Thème 6 - Langages et programmation

01 Les Variables en Python

1. Variables et Affectation

A retenir

Dans le langage Python, le symbole `=` correspondant à l'affectation (écrite ⟵ en pseudo-code). Pour affecter la valeur 2 à une variable `a` on écrit simplement `a = 2`.

Ainsi, l'algorithme

Algo

```
a ← 3
b ← 2 * a
```

s'écrit en Python de la façon suivante :

Script Python

```
a = 3
b = 2 * a
```

Algo

```
a = 3
b = 2 * a
```

```
print(a)
print(b)
```

Algo

```
3
6
```

Exercice 1

Écrivez le programme Python correspondant à l'algorithme ci-dessous. Que valent `N` et `P` après l'exécution de cet algorithme ?

Algo

```
N ← 2
P ← 3
N ← P + 1
P ← N
```

Les noms de variables sont des noms que vous choisissez vous-même assez librement. Efforcez-vous cependant de bien les choisir :
- aussi **explicites que possible**, de manière à exprimer clairement ce que la variable est censée contenir.

Par exemple, des noms de variables tels que altitude, altit ou alt conviennent mieux que x pour exprimer une altitude.

ATTENTION
Il existe des noms réservés :

and	del	from	None	True	as
elif	global	nonlocal	try	assert	else
if	not	while	break	except	import
or	with	class	False	in	pass
yield	continue	finally	is	raise	def
for	lambda	return			

2. L'incrémentation d'une variable

Vocabulaire

«Incrémenter» une variable signifie l'*augmenter*.

Imaginons une variable appelée `compteur`. Au démarrage de notre programme, elle est initialisée à la valeur 0.

Script Python

```
compteur = 0
```

Considérons qu'à un moment du programme, cette variable doit être modifiée, par exemple en lui ajoutant 1.

Ne perdez pas de vue que le signe = est une affectation, et non une égalité.

En Python, cela s'écrira :

- Syntaxe classique

  Script Python

  ```
  compteur = compteur + 1
  ```

- Syntaxe Pythonesque

  Script Python

  ```
  compteur += 1
  ```

Algo

```
compteur=0
compteur+=1
print(compteur)
```

Algo

```
1
```

Exercice 2

Ecrire le code «classique» et le code «Pythonesque» pour l'instruction suivante :
On initialise une variable `capital` à 1000 et on lui enlève 5%.

3. Derouler un code : Premier algorithme

Considérons ce code :

Script Python

```
a = 1
b = 3
```

```
c = a + b
```

On peut décrire le déroulement du code en faisant un tableau. + A gauche on indique le numéro de la ligne. + A droite, tout les changements d'état de la mémoire (c'est à dire la ou les variables modifiée(s) à cette ligne, avec leur nouvelle valeur).

n° ligne	état
1	a = 1
2	b = 3
3	c = 4

Exercice 3

Ecrire le déroulé du code, puis donnez la valeur de x à la fin du code.

Script Python

```
x = 2
x = 3 - x
y = 5 * x
x = y / 2
```

Correction

n° ligne	état
1	x = 2
2	x = 1
3	y = 5
4	x = 2.5

A la fin x vaut 2.5

Exercice 4

Ecrire le déroulé du code, puis donnez la valeur de nom à la fin du code.

Script Python

```
nom = "Cépadur"
prenom = "Alban"
nom = prenom + " " + nom
nom = nom +"."
```

Correction

n° ligne	état
1	nom = "Cépadur"
2	prenom = "Jean"
3	nom = "Alban Cépadur"
4	nom = "Alban Cépadur,"

A la fin nom vaut **"Alban Cépadur,"**

Exercice 5

Ecrire le déroulé du code et donnez les valeurs de x et de y à la fin du code.

Script Python

```
nb_de_tour = 0
nb_de_tour = nb_de_tour + 1
nb_de_tour = nb_de_tour + 2
nb_de_tour = nb_de_tour + 3
```

Correction

n° ligne	état
1	nb_de_tour = 0
2	nb_de_tour = 1
3	nb_de_tour = 3
4	nb_de_tour = 6

A la fin **nb_de_tour** vaut 6

4. L'échange de variables

Après l'incrémentation, une autre technique de base reviendra fréquemment dans nos codes : **l'échange de variables**.

Exercice 6

Ecrire le déroulé du code et donnez les valeurs de x et de y à la fin du code.

