Thème 6 - Langages et programmation

01 Les Variables en Python

Table des matières

1. Introduction

Le langage Python est un langage de programmation objet interprété. Il a été développer par Guido Von Rossum en 1989 à l'Université d'Amsterdam.

- Python a été créé dansl'optique d'être le plus simple possible à apprendre.
- Il s'agit d'un langage dit de « haut niveau ».
- Python est un langage open source. Libre et gratuit, il est supporté, développé et utilisé par une large communauté.
  • Python est une langage multiplateforme : vous érirez le même programme que soyez sous Linux, Mac OS X ou Windows et il pourra être exécuté indifférenment sur l'un de ces systèmes d'exploitation.
- Python est un langage opérationnel complet : il ne sert pas qu'à apprendre à programmer. De nombreuses applications scientifiques complexes sont développées en Python. A la NASA, au CNRS, chez Google, chez Yahoo, etc ... de nombreux projets sont développés en Python.

Avec le langage Python il est possible de faire :

- du calcul scientifique (librairie Numpy)
- des graphiques ((librairie Mathplotlib)
- du traitement du son
- du traitement de l'image (librairie PIL)
- des applications avec interface graphique GI (librairies Tkinter, PyQt, wxPython, PyGTK, ...)
- des jeux vidéo en temps réel (librairie Pygame)
- des applications web (serveur web Zope, framework web Django ; framework JavaScript Pyjama)
- interfacer des systèmes de gestion de base de données (librairie MySQLdb...)
- des applications réseaux (framework Twisted)

1. Variables et Affectation

DEFINITIONS

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

Ainsi, l'algorithme

```
a ← 3
b ← 2 * a
```

s'écrit en Python de la façon suivante :

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

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

```
3
6
```

Question 1 :

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

```
N ← 2
P ← 3
N ← P + 1
P ← N
```

In [ ]:
```

```

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

«Incrémenter» une variable signifie l'augmenter.

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

```
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

```
compteur = compteur + 1
```

- Syntaxe Pythonesque

```
compteur += 1
```

In [1]:
```
compteur=0
compteur+=1
print(compteur)
```

```
1
```

Question 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%.

In [ ]:
```

```

3. Derouler un code : Premier algorithme

Considérons ce code :

```
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

Question 3 :

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

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

otre réponse

Double cliquez la cellule et remplacez les ... par vos réponses.

n° ligne	état
1	...
2	...
3	...
4	...

A la fin x vaut $...$

Correction :

|n° ligne|état| |:-:|:-:| |1|x = 2| |2|x = 1| |3|y = 5| |4|x = 2.5| A la fin x vaut 2.5

Question 4 :

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

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

otre réponse

Double cliquez la cellule et remplacez les ... par vos réponses.

n° ligne	état
1	...
2	...
3	...
4	...

A la fin nom vaut **...** </p> </details>

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,"**

Question 5 :

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

```
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
```

otre réponse

Double cliquez la cellule et remplacez les ... par vos réponses.

n° ligne	état
1	...
2	...
3	...
4	...

A la fin **nb_de_tour** vaut $...$ </p> </details>

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

FICHE n°1 : Vocabulaire de base

affectation Une affectation est une instruction de type : ```python var = expression ``` L'expression à droite du `=` est d'abord évaluée, puis le résultat est écrit en mémoire, et le nom de la variable permet de réutiliser cette valeur. expression Une expression n'est pas à proprement parler une instruction, c'est quelques chose que python peut évaluer (c'est a dire donner sa valeur) : Par exemple, dans : ```python var = expression ``` expression peut être un nombre, un calcul, une chaine de caractères ou des opérations plus complexes que nous verrons plus tard. dérouler un code Pour **lire et comprendre** ce que fait un code, il faut souvent **écrire sur une feuille** le déroulé de ce code. Considérons ce code : ```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|

3. L'échange de variables

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

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

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

otre réponse

Double cliquez la cellule et remplacez les ... par vos réponses.

n° ligne	état
1	...
2	...
3	...
4	...

A la fin x vaut $...$ et y vaut $...$ </p> </details>

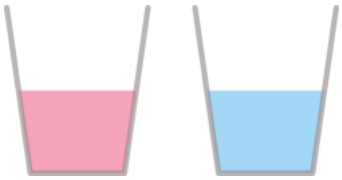
Correction :

|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 `a` (par exemple) avant que celle-ci ne se fasse écraser :

```
>>> 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 :

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

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

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

FICHE n°2 : les algos à connaitre

Algorithme d'échange #### avec une variable temporaire : ```python a = valeur_de_a b = valeur_de_b # problème : echanger les deux valeurs temp = valeur_de_a a = valeur_de_b b = temp ``` #### Il est aisé, en python, de faire cet échange de façon très simple : : ```python a, b = b, a ``` Nous expliquerons plus tard les secret de cette syntaxe, qui est propre à Python mais ne fonctionne pas dans tous les langages.

4. Types de variables

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

A RETENIR

Les types de données fondamentaux

Type	représente	exemples	remarque
str	chaîne de caractères	'A' '$' 'bonjour'	chaque caractère est codé sur 1 à 4 octet.
int	Entiers numériques	0 -1 102	En python, on ne distingue qu'un type, int, pour tous les entiers. Dans la plupart des autres langages, il existe des entiers de différentes tailles, qui peuvent être signés ou non signés, selon qu'ils supportent des valeurs négatives ou non.
float	Nombres à virgule flottante	3.14 ou 0.01	Dans de nombreux langages, on peut choisir le niveaux de précision, en fonction du type utilisé pour les trois types à virgule flottante.
bool	Booléen	True False	Ne peut représenter qu'un des deux états, vrai ou faux.

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`.

In [2]:
```
a = 1
type(a)
```

Out[2]:
```
<class 'int'>
```

In [3]:
```
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))
```

```
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.

In [5]:
```
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
```

5. Python et le typage dynamique

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

```
a = 3
```

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

```
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

```
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 ?

```
>>> 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**.

FICHE n°3 : les types

types de base |type|description| |--|--| |int|les nombres entiers| |float|les nombres avec une virgule| |str|les chaînes de caractères| |bool|les booléens| **Remarques :** Les float sont proches des nombre décimaux usuels, mais sont des valeurs approchées.
1 est un entier, et 1.0 est un float. Ils sont **égaux mais pas identiques** : 1 == 1.0 est True, ces deux écritures représentent la même valeur. 1 is 1.0 est False, ce sont bien deux objets différents
conversions d'un type à un autre On peut convertir des float en int, ou inversement. On peut également convertir des chaine **str** en **int** ou en **float** mais seulement si la chaine contient un nombre compréhensible. Exemples : a = int(1.23) # a vaudra 1 a = float(1) # a vaudra 1.0 a = str("12") # on convertit la chaine "12" en entier qui vaut 12 chaine = str(3) # on a converti l'entier 3 en str, chaine vaut "3" Mais : a = int("douze") # TypeError : int ne sait pas interpréter douze comme un nombre a = int(12.3) # TypeError : idem, 12.3 n'est pas un entier a = float(12.3) # a vaut bien 12.3 a = int(1.0) # TypeError : 1.0 non plus n'est pas un entier !

6. Bonnes pratiques de nommage

2. 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

3. 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** :

```
# 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 ?

4. 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`.

7. Les opérateurs

Les calculs mettant en jeu ce type de variables utilisent les opérateurs classiques, regroupés dans le tableau ci-dessous.

Opération	Opérateur Python
Addition	+
Soustraction	-
Multiplication	*
Division	/
Puissance	**
Division entière	//
Reste entier	%

> Vous devez absolument connaître ces opérateurs, notamment les trois derniers, vous l'aurez compris.

In [2]:
```
q = 14 // 5 # division entière
r = 14 % 5 # reste entier
print(q)
print(r)
print("14 = ", q, "*5 +", r)  # on peut afficher du texte et des variables
print(f"14 = {q} *5 + {r}")  # autre méthode
```

```
2
4
14 =  2 *5 + 4
14 = 2 *5 + 4
```

FICHE n°4 : les opérateurs

5. Calculs, opérations

Dans le tableau ci-après, sont présentés les symboles utilisés pour les opérations de base.

Opérations	Symboles	Exemples
addition	+	2 + 5 donne 7

8. Exercices

Exercice 1 :

Prédire la valeur affichée après les séquences

Réponse :

1. Séquence 1 :
2. Séquence 2 :
3. Séquence 3 :

In [1]:

Exercice 2 :

Le prix d'une matière première est de 873 euros la

In [ ]:

9. En / So

Opérations	Symboles	Exemples
soustraction	-	8 - 2 donne 6
multiplication	*	*6* * *7* donne 42
exponentiation (puissance)	**	*5* ** *3* donne 125
division	/	7 / 2 donne 3.5
reste de division entière	%	7 % 3 donne 1 car $7 = 2 \times 3 + 1$. Le reste de la division de 7 par 3 est 1
quotient de division entière	//	7 // 3 donne 2

5.1. modulo (%) et division entière (//)

L'opérateur modulo donne le reste de la division euclidienne. L'opérateur division entière donne le quotient.

exemple

```
17 divisé par 8 : il y va deux fois et il
reste 1.
17 % 8 vaut 1
17 // 8 vaut 2

Alors que :
17 / 8 vaut 2.125
```

5.2. exponentiation (**)

L'opérateur ** se lit puissance (c'est l'exponentiation)

exemple :

```
3**2 vaut 9
```

d'instructions suivantes.

1. Séquence 1 :

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

1. Séquence 2 :

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

1. Séquence 3 :

```
>>> 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)
```

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`.

```
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)
```