Chapitre 1 : Affectation Boucle et Si

I) Introduction

I.1) Définition

> ***Définition*:**
>
> Un **algorithme** est une séquence **finie** et **univoque** d'instructions permettant de résoudre une classe de problèmes

---

> **Exemple 1:**
>
> L'addition avec retenue est un algorithme que vous connaissez et utilisez depuis le primaire ! Il permet d'additionner deux entiers positifs quels qu'ils soient.

II) Variables et instructions élémentaires

II.1) Types de variables

En informatique, pour stocker les données, on utilise des **variables** . Une variable est définie par un nom qui permet de l'identifier de manière unique. Ce nom permet à l'ordinateur de retrouver la donnée contenue dans sa mémoire qui correspond à la valeur de la variable. Chaque variable est caractérisée par un **type** qui correspond au type de donnée qu'elle contient. Il existe de nombreux types de variables différents en informatique mais nous n'en étudierons que quatre à ce stade

> **Idée à retenir :**
>
> Une variable est la donnée de : **1 NOM et 1 type et 1 donnée**

Type booléen

Une variable de **type booléen** (ou variable booléenne) est une variable qui prend l'une des deux valeurs : **vrai** ou **faux**. En Python, le type booléen se note `bool` et les valeurs associées `True` et `False` respectivement.

Type entier

Le **type entier** désigne, comme son nom l'indique, des variables entières (qui appartiennent donc à l'ensemble $\mathbb{N}$). En Python, le type entier se note `int` (contraction

de «integer» qui signifie entier en anglais).

Type flottant

Une variable de **type flottant** (ou variable flottante) contient un nombre qui s'écrit avec un nombre fini de chiffres après la virgule. Le type flottant se note `float` en Python (qui signifie flotter en anglais). Attention, la virgule est remplacée par un point dans l'écriture anglo-saxonne des nombres et donc, en particulier, dans le langage Python

Type chaîne de caractères

Une variable de **type chaîne de caractères** contient du texte. Le terme caractère désigne les caractères typographiques que sont les lettres (minuscules ou majuscules, avec ou sans accents), les chiffres, les signes de ponctuation, etc. Dans le langage Python, ce type est noté `str` (contraction de «string» qui signifie fil ou enchaînement en anglais). Les chaînes de caractères s'écrivent en Python par du texte entre guillemets simples (par exemple : 'Ceci est un string en Python.') ou guillemets doubles (par exemple : "Ceci est également un string en Python!"

II.2) Affectation

> ***Définition*:**
>
> L'**affectation** d'une valeur à une variable est l'action de donner une valeur à cette variable.

On l'écrit en python avec le signe "=". Le code suivant affecte à a la valeur 3.

> ***Attention Important*:**
>
> La cellule suivante est une **cellule de code**. Il faut la lancer !
>
> Vous avez une bouton **play** pour la lancer. Vous pouvez aussi faire Maj + Entrée pour la lancer

---

In [ ]:
```
a = 3
```

Maintenant quand l'on regarde ce qu'il y a dans a on aura 3. Lancer (avec le bouton Play ou Maj+Entrée) la cellule suivante.

In [ ]:
```
a
```

Attention, le signe « = » n'a donc pas le même sens en Python qu'en mathématiques ! En particulier, en Python, « a = x » n'a pas le même sens que « x = a ».

En programmation, il est possible d'affecter et de **réaffecter** des variables. Dans ce cas, c'est la dernière affectation qui prévaut sur les autres. On peut également mettre à jour une variable en lui réaffectant une valeur qui dépend d'elle-même

---

> ***Exercice:***
>
> Le code python affecte plusieurs valeurs successives à `a` en réalisant plusieurs affichages (la fonction `print` permet de réaliser un affichage en Python). Sans l'exécuter que va renvoyer la console python lorsque l'on exécute ce code ?

In [ ]:
```
#Pour vous empecher d'exécuter le code.
assert 1 == 2, "Il ne faut pas exécuter le code bande de  Coquins"

a = -3
a = 1
print(a)
a = a + 1
print(a)
```

> Réponse : .....