Script Python

```
x = 2
y = 3
x = y
y = x
```

Correction

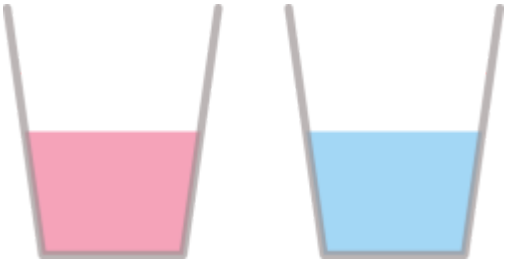
n° ligne	état
1	x = 2
2	y = 3
3	x = 3
4	y = 3

A la fin x vaut 3 et y vaut 3

Dans l'exemple précédent, la variable `x` a été écrasée dès qu'on lui a donné la valeur de la variable `y`, on n'a donc pas procédé à une échange de valeur.

Pour procéder à un échange de valeur comment faire ?

La situation est similaire au problème suivant : comment échanger le contenu de ces deux verres ?



La méthode est évidente : il nous faut un troisième verre.

Nous allons faire de même pour nos variables. Nous allons utiliser une variable **temporaire** (on parle aussi de variable **tampon**) pour conserver la mémoire de la valeur de `x` (par exemple) avant que celle-ci ne se fasse écraser :

Script Python

```
>>> x = 2
>>> y = 3
>>> temp = x
>>> x = y
>>> y = temp
```

Vous pouvez vérifier maintenant que les valeurs de `x` et de `y` ont bien été échangées.

Syntaxe classique et syntaxe Pythonesque :

L'échange de deux variables `x` et de `y` s'écrit donc :

Script Python

```
>>> temp = x
>>> a = y
>>> y = temp
```

Mais il existe aussi une syntaxe particulière à Python, bien plus courte :

Script Python

```
>>> x, y = y, x
```

5. Types de variables

Pour ce début d'année, il faut avoir à l'esprit qu'il existe 4 types de base :

Types de base :

Voici les types Python les plus fréquemment utilisés cette année:

Type Python	Traduction	Exemple
`int`	entier	`42`
`float`	flottant (décimal)	`3.1416`
`str`	chaîne de caractères (string)	`"NSI"`
`bool`	booléen (True ou False)	True
`tuple`	p-uplet	`(255, 127, 0)`
`list`	liste	`[0, 1, 2, 3, 4, 5]`
`dict`	dictionnaire	`{'Homer':43, 'Marge':41, 'Bart':12, 'Lisa':10, 'Maggie':4}`
`function`	fonction	`print`

int est l'abréviation de **interger** qui ignifie entier en anglais

float est l'abbréviation de **floating point** qui se traduit en français par **nombre à virgule flottante**. Nous aurons l'occasion plus tard de reparler de ce nom étrange, pour le moment considérez qu'il s'agit de nombres décimaux.

str est l'abréviation de **string**, et désigne les chaine de caractères (character strings).

bool est l'abréviation de **boolean**, soit booléen en français. C'est un type moins intuitif mais très important en informatique. Une variable booléenne ne peut avoir que 2 valeurs : **True** ou **False** (Vrai ou Faux)

Comment connaître le type d'une variable ? Il suffit dans la console d'utiliser la fonction `type`.

Algo

```
a = 1
type(a)
```

Algo

```
<class 'int'>
```

Script Python

```
un_entier = 1
print("le type de la variable un_entier est :", type(un_entier))

mot = "Albert"
print("le type de la variable mot est :", type(mot))

un_float = 1.0
print("le type de la variable un_float est :", type(un_float))
```

```
bool = True
print("le type de la variable bool est :", type(bool))
```

Algo

```
le type de la variable un_entier est : <class 'int'>
le type de la variable mot est : <class 'str'>
le type de la variable un_float est : <class 'float'>
le type de la variable bool est : <class 'bool'>
```

Vous vous posez peut-être la question de savoir si 1 et 1.0 sont identiques ?

En vérité ils ne le sont pas, puisqu'ils n'ont pas le même type. Cela veux dire qu'il ne sont pas représentés de la même façon dans votre ordinateur. Néanmoins, ils ont bien la même valeur.

Illustrons cela en utilisant le **test d'égalité : ==** et le **test d'identité : is**.

- Le premier, **a == b** est **True** (vrai) si a et b on la même valeur.
- Le second, **a is b** est **True** si a et b sont identiques.

Script Python

```
a = 1
b = 1.0
print( "a est égal à b ? -> ", a == b)

print( "a est identique à b ? -> ", a is b)
```

a est égal à b ? -> True a est identique à b ? -> False

6. Python et le typage dynamique

Jusqu'à présent, nous ne nous sommes pas occupés de préciser à Python le type de notre variable.

Script Python

```
a = 3
```

Mais dans certains langages, c'est obligatoire. En C par exemple, il faut écrire :

C

```
int a = 3;
```

Cela signifie (pour le langage C) que notre variable `a` n'aura pas le droit de contenir autre chose qu'un nombre entier.

