure 3.6. Cette fonctionnalité serait plus complexe à mettre en place car il faudrait parser le code en temps réel afin d'analyser la syntaxe pour reconnaître les types. Cela serait également plus lourd pour l'ordinateur.

## 5.2 Limitations du travail

Des exemples de visualisations ont été développés et testés au cours de ce travail mais, quels que soient les résultats, cela n'est pas suffisant afin d'assurer un enseignement qualitatif de la programmation. Comme indiqué dans l'état de l'art à la section 1.2.5, utiliser des outils de visualisation de programme lors des activités d'apprentissage ne garantit pas une compréhension optimale de la matière par les apprenants. Il faudrait intégrer l'outil et ses visualisations dans un processus pédagogique efficace, qui respecterait par exemple les 16 principes de l'apprentissage expliqués à la section 1.2.5. Cela garantirait un engagement suffisant de la part des étudiants dans les activités afin qu'elles soient optimales.

La pertinence des analyses n'est pas réellement vérifiée à cause de l'échantillon trop petit d'étudiants ayant participé aux tests utilisateurs. Seulement cinq étudiants au total ont pu tester l'outil, et seulement deux la dernière version. Le fait que les étudiants aient été interrogés au deuxième quadrimestre joue également fortement sur la pertinence des tests. Réaliser ces tests sur des étudiants du premier quadrimestre aurait peut être permis de déceler plus de défauts dans les visualisations.

Les tests utilisateur réalisés se sont fait à partir de programmes exemples assez basiques et simples. Aucunes constructions comprenant de la syntaxe plus complexe n'a été utilisée. Il faudrait que l'outil puisse fonctionner sur n'importe quel code, ce qui n'est pas encore le cas. Prenons la compréhension de liste comme exemple :

```
new_list = [function(item) for item in list if condition(item)]
```

Ceci englobe une assignation, une boucle et une conditionnelle en une ligne. Qu'afficher comme visualisations dans cette exemple ?

# Conclusions

## 5.3 Rappel de la problématique

Pour les débutants, la programmation est une matière relativement compliquée à appréhender. L'objectif principal de ce travail était d'explorer et expérimenter la visualisation afin de développer un outil d'animation de programme pouvant assister les étudiants dans la compréhension de certains concepts. Ces concepts sont l'allocation de la mémoire, les types de données, les fonctions et les conditionnelles. Le fait que les étudiants aient des difficultés concernant ces concepts a été confirmé lors du premier test utilisateur. Un rappel de ces difficultés peut être retrouvé à la section 4.2.1. Les étudiants avaient encore des gros problèmes de compréhension quand ils reconnaissaient les types de différentes expressions. Même si les concepts ciblés étaient matière du premier quadrimestre, les étudiants avaient encore des grosses lacunes au milieu du deuxième. Cela montre à quel point ces concepts ne sont pas si facile à aborder. Il est donc pertinent d'assister les étudiants dès le départ afin de les aider à construire des modèles mentaux robustes de chaque concept.

## 5.4 Démarche

Une première version du prototype a été testée avant les vacances de Pâques avec trois étudiants. Ce test vérifiait les versions de base des visualisations des types, fonctions et conditionnelles. Une amélioration de l'outil a ensuite eu lieu notamment grâce aux analyses du premier test. Un deuxième test a été fait après cela afin de valider les différents changements effectués. Les programmes exemple de ce deuxième test étaient également plus grands et complexes, et la notion de de stack/heap a été ajoutée à la partie "Variables" de l'interface.

## 5.5 Résultats

Le prototype d'outil créé lors de ce travail a été une combinaison des différentes visualisations liées aux différents concepts (voir figure 3.7 pour rappel). Les résultats concernant la compréhension des visualisations par les étudiants sont très positifs. Les visualisations concernant les types se sont avérés être les plus utiles lors du premier test utilisateur, les concepts de fonctions et conditionnelles étant déjà bien intégrés à ce stade. Les visualisations concernant ces dernières ont cependant été très bien adoptées par tous les étudiants. La séparation des stack/heap lors de l'affichage des contenus des variables a également été utile pour les étudiants lors du deuxième test utilisateur. Cela mettait en évidence des partages de références, ou des listes en contenant d'autres. Ces concepts ne semblaient pas clairs ou connus pour les étudiants, même s'ils avaient déjà terminé leur projet de programmation.

En plus de la partie technique, le fait de travailler avec des visualisations semblait décomplexifier la programmation et rendre l'exercice plus ludique. Un étudiant disait que ce genre d'outil l'aurait aidé à se motiver au premier quadrimestre (voir section 4.2.4).

Le prototype et les tests ont montré que la visualisation de programme, et particulièrement l'animation de programme avec un debugger visuel pouvait assister les étudiants dans l'apprentissage de la programmation, à condition que ces outils soient utilisés dans un cadre pédagogique efficace.

## 5.6 Perspectives

Les visualisations et fonctionnalités inclues dans l'outil forment évidemment une liste non exhaustive de tout ce qu'on pourrait imaginer dans ce domaine. En fonction de tests plus poussés dans le cadre des cours de programmation, d'autres fonctionnalités et visualisations (même déjà existantes dans d'autres outils) pourront être ajoutées.

En effet, certains concepts ne sont pas couverts par ce travail. Pour citer quelques exemples, il y a la visualisation des structures de données. Dans la partie "Variables" de l'outil se trouvent les structures de données comme elles sont affichées dans un IDE, c'est à dire sans aucune mise en forme. Des visualisations plus ludiques et métaphoriques pourraient être imaginées. Si nécessaire, une visualisation dédiée aux boucles pourrait aussi être intégrée. Cela montrerait clairement sur quels éléments nous itérons dans le cas d'une boucle for par exemple. Pour l'instant, le debugger montre simplement à l'aide de la flèche si l'on repart pour un tour de boucle ou non.

Comme cela existe dans l'outil Thonny, au fur et à mesure de l'exécution du programme, les différentes expressions pourraient être évaluées unes à unes. Pour l'instant, seules des informations de types sont affichées en détails sur les lignes de code. Les seules expressions étant évaluées sont les gardes des conditionnelles.

L'étendue de ce travail a été limitée à de la programmation procédurale. Cependant, il serait très intéressant de poursuivre le concept des visualisations de types avec des programmes orientés objets. Des nouvelles visualisations pourraient être introduites de façon à visualiser les différentes classes et objets ainsi que d'appeler des fonctions.

Seule la partie IHM a été travaillée dans ce projet. Aucun développement n'a réellement eu lieu afin d'intégrer les visualisations à un IDE. Une fois testé d'avantage, le prototype devra être développé de manière à être interactif et à réagir automatiquement au code qu'on lui fourni.

# Annexes

## Annexe A : Fragments de texte du premier test utilisateur

| **Étudiant 1** |
|---|
| (Ex1 variable1) : La variable 1 = 10 ça me semble être une integer et le contenu ben c'est 10 |
| (Ex1 variable2) : On met input entrez une valeur, l'utilisateur rentre 10.5, mais comme tout est entre guillemets normalement ça sera . . . J'suis pas sûr que si c'est entre guillemets ça reste integer ou bien String. |
| (Ex1 variable3) : C'est encore input mais cette fois on précise integer. Ça me convint un peu que le précédent était un string |
| (Ex1 variable4) : 10 oui mais quel type c'est ? (réfléchit) je mets integer |
| (Ex1 variable5) : C'est la même chose mais c'est entre guillemets donc je mets un string |
| (Ex1 variable7) : 10.5 + 12 integer |
| (Ex1 variable9) : Variable 9 c'est var8 + var7, mmmh je ne sais pas si ça va marcher |
| (Ex1 variable10) : Je ne suis pas sûr que ça fonctionne quand tu ajoutes 5 à ça. . . (passe l'exo) |
| (Ex2 variable1) : C'est facile à voir du coup ça c'est float |
| (Ex2 variable2) : Input entre une valeur je vois bien que c'est bleu donc le résultat sera d'office un string en fait |
| (Ex2 variable3) : Entrez une valeur, donc forcément je sais du coup quel est le type, ici elle est booléenne |
| (Ex2 variable4) : 48==47+1 c'est booléen (réfléchit) je me suis trompé, c'est True ou bien false. |
| J'ai téléchargé une extension en python pour les déplacements avec les tab pour voir s'ils sont bien alignés et ça met des couleurs à chaque fois, ça m'a fait penser à ça, depuis que je l'ai eu ça m'aide bien. |
| Oui avec le type du résultat, par exemple pour la 2 ça m'a aidé j'ai eu bon mais j'étais pas sûr, ça m'a aidé à voir directement |
| Ex2 variable9) : Il est rouge donc je sais que ça sera un int |
| (Ex2 variable11) : Heu pourquoi il est barré celui-ci ? il faut pas le faire ? ? Forcément pas possible car string divisé par integer |
| Pour les booléens c'est très utile, aussi pour le premier exo input mais là c'est une erreur de ma part, et les booléens je ne connais pas ceux-là (montre les bool) |

