

2. PARTIE 2 : Le langage de programmation Python : les variables et les fonctions

2.1. Variables, affectations et fonctions

2.1.1. Affectation

Pendant l'exécution d'un programme, les données que ce dernier manipule sont stockées en mémoire centrale. En nommant ces données, on peut les manipuler beaucoup plus facilement.

Les variables nous permettent donc de manipuler ces données sans avoir à se préoccuper de l'adresse explicite qu'elles occuperont effectivement en mémoire. Pour cela, il suffit de leur choisir un nom ou identificateur.

L'interpréteur (programme de traduction) s'occupe d'attribuer une adresse à chaque variable et de superviser tous les éventuels futurs changements d'adresse. Une variable peut être assimilée à une boîte aux lettres portant un nom.

Par exemple $x = 2$ signifie : « à l'adresse mémoire référencée par x -autrement dit, dans la boîte aux lettres dénommée x -, se trouve la valeur 2 ». Nous utilisons plus fréquemment l'abus de langage « la variable x vaut 2 ».

Attention, les instructions utilisées en Python ne sont pas des instructions mathématiques. En effet :

- `>>> x = 2` n'a pas la même signification qu'en mathématiques. Il s'agit d'une affectation et non pas un symbole d'égalité.
- `>>> x = x + 1` est très fréquente en informatique en revanche, elle est inacceptable en mathématiques.

En Python, il est possible d'affecter plusieurs variables simultanément :

```
*** Console de processus distant Réinitialisée ***
>>> a=b=2
>>> a
2
>>> b
2
>>> c,d=3,1
>>> c,d
(3, 1)
```

2.2. Les fonctions

En Python une fonction est définie par l'instruction composée **def** suivie du nom de la fonction et se termine obligatoirement par deux points : puis le bloc d'instructions qu'il ne faut pas oublier d'indenter.

La notion de fonction en informatique relève du même concept qu'une fonction mathématique, c'est-à-dire qu'on définit une fonction puis on l'applique à différentes valeurs.

```
1 def carree(a) :
2     """ La fonction carree calcule la valeur du carre du parametre """
3     valeur = a**2
4     return valeur # renvoie l'image de a par la fonction carree
```

`.\scripts\part2\fonction1.py`

C'est l'indentation qui délimite le bloc d'instructions

Lorsqu'on définit la fonction `carree()`, **a est appelé paramètre** de la fonction. Lorsqu'on appelle la fonction avec une valeur explicite pour `a`, comme dans `carree(3)`, on dira plutôt que **3 est un argument** de la fonction.

En appelant la fonction `carree()` d'argument 3, on obtient 9 :

```
>>> carree(3)
9
```

- La fonction est auto-documentée par un texte entre `"""` et `"""` (c'est ce que l'on appelle le **docstring** de la fonction).

Le **docstring** d'une fonction permet de donner des informations sur la fonction.

Quand on saisit dans la console, après l'exécution de la fonction, l'instruction **help(nom de la fonction)**, python affiche le **docstring** de la fonction ce qui nous permet d'avoir des informations sur la fonction en cas d'oubli.

```
>>> help(carree)
Help on function carree in module __main__:


carree(a)
    La fonction carree calcule la valeur du carre du parametre
```

Remarque : les `"""` permettent d'avoir un commentaire sur plusieurs lignes

- La fonction se termine avec une instruction **return**. Ce qui suit le **return** est l'image des entrées par la fonction. Dès que la fonction rencontre un **return**, elle renvoie ce qui suit le return et stoppe son exécution.
- Commentaires Le symbole **#** apparaîtra à maintes reprises. Il marque le début d'un commentaire que la fin de la ligne termine. Autrement dit, un commentaire est une information aidant à la compréhension du programme mais n'en faisant pas partie.

2.3. La portée et durée de vie des variables

Il faut faire la différence entre les variables utilisées dans le programme (variables globales) et les variables utilisées dans une fonction (variables locales).

 Exercice 1 : Taper les trois fonctions dans Edupython et faire des appels avec différentes valeurs de x.

```
1 def carree(x):
    x=x**2
    return x

def carree_2():
    x=x**2
    return x

a=2
10 def carree_3():
    global a
    a=a**2
    return a
```

Normalement vous devez détecter un problème, il y a une fonction qui ne peut pas être interprétée.

Il faut privilégier les variables locales. La première définition est la définition à privilégier.

```
>>> carree(3)
9
>>> carree_2(3)
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
TypeError: carree_2() takes 0 positional arguments but 1 was given
>>> carree_3()
4
```

`.\scripts\part2\exo1_portee.py`

Une variable définie dans une fonction est dite **locale**. Son contenu ne sera pas accessible en dehors de la fonction.

Une variable définie dans le programme principal est dite **globale**. Son contenu est accessible dans tout le programme.

	Variable locale	Variable globale
Défini où	Dans la fonction	Dans le programme principal
Visible dans la fonction	Oui	Oui
Visible hors de la fonction	Non	Oui
Durée de vie	Temps d'un appel de la fonction	Temps d'exécution du programme

2.4. Exercices

2.4.1. L'affectation

L'affectation se fait par l'intermédiaire du symbole =.