Si on écrit ensuite

C

```
a = "test";
```

Le compilateur C renverra une erreur : on ne peut pas stocker une chaîne de caractères dans une variable qu'on a créée comme étant de type entier.

Et en Python ?

Script Python

```
>>> a = 3
>>> type(a)
<class 'int'>
>>> a = "test"
>>> type(a)
<class 'str'>
```

Python a changé tout seul le type de notre variable, sans intervention. On parle de **typage dynamique**.

7. Bonnes pratiques de nommage

7.1. Ce qui est autorisé et ce qui ne l'est pas

Pour nommer correctement une variable, il existe des règles à respecter.

> "Les règles" - le nom de la variable peut contenir les caractères suivants : - des lettres **non accentuées** (attention, minuscule et majuscule sont des caractères différents) - des chiffres (mais pas comme premier caractère) - le tiret du bas _ (underscore, tiret du 8)

- le nom de la variable **ne doit pas** commencer par un chiffre
- le nom de la variable **ne doit pas** contenir d'espace
- le nom de la variable **ne doit pas** être un mot-clé du langage.

Liste des mots-clés réservés par Python

and	as	assert	break	class	continue	def	del
elif	else	except	False	finally	for	from	global
if	import	in	is	lambda	None	not	or
pass	raise	return	True	try	while	with	yield

7.2. Du sens, du sens, du sens

Hormis pour les indices (de boucles, de tableaux...) un nom de variable (dans un programme destiné à être lu, par vous ou quelqu'un d'autre) doit **impérativement avoir du sens** :

Script Python

```
# PAS BIEN
if d == 1:
    cep += vm

# BIEN
```

```
if date == 1:
    compte_epargne += versement_mensuel
```

Règle d'or :
On ne donne jamais un nom de variable au hasard, on le choisit pour qu'il soit **explicite**.

Oui mais pour donner du sens, il faut souvent plusieurs mots... La longueur du nom de la variable n'est plus un problème depuis que la grande majorité des IDE proposent la complétion automatique.
Mais comment former ces longs mots ?

7.3. Syntaxe des noms à rallonge

Comment accoler des mots
- S'il est composé, le nom peut être de la forme: - `snake_case` : les mots sont séparés par des underscores. Conseillé en Python.
- `camelCase` : les mots sont séparés par des majuscules mais la 1ère lettre est minuscule. Conseillé en Javascript.
- `PascalCase` : les mots sont séparés par des majuscules et la 1ère lettre est majuscule. Conseillé en C.
- `kebab-case` : les mots sont séparés par des tirets courts. Conseillé en HTML - CSS.

Sans surprise, en Python, nous utiliserons donc le `snake_case`.

8. Exercices

Exercice 7

Prédire la valeur affichée après les séquences d'instructions suivantes.

1. Séquence 1 :

Script Python

```
>>> a = 5
>>> a = a + 1
>>> b = a
>>> b = b ** 2 - a
>>> print(b)
```

2. Séquence 2 :

Script Python

```
>>> a = 5
>>> b = 6
>>> a = a - b
>>> b = b + a
>>> a = b - a
>>> print(a, b)
```

3. Séquence 3 :

Script Python

```
>>> from random import randint
>>> a = randint(1, 100)  #entier aléatoire entre 1 et 100
>>> b = randint(1, 100)  #entier aléatoire entre 1 et 100
>>> a = a - b
>>> b = b + a
>>> a = b - a
>>> print(a, b)
```

Exercice 8

Le prix d'une matière première est de 873 euros la tonne au début de l'année. Ce prix subit des variations saisonnières :

- au premier trimestre il augmente de 347 euros,
- au second trimestre il augmente de 25 %,
- au troisième trimestre il subit une baisse de 50 %
- et enfin il diminue de 100 euros.
- Compléter le code ci-dessous afin qu'il calcule les valeurs successives de la variable `prix`.

Script Python

```
prix = 873        #prix au début de l'année
prix = ....       #prix à la fin du premier trimestre
prix = ....       #prix à la fin du second trimestre
prix = ....       #prix à la fin du troisième trimestre
prix = ....       #prix à la fin de l'année
print("Prix final :", prix)
```

```
1
2
3
4
5
6
```

9. Entrées / Sorties

Un programme effectue des calculs. En général, il traite des informations, et nous renvoie le résultat du traitement.

Les informations traitées doivent donc être fournies **en entrée**. Les résultat renvoyés **en sortie**.

Dans ce qui précède, les codes que nous avons écrit **affiche un résultat dans la console**. Mais ils n'avaient rien en entrée. Si on exécute plusieurs fois ces programmes, ils feront à chaque fois exactement la même chose,

puisqu'aucune donnée ne change à chaque exécution.