| |
|---|
| (Ex2 variable3) : Pour la variable 3 ça sera false car ça vaut 10 et 10 est plus petit que 50 donc ça sera False. |
| (Ex1 variable10) : Hmmm je ne suis pas sur le 5 si je dois l'appliquer au 10 avant ou après. (les parenthèses) |
| (Ex1 variable10) : Alors tu vas devoir faire True + 5 ?<br>Prof : explique<br>Tu m'as appris quelque chose ! Je ne le savais pas avant de faire celui la |
| (en parlant de l'exo 2) : C'est clair, y'a juste pour la croix c'était pas précisé et je pensais que c'était toi qui disait de pas faire celui-là. |
| (exo3) On met 8 à num 2 Le maximum sera = à 15. (regarde le contenu les variables) |
| (Exo3) : Hein ? pourquoi il est bleu le num3, ha oui car on l'a mis entre guillemets. Ha oui donc c'est un string ce n'est pas un integer.<br>Prof : Ce n'est pas à cause des guillemets<br>Ha oui y'a pas le integer ! |
| (Exo3) : Ok donc ça compare num3 avec max et c'est un string avec un integer donc c'est pas possible oui là je comprends directement plus que c'est une erreur avec le panneau ! |
| (Exo3) : Oui c'est grisé car on l'utilise pas du coup. Puisque le cas est ici, on est dans le if pas dans le else. |
| (Exo4) : Ok, on rentre les paramètres et le résultat ici. C'est le fonctionnement de la fonction avec son nom |
| (Exo4) : Ha y'a une erreur ! Pourquoi y'a une erreur. (après 10 sec) Haaa on additionne encore un string avec un int ! |
| (Exo4) : Ouais c'est ça, sur le pc j'exécute ça ne marche pas je vais voir je dis ha oui c'est une virgule, ha oui d'accord. Si y'avait ça à l'examen ça serait très bien ! (rire) |
| (Exo4) : Elle est activée du coup elle devient grasse. Je trouve ça logique. Ça m'a aidé moins que les couleurs. Si on avait des fonctions qui s'appelaient entre elles même, ça serait utile ! |
| (Exo5) : Haa la fonction retourne un booléen en fait !<br>Prof : tu vois pourquoi ?<br>Non pas vraiment<br>Haaa oké c'est comme si on utilisait un if ! |
| (Exo5) : Prof : tu savais ce que ça voulait dire : :-1 ?<br>Non deux fois deux points je sais pas ce que ça veut dire<br>Prof : explique |

haaaa c'est pratique ça ! les petits trucs comme ça on les voit pas au cours on voit un peu les bases puis on dév avec le projet. On a tellement de restrictions qu'on a peur de google, on peut utiliser .pop, utiliser ça, mais ça on peut pas... On a beaucoup de restrictions donc on se dit que ça sert à rien d'aller voir sur internet, mais franchement ça c'est utile !

En fait c'est une qualité de vie, tu sais te débrouiller sans mais une fois que tu l'as c'est beaucoup mieux !

**Étudiant 2**

(Exo1 variable1) : Variable 1 entier contenu 10

(Exo1 variable2) : Le type n'est pas précisé, il peut rentrer un float ou ... etc. ça viendra après en fonction de ce qu'on rentre

(Exo1 variable3) : Ici donc c'est 100 et integer

(Exo1 variable4) : Ici c'est un entier qui vaut 10 > 50 Juste la variable 4 ça se peut être un entier comme réel ou float j'ai mis directement réel, car tout dépend il peut donner ... mais en fait ici c'est un seul exemple donc c'est pas l'utilisateur qui va donner une variable. Ici ça deviendrait un entier car c'est un 10

(Exo1 variable5) : 100%90 > 50, le contenu va être un peu bizarre, ha non c'est bon je prends en considération les guillemets !

(Exo1 variable6) : Input entrez une valeur, ça on ne sait pas encore le type

(Exo1 variable9) : Ici ça affiche une erreur

(Exo1 variable10) : Ici aussi une erreur

(Exo1 variable12) : Ici ça va afficher une erreur, je sais pas si c'est logique ou pas

(Exo1 variable14) : Par contre la variable 14 je n'ai pas bien compris variable 13.contains

Prof : Explique

Alors c'est un booléen qui vaut vrai

(Exo1 variable13) : Ici ça va afficher une erreur car y'a une virgule après. Y'a une virgule donc on va donner soit un autre commentaire sinon une autre variable qui va prendre par exemple cet entier = , tu fermes les guillemets, virgule a

(Exo2) : Ha ça va être facile ! Franchement oui.

...

Je regarde juste les couleurs, franchement ça aide bien.

(Exo2 variable8) : Append... ça va rien afficher y'a pas l'élément 12, et ici on a bonjour, ça va pas fonctionner (confond avec contains)

(Exo2 variable10) : : Tout ça c'est un booleen, ça je savais pas ! pour moi ça c'est une erreur

(Exo1 variable10) : Le 10 pour moi c'est une erreur car c'est pas logique de mettre (5<10)+5.

Prof : Explique

| |
|---|
| Ça je savais mais les utiliser je savais pas, c'est nouveau pour moi |
| (Exo2 variable9) : Ça va nous donner 0 comme réponse alors |
| (Exo1 variable11) : 10+5 = 15 et 15 est supérieur à 14. Donc ça vaut bien True |
| (Exo2 variable3) : Ha je me suis trompé ! c'est l'inverse j'aurais dû mettre False Prof : Explique que ça ne fonctionne pas comme ça. |
| (Exo2) : Oui la croix veut dire que c'est pas possible, tu peux pas diviser un str avec un bool |
| (Exo3) : Prof : Explique l'exercice et le système de debugger<br>Donc c'est comme si tu dessines un tableau, et prendre ligne par ligne de code et les variables (max, numéro1, etc) tu les prends ici et tu exécutes chaque ligne, ça va te donner quoi par rapport à ton code ? |
| (Exo3) : Le but du programme c'est de faire une comparaison, c'est le maximum entre les 3 variables(1, 2 puis 3) |
| (Exo3) : Ici ça va être saisie de données, ça va être des entiers, (défile) , ok donc c'est 15, là c'est 8. Il va prendre le max le numéro 1, le 15... |
| (Exo3) : Prof que vois-tu comme nouvelles visualisations ?<br>Le petit V qui veut dire que c'est vrai, c'est vérifié donc on a pas besoin de passer sur le Else. |
| (Exo3) : Ici c'est 45 donc le max va prendre 45 car supérieur à 15. mmh attends, ici c'est un string , donc c'est une erreur. Ha oui y'avait pas le int !ça va afficher le 15<br>Prof : Ça ne va rien afficher du tout, juste une erreur<br>Oui c'est vrai. Au début j'avais pas vu car c'était int int je pensais que le troisième aussi. |
| (Exo3) : Prof : tu as des avis sur le système des couleurs ? Juste un point de vue, au lieu de mettre tout en bleu avec le rouge (il parle des input qui sont castés en int). Ça serait plus utile de mettre la couleur finale. Par exemple ici, vert et bleu c'est un booléen, je laisserais tout en vert. Par contre si c'est juste un simple input, en bleu directement. Ça peut prêter à confusion, c'est mon point de vue Par contre c'est trop utile ! |
| (Exo4) : Oké, donc c'est rajouter terme1 + terme2, la fonction sert à calculer la somme des deux. Donc ici le terme2 est à 10 . . . |
| (Exo4) : Ça c'est l'entrée et la sortie de la fonction, avec ce qui rentre ce qui sort. |
| (Exo4) : Alors la réponse . . . Ici ça va afficher une erreur ! (L'a vu grâce à la visualisation). Ha oui c'est ça la remarque que j'ai fait tout avant pour la virgule si tu te souviens. Print c'était normalement la réponse du calcul est, après une virgule, la fonction. . . la ça aurait fonctionné, ici avec le + ça fonctionne pas |
| (Exo4) : Prof : que penses-tu du panneau d'erreur ?<br>Ça n'affiche pas l'erreur en elle-même, le type de l'erreur par exemple, ça va aider beaucoup plus. Parce que pour un débutant tu dis y'a une erreur ici il va dire non c'est logique. |

Prof : Tu l'afficherais où ?

Terminal par exemple ici ça serait erreur numéro 1, ici dans le terminal tu reprends le 1 et tu affiches qu'additionner un int + un str c'est illogique. (Propose de légender les erreurs)

(Exo4) : Prof : tu as vu la différence entre ici et ici ? (Montre la visualisation de définition et appel de fonction).