Vous effectuerez les différents exercices dans la console :

1. Affecter à la variable **a** la valeur 5 et à **b** la valeur 3. Affecter à la variable **c** la somme de **a** et de **b**.
2. Affecter en simultané les valeurs 1 et 2 aux variables **a** et **b**.
3. Déterminer le type de la variable **a** en utilisant l'instruction `type()`

4. Déterminer les types des variables ci-dessous :

```
a = 6
b = 4.5
c = "hello world !"
d = True
e = (1,4,5,6)
f = [3,5,7]
g = {"Paul":7,"Marc":17,"Marie":8}
h = {2,6,7,9}
```

`.\scripts\part2\tp1.py`

2.4.2. Les fonctions

Les exercices sont à réaliser dans la zone de saisie du programme. il faudra ensuite interpréter le programme en cliquant sur la flèche verte (ou CTRL + F9)

- A. Créer une fonction `cube(a)` qui prend une valeur comme paramètre et renvoie le cube de cette valeur.

`.\scripts\part2\fonctionCube.py`

- B. Commenter ce programme et créer un **docstring** expliquant le programme.

`.\scripts\part2\fonctionCubeCommentee.py`

- A. A partir de la fonction `cube(a)`, calculer a^6 .

- B. Il est possible d'utiliser une fonction comme paramètre.

- a) Créer une fonction nommée `f()` prenant un paramètre et renvoyant ce paramètre multiplié par 2.

`.\scripts\part2\fonction_f.py`

- b) Créer une fonction nommée `g()` prenant un paramètre et renvoyant ce paramètre auquel on a additionné 2.

`.\scripts\part2\fonction_fg.py`

- c) Créer une fonction nommée `cascade()` qui prend une valeur et deux fonctions en paramètres et effectue les fonctions `f` et `g` l'une à la suite de l'autre. Tester la fonction `cascade(x,f,g)` et `cascade(x,g,f)` et expliquer la différence entre les deux.

`.\scripts\part2\fonction_fg_cascade.py`

2.4.3. Tester une fonction

En phase de codage, il est souvent utile de savoir si une fonction est correctement codée. On peut le vérifier en utilisant des assertions, avec l'instruction **assert** :

Assert condition , " commentaire "

- Si la condition est respectée, il ne se passe rien et le reste du code s'exécute.
- Si la condition n'est pas respectée, le code cesse de s'exécuter et un message d'erreur apparaît.

Cette instruction peut être utilisée à l'intérieur de la fonction pour tester les **préconditions** et à l'extérieur pour tester les **postconditions**.

```
def division(a,b):
    assert (b != 0),"Le denominateur doit etre non nul !"
    return(a//b)
```

`.\scripts\part2\tp2.py`

Tester la fonction et analyser le résultat.

- a) Écrire un assert vérifiant qu'une variable a est supérieure ou égale à 0

`.\scripts\part2\tp2_assert_entier.py`

- b) Écrire un assert vérifiant qu'une valeur a est du type entier

`.\scripts\part2\tp2_assert_sup0.py`

2.4.4. Différence entre fonction et procédure

La différence entre fonction et procédure vient du fait que la procédure ne renvoie rien.

Cela arrive lorsque la fonction appelée n'a pas vocation à renvoyer une valeur mais peut-être uniquement à afficher une valeur.

```
def affichage(nom,prenom):
    print("Bonjour ",prenom,end=" ")
    print(nom)
```

`.\scripts\part2\tp3.py`

A noter :

- Les chaînes de caractères à imprimer et à ne pas interpréter sont mis `" "` entre
- Il est possible de mixer du texte non interprété et du code à interpréter. Il suffit de les séparer avec des `;`
- Dans le cas où on ne veut pas revenir à la ligne à la fin d'un `print`, il suffit d'ajouter `end="<séparateur>"` et remplacer le `<séparateur>` par le signe voulu.

- C. Ajouter un assert permettant de vérifier que les arguments **nom** et **prenom** sont bien des chaînes de caractères.

`.\scripts\part2\tp3_assert_str.py`

2.4.5. Exercices d'application

- D. Écrire une fonction `moyenne()` qui renvoie la moyenne de 4 nombres passés en argument.

`.\scripts\part2\moyenne.py`

- E. a) Écrire une fonction `pente()` prenant comme paramètre les coordonnées de deux points A et B et calcule la pente de la droite passant par ces deux points.

`.\scripts\part2\pente.py`

- a) Écrire une fonction `ordonneeOrigine()` qui prend comme paramètre les coordonnées de deux points A et B et renvoie l'ordonnée à l'origine de la droite passant par ces deux points. On pourra réutiliser la fonction `pente()`

`.\scripts\part2\ordonneeOrigine.py`

- b) Ajouter un assert pour vérifier que les deux points A et B n'ont pas la même abscisse.

`.\scripts\part2\ordonneeOrigine_assert.py`

2.4.6. Utiliser une bibliothèque

- F. Écrire une fonction `CirconferenceCercle()` prenant comme paramètre le rayon d'un cercle et renvoyant sa circonférence.

La valeur π n'étant pas connue par Python, il faut lui apprendre en commençant le programme par :

```
from math imports *
```

`.\scripts\part2\CirconferenceCercle.py`

Éléments de cours :

Pour développer de nouveaux algorithmes on se base généralement sur d'autres algorithmes déjà prêts, qui traitent une partie du problème et facilitent la solution. En programmation on utilise la notion de bibliothèque (library), fournissant des fonctions, des données, des classes (Programmation Objet).

La bibliothèque math regroupe toutes les fonctions utiles en mathématiques

Pour utiliser une fonction présente dans un module (bibliothèque), il faut :

Importer le module

Appeler la fonction

Code d'importation du module	Code d'appel de la fonction
<code>import module</code>	<code>module.fonction()</code>
<code>from module import fonction</code>	<code>fonction()</code>
<code>From module import *</code>	<code>fonction()</code>

- G. Écrire une fonction `norme()` prenant comme paramètre les coordonnées de deux points A et B et → renvoyant la norme du vecteur AB

`.\scripts\part2\norme.py`