Vérification :

In [ ]:
```
a = -3
a = 1
print(a)
a = a + 1
print(a)
```

II.3 Opérations élémentaires

À chaque type est associé un ensemble d'opérations. Vous trouverez ici la liste des opérations les plus élémentaires associées aux quatre types décrits précédemment. Tous les types admettent également deux opérations universelles qui sont l'opération de test d'égalité et l'opération de test de différence. L'opération de test d'égalité, no- tée « = » en pseudo-code et « == » en Python, permet de vérifier si deux variables sont égales. L'opération de test de différence, notée « 6 = » en pseudo-code et « != » en Python, permet de vérifier si deux variables sont différentes.

> ***Exercice:***

> Le code python suivant affecte la valeur 2 à la variable `a` et la valeur `-3` à la variable `b`.

In [ ]:
```
a= 2
b = -3
print(a ==b)
print(a != b)
```

Opérations sur les flottants

Pour les flottants, on peut utiliser les opérations classiques entre deux nombres : l'addition, la soustraction, la multiplication, la division, l'élévation à la puissance ; ainsi que les comparateurs d'ordre : inférieur ou égal, inférieur strict, supérieur ou égal, supérieur strict. La liste des symboles utilisés en Python figure dans le tableau ci-dessous :

Opération mathématique	Expression en python
$a+b$	`a+b`
$a-b$	`a-b`
$a\times b$	`a*b`
a/b	`a/b`
a^b	`a**b`
$a\leq b$	`a <= b`
$a<b$	`a<b`
$a\geq b$	`a >= b`
$a>b$	`a > b`

Opération sur les entiers

Parmi les opérations standards entre deux entiers, on retrouve toutes les opéra- tions décrites pour les flottants auxquelles on peut ajouter le quotient et le reste par la division euclidienne (c'est-à-dire la division entière).

Opération mathématiques	Expression Python
Quotient de la division entière de a par b	`a//b`
Reste de la division entière de a par b	`a%b`

> ***Exercice:***
>
> Dans le code suivant que valent les variables `a,b,c,d,e,f` ?

In [ ]:
```
a = 3 - 1
b = 1 + 2
c=  a**3
d = c / 5
e = c //5
f = (a <= b)
```

Réponse : .....

Opérations sur les booléens

Les opérations standards sur les booléens sont :

- la **négation** au mot clé **non**
- la **conjonction** associée au mot-clé **et**
- la **disjonction** associée au mot-clé **ou**

Elles sont implémentées en Python à l'aide des opérateurs `not` , `and` et `or` respectivement. Si a et b sont deux variables booléennes, on a :

- $\textbf{non}(a)$ vaut **vrai** si et seulement si a vaut **faux**
- a **et** b vaut **vrai** si et seulement si les deux variables a et b sont égales à **vrai**
- a $\textbf{ou}$ b vaut **vrai** si et seulement si l'une au moins des deux variables a $\textbf{ou}$ b vaut **vrai**

> ***Exercice:***
>
> Analysez le code Python suivant. Quelles valeurs prennent les variables booléennes `a,b,c` ?

In [ ]:
```
a = (0 <1) or (0>2)
b = not (1 <2)
c = a and b
```

Opérations sur les chaînes de caractère

Il existe un certain nombre d'opérations sur les chaînes de caractère en Python mais nous n'en considérons qu'une ici : la concaténation. La concaténation permet de créer une

nouvelle chaîne de caractère à partir de deux chaînes de caractère a et b, en mettant les caractères de b à la suite des caractères de a. La concaténation de a et b en Python se note `a + b`.

> ***Exercice:*** On considère le code Python suivant. Que valent les variables `a,b,c` ?

In [ ]:
```
a = "math"
b = a + "ematiques"
c = "ha"
c = c + c
d = c*4
```

II.4) Instructions conditionnelles

*Un père dit à son enfant : "**Si** tu finis tes légumes, **Alors** tu auras un dessert". L'enfant ne finit pas ses légumes mais le père donne quand même à son enfant un dessert. Le père a-t-il menti ?*

La réponse est non. Et ce n'est pas pour des raisons d'éducation ou de morale. Le père a dit ce qu'il ferait si son enfant finit son dessert mais il n'a rien dit si il ne finissait pas son dessert.