Pour moi ça ici on est dans la fonction, alors qu'ici pas. Ça va prendre terme1 terme2 ça va faire la somme des 2 et ça va sortir mais ici on rentre dans la fonction parce qu'on fait un appel, voilà c'est ça

(Exo5) : Ha oké le mot est égal à test. Ta fonction permet d'inverser le mot, ça va renvoyer toujours un string. Donc ici on est pas dans la fonction donc c'est logique, après on a la valeur string le mot c'est le test. Après on fait un appel pour la fonction ça devient noir ok.

(Exo5) : Prof : Que te disent les couleurs ici ?

c'est un booléen c'est donc true, donc ça va pas être exécuté. Donc c'est == la chaine, les mots sont égaux, mais ici il manque un truc dans la condition non ? (A du mal à comprendre le return booléen.)

(Exo5) : J'aime bien le panneau (en parlant de la croix)

Prof : Explique que ce n'est pas un panneau et que ça veut dire que la condition est fausse.

ha donc le panneau c'est pour les erreurs, et la croix ... Ok. (a confondu la croix et le panneau d'erreur)

(Exo5) : Prof : Tu vois que du code est grisé ici, tu sais pourquoi ?

Car ça va pas être exécuté on va passer directement à la dernière (montre le else)

Prof : Tu aimerais bien avoir ces couleurs en temps réel ?

Oui personnellement oui, mais trop de couleurs ça va gêner un peu. Ça te facilite le codage, mais trop de couleurs ça me gêne un peu.

Prof : On devrait pouvoir l'activer quand on veut alors ?

Oui voilà car quand on a 100 lignes de code... Si on a un bouton activer désactiver ça serait mieux Par ex t'as une erreur, tu peux voir les types, tu sais ce qu'il se passe. Par ex dans le projet de progra on avait un problème avec les coordonnées x et y, il la prend comme une liste mais la virgule entre les deux il la prend aussi comme un type dans la liste, au début on avait une erreur. Si on avait ça on saurait directement.

Une idée, pour les croix et le panneau,il y a aurait moyen d'ajouter ce que ça signifie, car personnellement j'ai confondu .

**Étudiant 3**

(Exo1 variable1) : Bah ici variable 1 c'est un numéro donc je sais que c'est un Int

(Exo1 variable2) : Variable 2 c'est input ou on entre une valeur et on entre 10.5 donc je dirais que c'est un heu.... Float et le contenu c'est 10.5

(Exo1 variable3) : La variable3 c'est un input avec un Int et on entre 100 donc c'est int et 100

(Exo1 variable4) : Variable 4 c'est 100 divisé par 90 qui doit être plus grand que 50 donc je dirais que c'est un ... bah ça ferait un float mais le problème c'est que la condition serait pas très bonne car ça serait jamais plus grand que 50 le reste. J'aurais tendance à dire que ça serait un bool, de dire que si c'était possible oui ou non de ... je dirais bool et comme le reste de 100%90 c'est plus petit la réponse c'est no (veut dire false)

(Exo1 variable5) : La 5 je vois pas du tout ce que c'est, le fait qu'il y ait des guillemets. Ça veut juste dire que ça va renvoyer le contenu total. Je sais plus comment ça s'appelle, c'est du string ? ça va renvoyer 100 % 90 >50

(Exo1 variable6) : La variable 6 c'est ou il faut rentrer la valeur du reste entre 10 et 3, ça fait 1. Ça va être un Int. On pourrait mettre float pour être sûr mais je mets int.

(Exo1 variable7) : Variable 7 c'est 10.5+12 donc c'est un float et c'est 22.5

(Exo1 variable8) : Variable 8 c'est un str et ça renvoie bonjour

(Exo1 variable9) : Variable9 ça renverra bonjour et 22.5 et il me semble que ça sera un string parce que si y'a des caractères c'est forcément un string. Je sais pas si je dois écrire comme c'est écrit, bonjour + 22,5

(Exo1 variable10) : Donc heu variable10 c'est 5 < 10 ce qui est vrai, + 5 ca fait 10.

Prof : Vrai + 5 ça fait 10 ?

Ha non c'est bizarre .. J'aurais tendance à dire que c'est une erreur. Car un bool + un int c'est pas possible.

(Exo1 variable11) : La 11 ça dépend car je sais pas si c'est 14 <10 ou 14 < 10+5. Je dirais que comme y'a pas de parenthèses c'est 14 <10 et puis + 5 après. Roh j'hésite.

...

La variable 11 vu que je savais pas je sais qu'elle est possible haha. À moins que ça renvoie encore une erreur. Là je dirais que la 11 c'est un bool et c'est vrai.

(Exo1 variable12) : La variable 12 ça dirait vrai + 50. C'est pas possible.

(Exo1 variable13) : La 13 ça renvoie bonjour puis une virgule... heu je sais plus ce que ça fait quand c'est comme ça,

(Exo1 variable14) : Variable 14 c'est 13 contains bonjour... bah la ici c'est une espèce de condition ou on dit que la var 13 doit contenir bonjour. Donc c'est vrai, donc c'est bool

(Exo1 variable15) : Variable 15 c'est la var8 + dora donc c'est bonjour dora. C'est tous les deux des STR donc on peut les additionner.

(Exo2 variable1) : Le premier c'est float donc c'est float et ça renverra 11.5

(Exo2 variable2) : Le deuxième c'est écrit Str entrez une valeur c'est 10, mais ça renvoie un nb donc je dirais que c'est int mais c'est ça le problème (voit la couleur)
Prof (à la fin de l'exercice 2) : explique que input() renvoie un string.

(Exo2 variable3) : Bool input entrez une valeur comme j'avais dit avant c'est vrai ou faux donc ici c'est faux parce que le reste ça fera 10

(Exo2 variable4) : Ici ça sera un bool en vert on a 48 == 47+1 c'est vrai.

(Exo2 variable5) : Vu que c'est entre guillemets ça veut dire que c'est la même chose, ça sera un string comme on peut le voir

(Exo2 variable6) : Ici encore même cas qu'au-dessus, mais je dirais que juste on a pas mis le petit int ou float devant. On pourrait mettre int(input())...

Prof (à la fin de l'exercice 2) : explique que input() renvoie un string.

La 6 ça sera aussi un string et doit mettre en guillemets pour dire que c'est une chaîne

(Exo2 variable7) : La var 7 c'est la var1 et c'est 11,5 avec 50 qui est un int, ça fait 61,5 et c'est un float.

(Exo2 variable8) : Ici on voit que la var 8 c'est une liste, et elle renverra bonjour.

(Exo2 variable8) : Ici c'est var8. Append ça veut dire qu'on ajoute quelque chose à la liste mais je sais pas si ça fonctionne avec les listes. Ça veut dire qu'on ajoute 12 à la liste, donc elle aura bonjour et 12.

(Exo2 variable8) : Ici c'est var8. Append ça veut dire qu'on ajoute quelque chose à la liste mais je sais pas si ça fonctionne avec les listes. Ça veut dire qu'on ajoute 12 à la liste, donc elle aura bonjour et 12.

(Exo2 variable9) : La var9 c'est un int, c'est 8 >4 ce qui est vrai, -1. Ça donnerait 7.

Prof : 8 > 4 vaut 7 ?

On vérifie que 8 est > que 7, et si c'est le cas on fait -1 ! Ça donnerait 7.

(Exo2 variable10) : La var 10 c'est comme j'avais hésité tantôt c'est un bool et faut dire si c'est vrai ou faux, je vais rester sur ma première idée ou je disais que c'était faux

(Exo2 variable11) : La var 11 ou on mélange deux types et donc c pas possible. Pas possible de diviser un str par un bool donc c'est une erreur.

Prof : Qu'est-ce que ça fait quand on additionne un boolean avec un chiffre ?
On prend la première valeur du bool et on l'additionne au reste

Prof : Tu as des remarques concernant les couleurs ?

Oui ça semble logique ! Au début j'avais du mal car je connaissais un peu et je savais que ça allait rentrer 10 mais j'avais oublié que ça allait renvoyer une chaîne de caractère mais je pense que pour quelqu'un qui s'y connaît pas du tout , il saura que si c'est un string ça renvoie direct une chaîne il se posera pas de questions.

(Exo2 variable3) : Prof : La troisième ici y'a du vert mélangé à du bleu tu vois pourquoi ?

Je dirais que le bleu c'est un string ou on va rentrer des valeurs et après ça sera pour vérifier la chaîne de caractère et la mettre en bool.

(Exo2 variable10) : C'est 14 < 15 donc ça c'est vrai

(Exo3) : Donc ici heu on demande à la personne de donner le premier numéro, on lui demande de faire un deuxième numéro

(Exo3) : On lui donne une condition que si le numéro 1 est plus grand que le numéro 2 alors on renvoie max = numéro1 ou sinon max = numéro 2

(Exo3) : Après on fait la même chose on demande une valeur au numéro 3

(Exo3) : Et la si c'est le 3 est plus grand que max alors on remplace soit num1 ou 2 par numéro 3.