La grande majorité des programmes ne fonctionnent pas ainsi. En général, un programme lit des informations, les traite, et renvoie le résultat.

Les informations lues peuvent être saisies au clavier, lues dans un fichier, provenir d'un appareil photo ou scanner, enregistrées via un dispositif audio etc....

De même, le résultat peut être un simple affichage à l'écran (texte ou image), ou sur une imprimante, dans un fichier, un son etc...

Les codes précédents de ce document affichent un résultat (avec la fonction `print()` qui affiche dans la console). `print()` peut afficher n'importe quelle variable (int, float, str ou bool) ou texte.

Pour entrer une information, on peut utiliser la fonction `input()`.

9.1. Entrer des données de type texte

Script Python

```
print("-------- entrées -------")
votre_nom = input("quel est votre nom ? ")
votre_prenom = input("quel est votre prénom ? ")
print("-------- sorties -------")
print("Bonjour ", votre_prenom, votre_nom,",")
print()
print("Bienvenue à Pythonworld !")
```

Script Python

```
-------- entrées -------
quel est votre nom ? Mika
quel est votre prénom ? mm
-------- sorties -------
Bonjour  mm Mika ,

Bienvenue à Pythonworld !
```

La fonction `input(message)` :

affiche le message, attend que l'on entre une information, que l'on termine en appuyant sur la touche Entrée, et renvoie cette information. L'instruction :

Script Python

```
ma_var = intput("un message :")
```

permet donc d'affecter la donnée entrée à la variable `ma_var`

9.2. Entrer des données numériques

La méthode est la même, a ceci près : la fonction `input()` nous permet de demander d'entrer une valeur et d'affecter la valeur entrée à une variable. **Mais la valeur entrée est toujours de type str**.

Voyez ce code, et comprenez l'erreur générée à l'exécution :

Script Python

```
age = input("Entre ton age : ")
print("Tu as ", age, "ans.")
age_2050 = age + (2050 - 2022)
print("En 2050 tu auras : ", age_2050, "ans.")
```

Algo

```
Entre ton age : 22
Tu as  22 ans.
Traceback (most recent call last):
File "<input>", line 3, in <module>
TypeError: can only concatenate str (not "int") to str
```

Vous obtenez :

File "<input>", line 3, in <module>
TypeError: can only concatenate str (not "int") to str

ce qui en français donne :

Algo

```
on ne peut concaténer que des str (pas des int) à des str.
```

Pourtant `age + (2050 - 2022)` semblait bien être une **addition de deux entiers** non ?

En fait non, car `age` est de type **str**.

Si vous voulez que age soit un entier, il faut **convertir** le résultat de `input()`. Voici le code précédent, corrigé pour que l'entrée soit convertie en entier :

Script Python

```
age = int( input("Entre ton age : ") )
print("Tu as ", age, "ans.")
age_2050 = age + (2050 - 2022)
print("En 2050 tu auras : ", age_2050, "ans.")
```

Algo

```
Entre ton age : 22
Tu as  22 ans.
En 2050 tu auras :  50 ans.
```

Dans l'exemple ci-dessous que nous avons étudié où `age` était de type `str` :

Script Python

```
age = input("Entre ton age : ")
print("Tu as ", age, "ans.")
age_2050 = age + (2050 - 2022)
```

le code génèrait une erreur.

Mais ce ne sera pas forcément le cas....

Ou du moins, il y aura surement une erreur, mais le code ne va pas forcément s'arrêter, il affichera des choses étranges....

Regardez ce code, comprenez le problème, puis corrigez le code :

Script Python

```
age = input("Entre ton age : ")
print("Tu as ", age, "ans.")
age_duclos = age * 2
print("Monsieur Duclos a deux fois ton age. Il a", age_duclos, "ans.")
```

Algo

```
Entre ton age : 22
Tu as  22 ans.
Monsieur Duclos a deux fois ton age. Il a 2222 ans.
```

9.3. Que s'est-il passé ?

Si `age` est de type chaine de caractères, il est possible de faire `2 * age`. Cela revient en fait à faire `age + age`.
On dit qu'on a fait une **concaténation** des chaines de caractères. Cela revient tout simplement à les juxtaposer.

Exercice 9

La température f en degrés Fahrenheit s'obtient à partir de la température c en degrés Celsius par la formule de conversion $f = 1,8 * c + 32$.

Ecrire un programme qui réponde à la **spécification** suivante : *convertir une mesure de température de l'échelle Celsius vers l'échelle Fahrenheit.*

```
1
2
3
4
5
6
```