Dans un algorithme, on peut choisir qu'une instruction ne s'exécute que si certaines conditions sont remplies. En pseudo-code, on peut utiliser les mots-clés si et alors pour indiquer une condition et la séquence d'instructions à exécuter si la condition est remplie. On peut également utiliser le mot-clé Fin si pour indiquer la fin de la séquence d'instructions conditionnelles. Une instruction conditionnelle peut donc s'écrire en pseudo-code sous la forme suivante :

```
Si la condition est vraie
    Alors faire
        instructions
Fin Si
```

En Python, on utilise la commande `if` associé à l'utilisation des deux points `:` et d'une indentation (décalage vers la droite des lignes d'instructions 4), selon la forme suivante :

In [ ]:
```
if condition:
    instruction
```

Attention, les règles pour les deux points et l'indentation ne sont pas facultatives et le code ne fonctionnera pas correctement si elles ne sont pas respectées. Par ailleurs,

l'indentation doit courir sur l'ensemble des instructions couvertes par la condition (donc éventuellement sur plusieurs lignes).

On peut également différencier entre un certain nombre de cas en utilisant les mots-clés **« sinon si »** et **« sinon »** en pseudo-code ou les commandes `elif` et `else` en Python, selon le schéma qui suit :

```
Si la condition1 est vraie:
    Alors faire
        instructions1
Sinon si la condtition2 est vraie:
    Alors faire
        instruction2
Sinon si la condition3 est vraie:
    Alors faire
        instruction3
Sinon:
    Faire
        instruction4
Fin si
```

En python, cela donne :

In [ ]:
```
if condition1:
    instruction1
elif condition2:
    instruction2
elif condition3:
    instruction3
else:
    instruction4
```

> ***Exemple*:**
>
> Deux joueurs s'affrontent aux dés. Le joueur qui obtient la plus grande valeur a gagné. Le programme suivant récupère la valeur obtenue par chacun des joueurs puis annonce qui a gagné la partie (on utilise pour cela les fonctions « `int` » et « `input` » de Python qui seront présentées en détails dans la suite de l'année).

In [ ]:
```
d1 = int(input("Entrez la valeur obtenue par le joueur 1"))
d2 = int(input("Entree la valeur obtenue par le joueur 2"))
if d1 > d2:
    print("Le joueur 1 a gagné")
elif d2 > d1:
    print("Le joueur 2 a gagné")
else:
    print("Match nul ! ")
```

II.5) Boucles

En algorithmique, les boucles permettent de répéter une séquence d'instructions sans avoir à réécrire la séquence. On distingue deux formes de boucles :

- les boucles bornées «pour», pour lesquelles la répétition de la séquence d'instructions correspond au parcours de tous les éléments d'un ensemble fini par une variable ;
- les boucles non bornées «tant que», pour lesquelles la répétition de la séquence d'instruction est soumise à condition de répétition à chaque tour de la boucle.

Les boucles bornées sont appelées ainsi parce qu'elles s'arrêtent quand on arrive au bout de l'ensemble fini correspondant. Les boucles non bornées sont appelées ainsi parce qu'elles ne s'arrêtent que si la condition de répétition est fausse.

La boucle bornée «pour»

Dans ce cours, on ne considère que les boucles **«pour»** où la répétition de la séquence correspond au parcours par une variable d'un intervalle d'entiers (par exemple, {4, 5, 6, 7, 8, 9}) et, le plus souvent, un intervalle d'entiers débutant en 0 (par exemple, {0, 1, 2, 3, 4, 5, }). En pseudo-code, on pourra utiliser le mot-clé **«pour»** en début de boucle et indiquer la fin de cette boucle par un **«fin pour»**

```
Pour i allant de 1 à n
    Faire
        instructions
Fin Pour
```

En Python, on utilise la commande `for` associée à une instruction de la forme `i in range(n)` qui signifi que `i` parcourt l'intervalle $[0, 1, 2, \ldots, -1]$, et suivie d'un ":". Une indentation permet ensuite d'indiquer les instructions correspondant à la séquence d'instructions à répéter.