(Exo3) : Donc la num1 ils disent que c'est 15, on demande à la personne. Après l'utilisateur va envoyer 8

(Exo3) : Donc ici on voit que la première condition elle et correcte, max sera = a num1. Le else est forcément oublié.

(Exo3) : C'est un string car y'a pas de int devant. Ça va renvoyer une chaîne de caractère. Ici une chaîne de caractères > qu'un int c'est pas possible, ça va forcément renvoyer une erreur.

Prof : tu avais anticipé l'erreur ?

J'avais pas vu qu'il n"y avait pas le int mais la couleur m'a montré.

(Exo3) : Prof : Que penserais-tu d'un debugger de ce genre ?

Ça serait pas mal de voir chaque étape car parfois quand on lance le projet, et qu'on voit que y'a un problème , on saurait pas dire si c'était avant ou après... ou même voir ce qui fonctionne pas. Ici on voit bien que grâce aux couleurs on voit bien qu'il y a une erreur. Je pense que ça pourrait être pas mal...

(Exo4 sans couleurs) : Donc ici au début on établit que terme1 = 50, on sait que c'est une fonction avec comme indice terme1 et terme2, on retourne terme1 + terme2. On dit que la réponse du calcul est + fonction somme terme 1, 10. Ici terme 1 = à 50 et terme2 c'est = à 10 donc je dirais que ça ferait 50 + 10. Ce qui est bizarre c'est que ça retourne directement terme1 et terme2. J'aurais préféré que ça soit à la fin pour pouvoir le réutiliser. Ha non ... (s'embrouille)

(Exo4) : Terme1 = à 50

(Exo4) : Ici on a un def avec terme 1 et 2 c'est pour pouvoir le réutiliser après.

(Exo4) : Et donc la réponse ça sera avec la réponse du calcul qui est un string, et après le int avec le terme1 +10

(Exo4) : Et ça c'est faux (l'avait pas vu) parce qu'un string + un int c'est pas possible.

Profs : tu avais vu l'erreur ?

Non j'avais pas compris haha.

(Exo4) : Prof : Que penses-tu du petit carré pour la fonction ?

Ça peut aider pour bien comprendre le return pour dire qu'on va entre ces termes la dans le code et qu'après on pourra les réutiliser quand on veut grâce au return ?

(Exo4) : Prof : Ici tout est noir, alors que là c'est gris, pourquoi ?

Parce que ça retourne rien pour le moment. Si on lance juste ça, ça fera rien. C'est après qu'on peut le réutiliser et ça devient noir.

Prof : Ça te semble logique ? Oui les flèches pour dire qu'on entre les termes et qu'après on peut les réutiliser quand elles ressortent c'est pas mal. Les couleurs c bien aussi pour dire là c'est grisé donc si on le lance ça fera rien. Quand c'est en noir ça retournera des chiffres.

(Exo5) : Donc ici palindrome c'est une chaîne, donc heu chaîne c'est heu... ça je sais plus... (connaît pas le : :-1)

(Exo5) : Ok donc la ça voudrait dire que la chaîne de base égale à l'inverse, c'est le principe du palindrome. Et puis c'est est_palindrome c'est pour dire si c'est un palindrome ou pas. On établit un = pour renvoyer le palindrome.

Prof : De quel type sera la variable est_palindrome ?

J'aurais tendance à dire que c'est un string

(Exo5) : Donc ici comme avant avec les couleurs. Si on lance comme ça, ça fera rien (parle de la fonction def)

(Exo5) : Le mot c'est un string test. Donc heu ici c'est un string ou y'a un bool donc il faut que la chaîne soit = à l'inverse de la chaîne

. Prof : Tu vois pourquoi ça retourne un bool et pas un Str ?

Heu ... non

| |
|---|
| Prof : Car y'a le double '=' c'est une comparaison |
| (Exo5) : Donc ici-là on a établi est_palindrome, nouvelle variable, c'est un bool qui contient un string (parle du call). |
| (Exo5) : Est palindrome vaut false donc forcément c'est pas le premier if, on passe directement au else. Ça affiche, n'est pas un palindrome, comme indiqué dans le terminal. |
| (Exo5) : Prof : Tu penses que ça serait utile d'avoir ces visualisations en temps réel lors des activités de programmation ?<br>Je pense que ça pourrait porter à confusion car c'est pas mal aussi dans visual studio quand on fait un def c'est pas noir on peut mettre des couleurs. Ça serait dérangeant. Par exemple quand on fait mot = quelque chose, il s'affiche déjà en couleurs donc ça pourrait confondre.<br>Prof : Alors je mettrais seulement les visualisations qu'on a vu ici, sans celles de vscode<br>A ce moment-là ça pourrait être utile<br>Prof : Mais les couleurs dans vscode ne sont pas rapport aux types si ?<br>Heu si y'a aussi par rapports aux types si on fait un def le palindrome sera en couleurs |
| En conclusion, ça pourrait être pas mal si quelqu'un s'y connaît pas et veut se lancer la dedans et a peur de coder. Je sais bien qu'au début quand je voyais des lignes de code ça me faisait un peu peur. Avec des couleurs ça pourrait rendre ça un peu plus ludique<br>Prof : ça pourrait aider le toi du premier quadri ?<br>Je ne sais pas le moi du premier quadri mais le moi avant de faire informatique. De voir des couleurs ça m'aurait peut-être plus apaisé. |
| Profs : tu as constaté des choses illogiques ?<br>Heu non, mis à part que je savais déjà grâce aux cours du q1 qu'on pouvait réutiliser une fonction mais les débutants vont l'apprendre avec le temps |

# Annexe B : Fragments de texte du deuxième test utilisateur

| **Étudiant 1** |
|---|
| (Programme1) : Variable1 c'est un int, contenu 20.5 |
| (Programme1) : Input c'est string, je rentre dans le contenu 87 |
| (Programme1) : Variable3 c'est heuuu du coup c'est aussi 3 int, la variable sera un bool car c'est faux, heu c'est vrai. |
| (Programme1) : Variable4 ça va faire une liste avec trois éléments, c'est un string un in et un booléen. |

| |
|---|
| (Programme1) : Variable 5 ça va être un string la valeur, et quand on va rentrer ça va être un int. On rentre 30 ça devient un int. |
| (Programme1) : Variable6 va devenir la même liste que le contenu de la 4. |
| (Programme1) : Du coup la si on ajoute 19 ça va devenir un string. Tu ajoutes dans la liste 4 un string entre guillemets. |
| (Programme1) : If la variable 4 c'est le deuxième élément qu'on regarde, c'est pas = à 16 donc on va voir en dessous, ça vaut 'non' et c'est un string |
| (Programme1) : Ensuite c'est une fonction on t'as deux paramètres terme1 et terme2 tu retournes terme1 + terme2 du coup la variable 8 ça va être 20,5 + 30 et ça va faire 50,50. |
| (Programme1) : Du coup float j'hésitais entre float et int, je pensais que float c'était quand on avait plusieurs nombres à virgules, mais en fait ça le fait avec un seul nombre à virgule. |
| (Programme1) : Input c'est string tout le temps |
| (Programme1) : Variable 3 c'est des int mais la valeur de la variable au final c'est un booléen. |
| (Programme1) : Variable4 c'est une liste avec un booléen, un int et un string dedans. |
| (Programme1) : La variable5 tu rentres un string et ça devient un integer |
| (Programme1) : Variable6 tu prends la même variable que la var4 du coup ça devient aussi une liste. |
| (Programme1) : Tu rajoutes une valeur (string) dans la variable4<br><br>Prof : tu aurais su que ça allait modifier les deux variables ou pas ?<br><br>Je pense que j'aurais pas pensé à variable6 et je pense que je l'ai pas noté. J'ai pas pensé parce que j'ai un peu oublié que 6 et 4 étaient égales. |
| (Programme1) : Ensuite le if c'est refusé car != de 16 et else non du coup et ça fait un string |
| (Programme1) : Ensuite la fonction, c'est les paramètres et ton return au final. Du coup ça va être un float et un int ce qui va faire un float au final. Comme ça va être un nombre à virgule. J'ai noté int car j'avais une autre idée de float au final. |
| (Programme1) : Prof : Tu vois la différence entre la fonction grise et noire ? logique ?<br><br>E : Parce qu'elle est exécutée et qu'avant elle était juste définie. |
| (Programme2) : Heuu du coup on a une liste au début, on fait une fonction avec les années bissextiles |
| (Programme2) : Du coup heu c'est une fonction pour print les années bissextiles, elle retourne le nombre d'années bissextiles. |

(Programme2) : On commence au count = 0, on boucle dans la year list qui est celle qu'on donne au final, si elle est bissextile, le compte va être + 1 au final.

Prof : Combien d'années bissextiles ? Je pense qu'il y a aura une erreur

(Programme2) : Du coup y'a une liste avec des int. On appelle la fonction ou on la fait pas encore intervenir au final

(Programme2) : Ensuite on prend en compte l'autre liste

(Programme2) : Ensuite on rajoute en fait celle-ci à celle la. Les deux listes vont se mettre ensemble (regarde variables) ce qui va provoquer qu'il y a qu'une liste.

(Programme2) : Du coup après on appelle les fonctions, on tourne dedans, mais va y avoir un problème car c'est une liste dans une liste. Ça va donner la liste et pas les éléments de la liste.

(Programme2) : Le count va compter le 1920, mais pas les deux autres.

(Programme2) : (Montre la flèche) c'est l'élément sur lequel on est

(Programme2) : Le count augmente de 1

(Programme2) : Va y avoir une erreur comme c'est une liste et pas dans un int.

Prof : Tu penses que ça t'aiderait ?

Ouais franchement ça pourrait aider car au premier quadri on nous a jamais dit comment utiliser le debugger etc et on est pas vraiment mis au courant et à quoi ça sert. On et plus en mode c'est un peu magique.

**Étudiant 5**

(Programme1) : Variable1 c'est type float et contenu 20.5

(Programme1) : Variable 2 c'est string car input renvoie forcément une chaîne de caractère

(Programme1) : Variable3 14 plus petit que 15 et donc ça fait un booléen je suppose et du coup ça renverrait True

(Programme1) : Variable4 ici c'est une liste, ça renvoie

(Programme1) : Variable 5 int input en l'occurrence ça serait plus str mais int et la valeur c'est 30

(Programme1) : Variable6 = variable4 donc ça serait de nouveau une liste

(Programme1) : Variable4.append19 on lui rajouterait un 19 du coup

Prof : Que à cet endroit la ?

Du coup, comme c'est la variable 6 et que ça c'est exécuté avant Si heuuu, comme il l'exécute dans ce sens la et ici il est en dernier sens il l'ajoute ici et ça c'est dans la variable4 ici ça s'ajoute ici aussi

(Programme1) : If variable4 1 , ça fait 15, heuuu non donc ça serait str et "non"

(Programme1) : Def somme... on créé une fonction somme et on prend en paramètres terme1 et terme2 et on renvoie la somme des deux. Ici on assigne la variable la fonction somme avec variable1 et 5, en l'occurrence 20.5 et 30

(Programme1) : Donc ici on renvoie directement la variable

(Montre la section variable) ça c'est ce qu'il s'affiche on est d'accord ?

Prof : explique que c'est le terminal et ça le contenu des variables en temps réel

L'utilisateur aura accès à ça ? Ha ça va servir pour mon hésitation ici !

(Programme1) : Variable3 c'est plus utile c'est int et int mais le total c'est bool, ça renvoie True C'est pratique ça en vrai

(Programme1) : Variable4 ici de nouveau on précise chaque valeur c'est pas mal

Prof : Tu vois ce qu c'est les objets ?

Variable c'est quand on attribue un seul truc, objet c'est ce qui est tuple, dictionnaire et liste.

(Programme1) : Variable5 : Ici c'est intéressant de voir qu'en bleu on sait que c'est un string et qu'au final ça devient du int globalement. Je trouve ça sympa de préciser.

Prof : Explique qu'un autre étudiant disait qu'il fallait mettre une seule couleur

Moi je trouve ça intéressant car on sait qu'un input renvoie forcément un str, mais on sait qu'en mettant un int et si on y prête pas forcément attention l'utilisateur il va se dire mais ça renvoie une chaîne de caractères alors pourquoi ça renvoie un int ? Avec les couleurs on comprends le comment du pourquoi.

(Programme1) : Ici var6 = var4, ici c'est sympa on fait la liaison avec

Prof : Sans faire l'exo t'aurais créé une nouvelle liste ou t'aurais pointé vers la même ? E Heuu maintenant forcément je vois que c'est la même mais ça je ne sais pas Prof : Explique partage référence

Ça l'ajoute dans la liste partout. Intuitivement j'aurais créé une deuxième liste, la même C'est très intéressant de le voir comme ça.

(Programme1) : Ici du coup (if else), on dit que c'est faux mais c'est pas spécifié pourquoi c'est faux. La condition est pas respectée donc on va forcément dans le else.

(Programme1) : Variable7 on lui attribue "non".

| |
|---|
| (Programme1) : Fonction : c'est par ici qu'on rentre et ce qu'on renvoie ici, ça semble logique. |
| (Programme1) : Terme1 et terme2, c'est intéressant d'avoir en rouge le terme2 car c'est int, et toute l'expression en float car forcément le float est au dessus du int. |
| (Programme1) : La fonction est grisée car on la définit mais on a pas fait appel à la fonction. |
| (Programme1) : C'est bien d'avoir les variables locales à chaque fonction sinon on s'embrouille |
| (Programme2) : Donc ici il permet de définir si une année est bissextile et ici il affiche le nombre d'années bissextiles c'est ça ? Entre ce qu'on rentre, il ajoute à chaque fois 1 dans la boucle |
| (Programme2) : La on créé la fonction avec le long booléen |
| (Programme2) : Donc ici on créé la deuxième fonction, toujours en grisé du coup car on y fait pas encore appel |
| (Programme2) : Ici ça s'affiche dans les variables de nouveau, une nouvelle liste |
| (Programme2) : Ici year list last century, on reprend.. Ha oui il affiche ici qu'il ajoute dans l'autre liste, et que c'est de nouveau toujours la même liste et qu'il en fait pas une nouvelle<br><br>C'est intéressant de savoir car on pourrait croire qu'il va créer une autre liste en mettant un élément en plus |
| (Programme2) : Donc ici on fait appel à la deuxième fonction celle qui compte. On définit le compte à 0. Du coup on a year list, c'est la liste avec les variables. Et la ça va renvoyer quelque chose ici je suppose ? |
| Prof : Ça va renvoyer ligne par ligne dans la fonction Haaa dans la fonction aussi ! Je pensais que ça allait juste renvoyer quelque chose |
| (Programme2) : Donc la boucle, on voit aussi une petite flèche pour montrer par ou il commence. Il va s'arrêter sur chaque, et on appelle l'autre fonction, on vérifie que chaque année est bissextile en fait ! |
| (Programme2) : Avec la croix on voit que c'est faux pour 1905 et je suppose que pour l'autre il va de nouveau rentrer dans la fonction et ici il va renvoyer... Vrai car 1920 est bissextile. |
| (Programme2) : Ça va être des cubes qui vont apparaître un derrière l'autre ? Ou ça va être un cube interactif ou on va cliquer et ça va se mettre à jour ? ? Et ici il calcule de nouveau |
| Prof : Tu penses qu'il va y avoir un soucis ?<br><br>C'est deux listes différentes, ha non, une grande liste, avec une autre liste dedans, donc va falloir aller chercher l'élément 1 de cette liste la.<br><br>Ai-je géré ce cas la ? |

Non du coup il va prendre la liste 2000

02017 au lieu de 2000 et 2017.

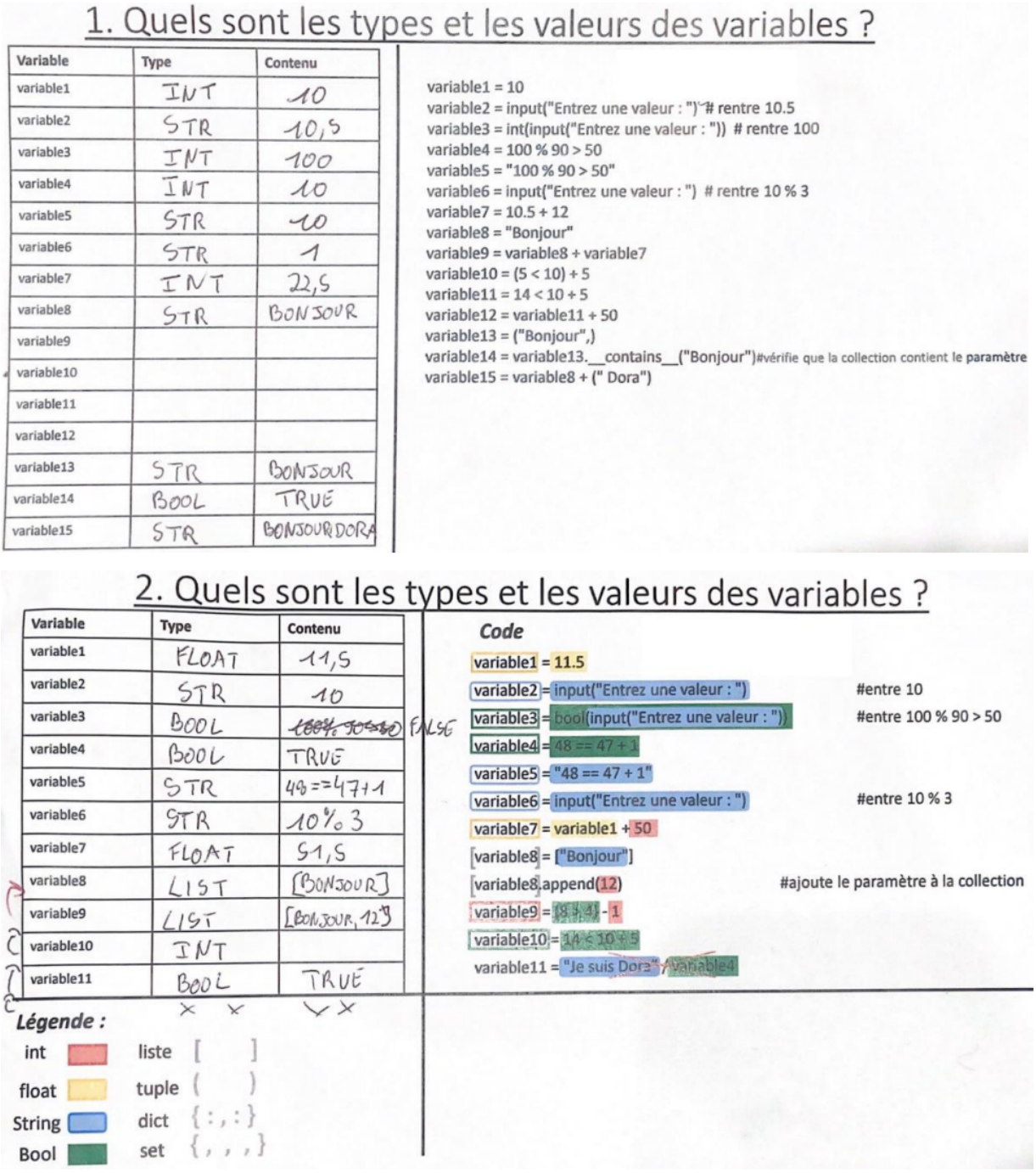
(Programme2) : Ici on voit le count qui s'est mis à 1 car l'année était bissextile.

(Programme2) : Ici il renvoie une erreur forcément, year correspond à une liste et ça ne convient pas. Il renvoie l'erreur ici (montre le booléen).

(Programme2) : Prof : Le toi du premier quadri ça te serait utile tu penses ? Je pense que ça m'aurait vraiment beaucoup servi ça au premier Q1 on a eu les minis projets et j'avoue que j'avais beaucoup de mal avec la programmation, j'ai bossé à fond et j'ai réussi avec 13. Ça va mais y'avait les points des minis projets qui rentraient en compte. J'étais pas du tout à l'aise à part avec les conditionnelles mais pas plus que ça. Ici quand on a eu les consignes des projets, on a fait des groupes et on s'est dit qu'on allait jamais y arriver, c'est impossible. C'est trop compliqué je voyais pas du tout comment faire. J'étais avec des camarades qui étaient encore moins doués que moi. Du coup j'ai cherché sur youtube pour trouver quelque chose pour apprendre mais ça pourrait être intéressant d'avoir quelque chose comme ça parce que quand je dois vérifier si j'ai une erreur quelque part, je ne sais jamais dans le long code de notre jeu, qu'on a du faire fallait rechercher des variables dans le dictionnaire. Je ne savais jamais quelle variable j'allais rechercher, donc je les printais à chaque fois en dessous. Je n'ai jamais eu le réflexe d'utiliser le debugger. Donc même pour apprendre ça peut être super bien. Au début c'est compliqué on comprend pas pourquoi il va chercher la, pourquoi il assigne la etc. Je pense qu'au premier quadri ça m'aurait vraiment aidé. On a eu au Q1 un exercice comme ça mais je ne l'ai jamais réussi et je ne savais pas pourquoi.

(Programme2) : Prof : Explique que le debugger existe déjà, et les types fonctions etc ? Je trouve ça surtout intéressant dans les cas on en affiche plusieurs, comme ici avec float et int. Moi je me disais à chaque fois oui ça renvoie une chaîne de caractères mais pourquoi on le transforme la ou pourquoi est ce que ça change la ? Alors que la on le voit vraiment bien ! On a la réflexe de se dire celui la il est int mais comme ça c'est tout du float, et bien il prend du float. Les fonctions c'est top de voir qu'elles s'actualisent. Par exemple pour le count, ce qui rentre et sort en grisé et en noir. C'est vrai qu'au début on faisait une seule fonction et on l'appelait qu'une seule fois dans le fond mais en fait on peut les appeler partout et ça effectue le même morceau de code donc c'est vrai que c'est intéressant de voir pourquoi avec ce morceau de code la ça fonctionne et avec ce morceau de code la ça ne fonctionne pas.

# Annexe C : Réponses des étudiants aux parties écrites des tests utilisateur



1. Quels sont les types et les valeurs des variables ?

| Variable | Type | Contenu |
|---|---|---|
| variable1 | INT | 10 |
| variable2 | STR | 10,5 |
| variable3 | INT | 100 |
| variable4 | INT | 10 |
| variable5 | STR | 10 |
| variable6 | STR | 1 |
| variable7 | INT | 22,5 |
| variable8 | STR | BONJOUR |
| variable9 | | |
| variable10 | | |
| variable11 | | |
| variable12 | | |
| variable13 | STR | BONJOUR |
| variable14 | BOOL | TRUE |
| variable15 | STR | BONJOURDORA |

```
variable1 = 10
variable2 = input("Entrez une valeur : ") # rentre 10.5
variable3 = int(input("Entrez une valeur : ")) # rentre 100
variable4 = 100 % 90 > 50
variable5 = "100 % 90 > 50"
variable6 = input("Entrez une valeur : ") # rentre 10 % 3
variable7 = 10.5 + 12
variable8 = "Bonjour"
variable9 = variable8 + variable7
variable10 = (5 < 10) + 5
variable11 = 14 < 10 + 5
variable12 = variable11 + 50
variable13 = ("Bonjour",)
variable14 = variable13.__contains__("Bonjour")#vérifie que la collection contient le paramètre
variable15 = variable8 + (" Dora")
```

2. Quels sont les types et les valeurs des variables ?

| Variable | Type | Contenu |
|---|---|---|
| variable1 | FLOAT | 11,5 |
| variable2 | STR | 10 |
| variable3 | BOOL | FALSE |
| variable4 | BOOL | TRUE |
| variable5 | STR | 48==47+1 |
| variable6 | STR | 10%3 |
| variable7 | FLOAT | 51,5 |
| variable8 | LIST | [BONJOUR] |
| variable9 | LIST | [BONJOUR, 12] |
| variable10 | INT | |
| variable11 | BOOL | TRUE |

**Code**

```
variable1 = 11.5
variable2 = input("Entrez une valeur : ")          #entre 10
variable3 = bool(input("Entrez une valeur : "))    #entre 100 % 90 > 50
variable4 = 48 == 47 + 1
variable5 = "48 == 47 + 1"
variable6 = input("Entrez une valeur : ")          #entre 10 % 3
variable7 = variable1 + 50
variable8 = ["Bonjour"]
variable8.append(12)                               #ajoute le paramètre à la collection
variable9 = variable8 - 1
variable10 = 14 < 10 + 5
variable11 = "Je suis Dora" / variable4
```

**Légende :**

int — liste [ ]
float — tuple ( )
String — dict { : , : }
Bool — set { , , , }

FIGURE 5.1 – Réponses écrites de l'étudiant 1 (premier test)

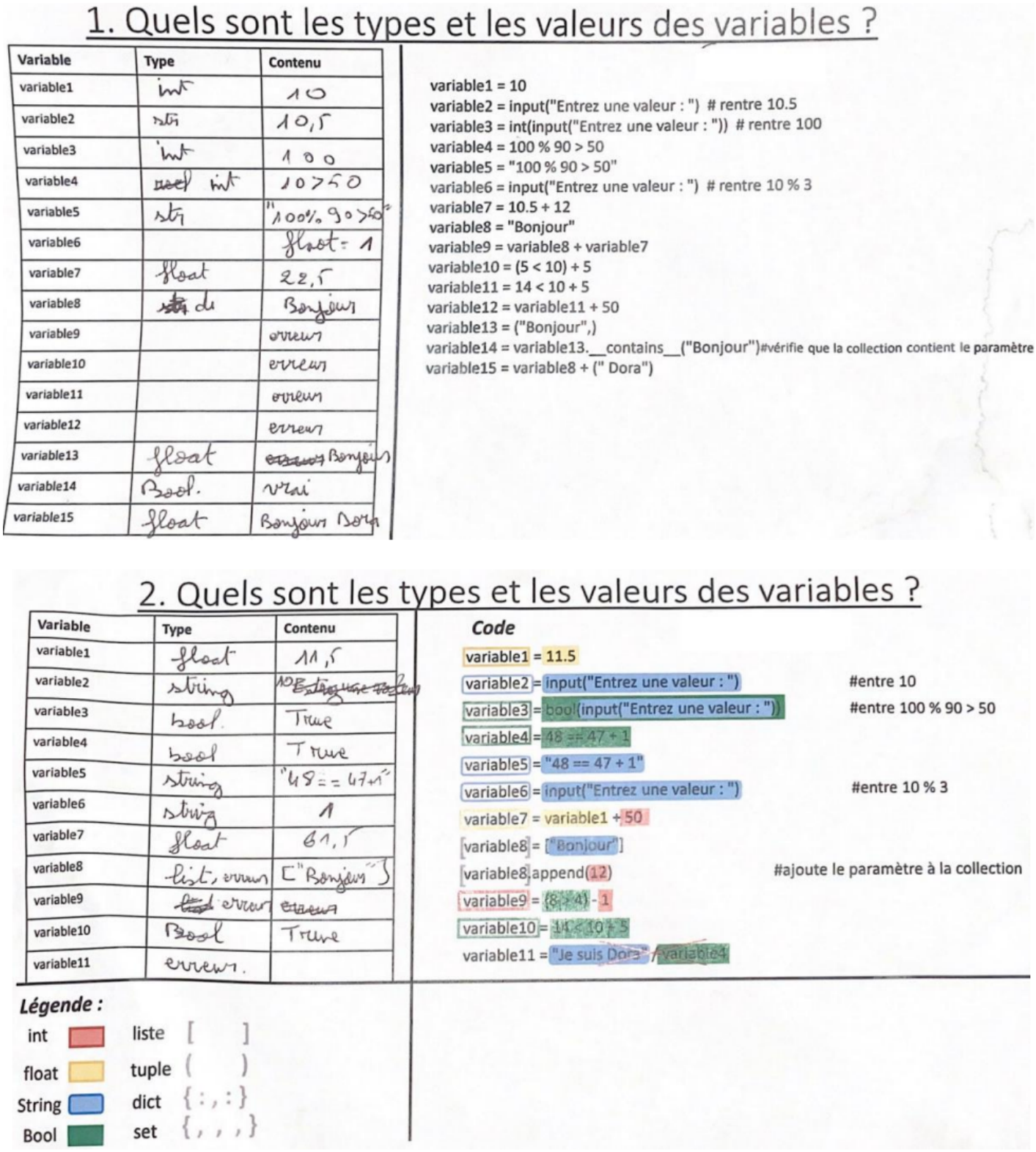



FIGURE 5.2 – Réponses écrites de l'étudiant 2 (premier test)

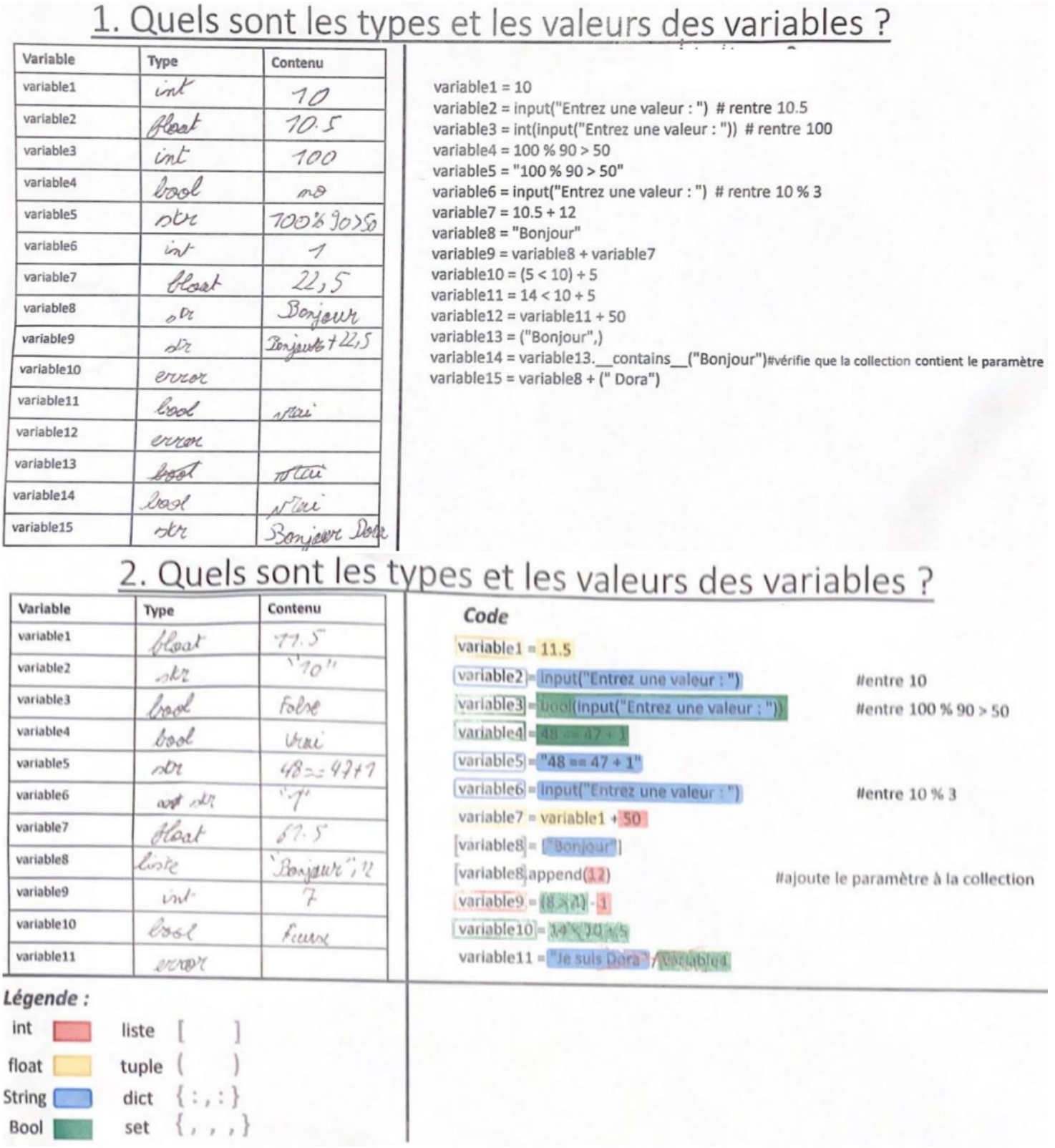
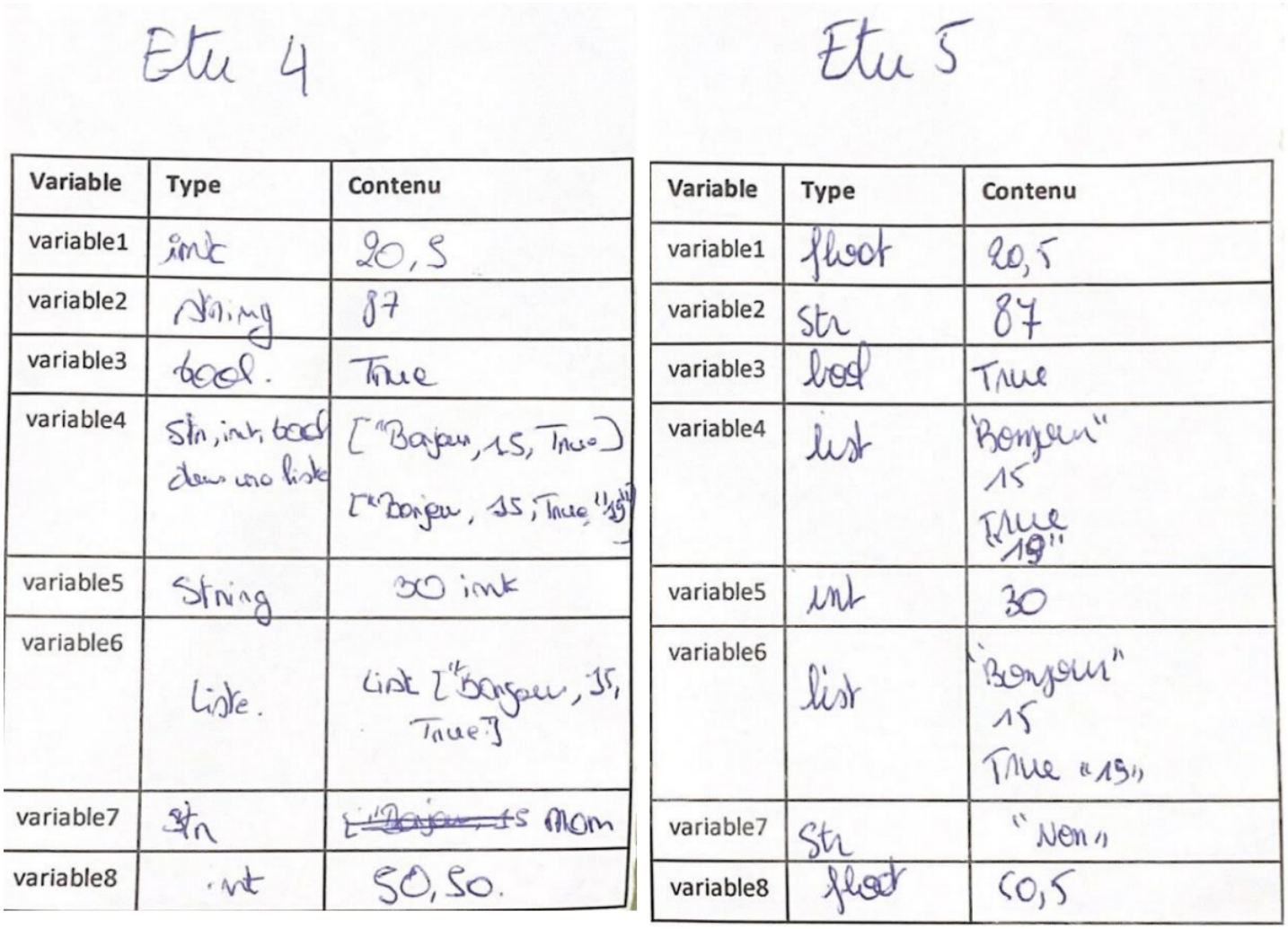



FIGURE 5.3 – Réponses écrites de l'étudiant 3 (premier test)




| Variable | Type | Contenu |
|---|---|---|
| variable1 | int | 20,5 |
| variable2 | string | 87 |
| variable3 | bool. | True |
| variable4 | Str, int, bool dans une liste | ["Bonjour, 15, True] ["Bonjour, 15, True, "19"] |
| variable5 | String | 30 int |
| variable6 | Liste. | List ["Bonjour, 15, True"] |
| variable7 | Str | ~~"Bonjour, 15~~ Non |
| variable8 | int | 50,50. |

| Variable | Type | Contenu |
|---|---|---|
| variable1 | float | 20,5 |
| variable2 | Str | 87 |
| variable3 | bool | True |
| variable4 | list | "Bonjour" 15 True "19" |
| variable5 | int | 30 |
| variable6 | list | "Bonjour" 15 True "19" |
| variable7 | Str | "Non" |
| variable8 | float | 50,5 |

FIGURE 5.4 – Réponses écrites des étudiants 4 et 5 (deuxième test)

# Bibliographie


[1] Kousuke Abe, Yuki Fukawa, and Tetsuo Tanaka. Prototype of visual programming environment for c language novice programmer. pages 140–145, 07 2019.

[2] Aivar Annamaa. Introducing thonny, a python ide for learning programming. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, Koli Calling '15, page 117–121, New York, NY, USA, 2015. Association for Computing Machinery.

[3] Evans Data Corporation. "worldwide developer population and demographic study 2021 v2". `https://evansdata.com/reports/viewRelease.php?reportID=9`.

[4] Kathryn Cunningham, Sarah Blanchard, and Mark Guzdial. Using tracing and sketching to solve programming problems : Replicating and extending an analysis of what students draw. pages 164–172, 08 2017.

[5] BENEDICT DU BOULAY, TIM O'SHEA, and JOHN MONK. The black box inside the glass box : presenting computing concepts to novices. *International Journal of Human-Computer Studies*, 51(2) :265–277, 1999.

[6] Philip J. Guo. Online python tutor : Embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, page 579–584, New York, NY, USA, 2013. Association for Computing Machinery.

[7] Jeisson Hidalgo-Céspedes, Gabriela Marín-Raventós, and Vladimir Lara-Villagrán. Learning principles in program visualizations : A systematic literature review. In *2016 IEEE Frontiers in Education Conference (FIE)*, pages 1–9, 2016.

[8] Cindy E. Hmelo and Mark Guzdial. Of black and glass boxes : Scaffolding for doing and learning. In *Proceedings of the 1996 International Conference on Learning Sciences*, ICLS '96, page 128–134. International Society of the Learning Sciences, 1996.

[9] Fionnuala Johnson, Stephen McQuistin, and John O'Donnell. Analysis of student misconceptions using python as an introductory programming language. pages 1–4, 01 2020.

[10] Linxiao Ma, John Ferguson, Marc Roper, Isla Ross, and Murray Wood. Using cognitive conflict and visualisation to improve mental models held by novice programmers. volume 40, pages 342–346, 02 2008.

[11] Richard Mayer. *The Cambridge Handbook Of Multimedia Learning*. 08 2005.

[12] Alexander Miller, Stuart Reges, and Allison Obourn. Jgrasp : A simple, visual, intuitive programming environment for cs1 and cs2. *ACM Inroads*, 8 :53–58, 10 2017.

[13] Monika Mladenović, Žana Žanko, and Marin Aglić. The impact of using program visualization techniques on learning basic programming concepts at the k–12 level. *Computer Applications in Engineering Education*, 29 :145–159, 01 2021.

[14] Thomas Naps, Guido Rößling, Vicki Almstrum, Wanda Dann, Rudolf Fleischer, Christopher Hundhausen, Ari Korhonen, Lauri Malmi, Myles Mcnally, Susan Rodger, and J. Ángel Velázquez-Iturbide. Exploring the role of visualization and engagement in computer science education. *SIGCSE Bulletin*, 35 :131–152, 05 2003.

[15] Stack Overflow. "developer survey results 2016". https://insights.stackoverflow.com/survey/2016.

[16] Teemu Rajala, Mikko-Jussi Laakso, Erkki Kaila, and Tapio Salakoski. Effectiveness of program visualization : A case study with the ville tool. 01 2008.

[17] Teemu Rajala, Mikko-Jussi Laakso, Erkki Kaila, and Tapio Salakoski. Ville – a language-independent program visualization tool. 02 2022.

[18] Jorma Sajaniemi and Marja Kuittinen. Program animation based on the roles of variables. In *Proceedings of the 2003 ACM Symposium on Software Visualization*, SoftVis '03, page 7–ff, New York, NY, USA, 2003. Association for Computing Machinery.

[19] Jorma Sajaniemi and Marja Kuittinen. Visualizing roles of variables in program animation. *Information Visualization*, 3 :137–153, 06 2004.

[20] Santiago Schez-Sobrino, María García, Carmen Lacave, Ana Molina Díaz, Carlos Glez-Morcillo, David Vallejo, and Miguel Redondo. A modern approach to supporting program visualization : from a 2d notation to 3d representations using augmented reality. *Multimedia Tools and Applications*, 80 :1–32, 01 2021.

[21] Cheryl Seals. Visual programming for novice programmer teachers. pages 26–27, 01 2005.

[22] Slavomír Simoňák. Algorithm visualizations as a way of increasing the quality in computer science education. In *2016 IEEE 14th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pages 153–157, 2016.

[23] Juha Sorva. *Visual Program Simulation in Introductory Programming Education*. PhD thesis, 05 2012.

[24] Juha Sorva. Notional machines and introductory programming education. *ACM Transactions on Computing Education*, 13 :8 :1–8 :31, 06 2013.

[25] Juha Sorva, Ville Karavirta, and Lauri Malmi. A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education*, 13 :1–64, 10 2013.

[26] Juha Sorva and Teemu Sirkiä. Uuhistle - a software tool for visual program simulation. pages 49–54, 01 2010.

[27] Tuija Stützle and Jorma Sajaniemi. An empirical evaluation of visual metaphors in the animation of roles of variables. *Informing Science : The International Journal of an Emerging Transdiscipline*, 8 :087–100, 01 2005.

[28] Bing Sun. Java teaching based on bluej platform. In *2010 2nd International Conference on Information Engineering and Computer Science*, pages 1–4, 2010.

[29] Alaaeddin Swidan, Felienne Hermans, and Marileen Smit. Programming misconceptions for school students. pages 151–159, 08 2018.

[30] Žana Žanko, Monika Mladenović, and Ivica Boljat. Misconceptions about variables at the k-12 level. *Education and Information Technologies*, 24 :1251–1268, 03 2019.