In [ ]:
```
for i in range(n):
    instructions #Il va y avoir une erreur
```

La boucle non bornée «tant que»

Dans la boucle «tant que», la boucle se répète tant qu'une condition de répétition est vraie. En pseudo-code, on peut indiquer le commencement d'une telle boucle par «tant que» et la fin des instructions de la boucle par «fin tant que».

```
Tant que la condition est vraie
    Faire
     instructions
Fin tant que
```

En Python, la boucle **«tant que»** passe par la commande **«while»**, associée à une condition, et suivie d'un deux points. Comme usuellement en Python, une indentation permet d'indiquer les instructions correspondant à la séquence d'instructions à répéter.

In [ ]:
```
while condition:
    instruction
```

Attention, si la condition est toujours vérifiée, la boucle ne s'arrêtera jamais. On dit que le programme tourne en **boucle infinie**. Généralement, ce n'est pas souhaitable. Il faut donc s'assurer que la condition de répétition puisse passer de la valeur vrai à la valeur faux lors de la séquence d'instructions de la boucle.

> ***Exercice*:**
>
> Dans le programme Python qui suit, l'une des deux boucles tourne en boucle infinie. Laquelle ?
>
> Boucle 1 :
>
> ```
> i = 0
> while i >= 0:
>   i = i+1
>   print(i)
> ```
> Boucle 2:
>
> ```
> i = 10
> while i>=0:
>   i = i-1
>   print(i)
> ```

On remarquera que dans l'exercice précédent, la variable i est affectée à une va- leur avant le début de la boucle. Ceci est un schéma classique en programmation qui s'appelle l'initialisation. On dit, par exemple, que la variable i est initialisée à 0 avant la première boucle et initialisée à 10 avant la deuxième boucle

Erreurs et bugs

- `SyntaxError` : Les erreurs de Syntaxe sont dues au non-respect des règles d'écri- ture de Python, comme un oubli de parenthèse ou un manque de " :"
- `NameError` : Les erreurs de définition sont dues à l'usage d'un nom qui n'a pas été défini, par exemple une variable, fonction qui n'a pas été définie.
- `TypeError` : Les erreurs de type sont dues à des valeurs dont le type n'est pas com- patible avec l'expression dans laquelle elles apparaissent.
- ***Erreur d'exécution*** : les erreurs sont dues à des instructions que l'ordinateur ne sait pas exécuter comme une division par zéro.

- ***Erreur de logique*** : Ces dernières n'occasionnent pas en général pas de message d'erreur car il s'agit d'erreurs dans logique du programme, qui n'enfreignent pas les règles de Python. Ce sont les erreurs les plus difficiles à résoudre. Une erreur de logique courante est la boucle infinie : le programme ne s'arrête jamais car la condition d'une boucle non bornée ne devient jamais fausse. Dans ce cas, il faut en général interrompre le programme "de force" en tapant ***Control-C***.

III) Exercices

Exercice 1

On considère les trois séquences d'instructions suivantes en pseudo-code.

1. On suppose que la variable x contient la valeur 2 avant l'exécution de la séquence. Dans chacun des cas, déterminez la valeur dans x après exécution de la séquence.
2. Identifiez la ou les séquences pour lesquelles, si x est initialisée à a avant l'exécution de la séquence, alors x contient $(a-1)^2+(a+1)^2$ après exécution de la séquence

In [ ]:
```
# Séquence 1
x = 3 # On donne une valeur à x

x = x + 1
b = x**2
a = x - 1
c = a**2
x = b - c
```

In [ ]:
```
# Séquence 2
x = 2 # On donne une valeur à x

x = x-1
a = x**2
x = x+2
b = x**2
x = a + b
```

In [ ]:
```
# Séquence 3
x = 4 # On donne une valeur à x

a = x - 1
b = a**2
c = x+1
d = c**2
x = b + d
```

Exercice 2

Aladin propose le code Python suivant pour échanger la valeur de deux variables a et b

1. Expliquer pourquoi le code fourni par Aladin ne remplit pas ses objectifs.
2. Proposer une autre solution qui échange effectivement les deux variables.

In [ ]:
```
a = 42
b = 23
a = b
b = a
```

In [ ]:
```

```

Exercice 3

On considère le code Python suivant.

1. Que fait le programme si l'utilisateur rentre la valeur 15 ?
2. Et la valeur 1515 ?

In [ ]:
```
x = int(input("Veuillez saisir un nombre entier."))
if x < 100:
    print("Votre nombre est bien faible !")
else:
    print("Quel beau nombre ! ")
print("Au revoir !")
```

In [ ]:
```

```

Exercice 4

À la fête foraine, l'accès aux montagnes russes est réservé aux personnes mesurant au moins 1m20 (compris) et au plus 2m10 (compris). Faites un programme Python qui demande à un utilisateur de rentrer sa taille en mètres et l'informe, en fonction de sa réponse, s'il :

- peut monter dans l'attraction ;
- est trop petit ;
- est trop grand

In [ ]:
```

```

Exercice 5

On considère le code Python suivant.

1. Quelle valeur est contenu dans la variable a après exécution du programme.
2. Comment modifier le code pour produire un fou rire.

In [ ]:
```
a = "A"
for k in range(2):
    a = a + "ha"
a = a + "!"
print(a)
```

Exercice 6

Pour chacun des programmes suivants, dire quelle valeur prend la variable d après exécution du programme.

In [ ]:
```
a = 1
b = 1
c = 1
d = 0
for k in range(2):
    a = a + b
    b = b + a
    c = c + b
    d = d + c
```

In [ ]:

In [ ]:
```
a = 1
b = 1
c = 1
d = 0
for k in range(2):
    a = a + b
    b = b + a
    c = c + b
d = d + c
```

In [ ]:

In [ ]:
```
a = 1
b = 1
c = 1
d = 0
for k in range(2):
    a = a + b
    b = b + a
c = c + b
d = d + c
```

In [ ]:

In [ ]:
```
a = 1
b = 1
c = 1
d = 0
for k in range(2):
    a = a + b
b = b + a
c = c + b
d = d + c
```

In [ ]:

Exercice 7

On considère la séquence d'instructions en pseudo-code qui suit.

```
x = 3
y = 11
k = 1
Tant que $x<y$
    Faire
     x = 3x + 2
     y = 2y + 1
     k = k+1
Fin Tant que
Afficher k
```

1. Quelle valeur est affichée lorsque l'on exécute cette séquence ?
2. Implémentez cette séquence en Python et exécutez votre code afin de vérifier votre réponse.

In [ ]:

Exercice 8

Un étudiant fauché place 10€ sur son livret A en 2020, rémunéré 0.5% d'intérêt par an. À partir de quelle année cet étudiant aura-t-il au moins 20€ sur son livret ? Réali- sez un code Python qui permette de répondre à la question.

In [ ]:

Exercice 9

En utilisant le package random (taper import random) et la commande random.randint(a,b) qui tire un nombre au hasard en a et b.

1. Faites une boucle conditionnelle tirant au hasard un nombre entre 0 et 50 jusqu'à être le numéro 1. On affiche le numéro dans la boucle.
2. Cette boucle est elle infini et pourquoi ?
3. Faire une boucle qui tire des nombres au hasard jusqu'à avoir un nombre pair.
4. Faire une boucle où le nombre augmente de 3 en 3 jusqu'à dépasser le numéro 1802.

In [ ]:

Exercice 10

1. Faire une boucle qui affiche les noms de toute votre famille
2. Faire une boucle qui compte tous les nombre de 1 à 1000
3. Faire une boucle qui dit le nombre de lettres pour le nom de chaque personne de votre famille (utiliser la fonction len())

In [ ]: