Python 2: Les variables et quelques instructions simples

Définitions :

Les variables représentent les informations importantes d'un problème.

On essaie de choisir un nom de variable pas trop long mais qui donne déjà une idée de ce qu'elle représente.

Si on cherche la hauteur d'un arbre, la variable peut s'appeler h ou hauteur ; éviter de l'appeler hauteur_d'un_arbre, car si on appelle souvent cette variable, cela risque de faire des longueurs inutiles dans le programme.

Si on doit écrire quand même une variable longue entre chaque partie, mettre le tiret du 8 plutôt qu'un espace, ça évite les erreurs d'espaces qui peuvent être confondues avec une indentation et c'est mieux accepté par les interpréteurs.

La variable s'écrit toujours à gauche d'une expression avec un égal ; ex : a = 3

La variable peut être une majuscule ou une minuscule, ou une combinaison

La variable doit être définie avant de lancer le programme sinon celui-ci reverra une erreur.

La variable peut contenir un nombre, mais ne peut commencer par un nombre ; ex 5a = 8 renverra une erreur.

Attention : il y a des noms réservés au langage python qu'on ne peut pas donner comme nom de variable ; ex def sert dans python pour définir une fonction ; on ne pourra pas choisir def comme nom pour une par exemple, sinon ça renverra une erreur.

Certains caractères spéciaux sont aussi interdits dans un nom de variable (' «]…) voir la liste dans python. Ils sont utilisés par des fonctions ou des instructions python.

Types de variables :

1) Les variables peuvent contenir des nombres entiers <class 'int'> :

Ex : >>> a = 3 affecte la valeur 3 à la variable a

Ex : >>> a = int (input(« entrer un nombre : »)) attend un nombre entier ; si on entre un décimal, ça renvoie une erreur.

Ex : >>> a = input(« entrer un nombre : »)
>>> print (int(a)) renvoie a si a est un entier ou une erreur si a est un décimal

2) Les variables peuvent contenir des nombres décimaux appelés flottants<class 'float'> :

Ex : >>> a = float(input("entrer un nombre :")) attend un nombre décimal ;

>>> print(a) ; si on entre un entier, ça fonctionne car 3 = 3.0 en flottant. Si a vaut 2.54, ça écrira 2.54.

On peut obtenir la partie entière d'un flottant en écrivant :

```
>>> a = float(input("entrer un nombre :"))
>>> a = int(a)
>>> print(a)
```

La division de 2 entiers donne forcément un flottant :

Ex : >>> a = 3/4
>>> print(a)
donne 0.75

>>> print(int(a)) donne 0 qui est la partie entière du résultat de la division 3/4

La division euclidienne donne également la partie entière du résultat d'une division :

Ex : nbBoites = 567 // 12 # le double slash (//) est le symbole de cette division euclidienne
nbReste = 567 % 12 # le pourcent (%) est le symbole du dernier reste entier de la division
print(nbBoites)
print(nbReste)

Tester ce programme : vous devez obtenir 47 boites reste 3 boites ; vérifions : 47 x 12 + 3 donne bien 567 boites

Ex : test de Parité

```
nombre = int(input("entrer un nombre entier : "))
if (nombre % 2) == 0 :
   print("Le nombre est pair")
else :
    print("Le nombre est impair")
```

3) les variables peuvent représenter des booléens <class 'bool'>:

Ce sont des comparaisons chiffrées ou des comparaisons de variables ou de équations logiques dont le résultat donnera vrai ou faux.

Ex : a = (4/5 >1) donnera false car cette comparaison est fausse

4) les variables peuvent être des listes <class 'list'>:

```
>>> a = ["bleu" ,2,"canard3", 4]
>>> print(a)
```

Ça imprimera la liste ['bleu', 2, 'canard3', 4]. Les parties entre guillemets sont des chaines de caractères. L'absence de guillemets donnerait une erreur car bleu ou canard3 seraient des variables qu'il aurait fallu définir.

5) les variables peuvent être des chaines de caractères <class 'str'>:

Ex : a = "voiture" correspond à la chaine de caractères voiture.

Il existe plusieurs façons d'écrire une chaîne de caractères :

- entre guillemets ("ceci est une chaîne de caractères") ;
- entre apostrophes ('ceci est une chaîne de caractères') ;
- entre triples guillemets ("""ceci est une chaîne de caractères""").

On peut stocker une chaîne de caractères dans une variable (ma_chaine = "Bonjour, les enfants !")

Si vous utilisez les délimiteurs simples (le guillemet ou l'apostrophe) pour encadrer une chaîne de caractères, il se pose le problème des guillemets ou apostrophes que peut contenir ladite chaîne. Par exemple, si vous tapez :

chaine = 'J'aime le foot !', vous obtenez le message suivant :

```
File "<stdin>", line 1
chaine = 'J'aime le foot !'
^
SyntaxError: invalid syntax
```

Ceci est dû au fait que l'apostrophe de « J'aime » est considérée par Python comme la fin de la chaîne et qu'il ne sait pas quoi faire de tout ce qui se trouve au-delà.
Pour pallier ce problème, il faut **échapper** les apostrophes se trouvant au cœur de la chaîne **(c'est-à-dire faire en sorte qu'elles ne soient pas considérées comme fin de chaines)**. On insère ainsi un caractère anti-slash « \ » avant les apostrophes contenues dans le message.

chaine = 'J\'aime le foot !'

On doit également échapper les guillemets si on utilise les guillemets comme délimiteurs.

chaine2 = "\"Dans la vie, y\'\a pas de grands, y\'\a pas de petits…la bonne longueur pour les jambes, c\'\est quand les pieds touchent bien par terre (...)\" (Coluche)"

L'interpréteur affiche les sauts de lignes comme on les saisit, c'est-à-dire sous forme de « \n ».

Utiliser les triples guillemets pour encadrer une chaîne de caractères dispense d'échapper les guillemets et apostrophes, et permet d'écrire plusieurs lignes sans symboliser les retours à la ligne au moyen de « \n ».

```
>>> chaine3 = """Ceci est un nouvel

... essai sur plusieurs

... lignes"""

>>>
```

Notez que les trois chevrons sont remplacés par trois points : cela signifie que l'interpréteur considère que vous n'avez pas fini d'écrire cette instruction qui ne s'achève qu'une fois la chaîne refermée avec trois nouveaux guillemets. Les sauts de lignes seront automatiquement remplacés, dans la chaîne, par des « \n ».

Vous pouvez utiliser, à la place des trois guillemets, trois apostrophes qui jouent exactement le même rôle.

Un petit plus

Incrémentation des variables : L'incrémentation désigne l'augmentation de la valeur d'une variable d'un certain nombre. Exemple : pour augmenter une variable de 1 :

variable = variable + 1

Les programmeurs préfèrent utiliser :

variable += 1

L'opérateur **+=** revient à ajouter à la variable la valeur qui suit l'opérateur. Les opérateurs - =, * = et / = sont également utilisés.

Quelques trucs et astuces pour vous faciliter la vie

Python propose un moyen simple de permuter deux variables (échanger leur valeur). Dans d'autres langages, il est nécessaire de passer par une troisième variable qui retient l'une des deux valeurs… ici c'est bien plus simple :

```
>>> a = 5
>>> b = 32
>>> a,b = b,a # permutation
>>> a
32
>>> b
5
>>>
```

Comme vous le voyez, après l'exécution de la ligne 3, les variables **a** et **b** ont échangé leurs valeurs.

On peut aussi affecter assez simplement une même valeur à plusieurs variables :

```
>>> x = y = 3
>>> x
3
>>> y
3
>>>
```

On peut couper une instruction Python, pour l'écrire sur deux lignes ou plus.

```
>>> 1 + 4 - 3 * 19 + 33 - 45 * 2 + (8 - 3) \
... -6 + 23.5
-86.5
>>>
```

Comme vous le voyez, le symbole « \ » permet, avant un saut de ligne, d'indiquer à Python que « cette instruction se poursuit à la ligne suivante ». Vous pouvez ainsi morceler votre instruction sur plusieurs lignes.

Applications : quelques petits programmes mathématiques

Triangle isocèle, rectangle, équilatéral, qui suis-je ? :

Nous allons établir un programme en python qui va permettre de dire, en fonction de la longueur de ses cotés, quel est la nature d'un triangle.

Ecriture d'un algorithme correspondant à la situation :

Entrer la valeur du coté 1 du triangle.

Entrer la valeur du coté 2 du triangle.

Entrer la valeur du coté 3 du triangle.

Si les 3 cotés sont égaux alors écrire : le triangle est équilatéral.

Sinon si 2 cotés seulement sont égaux, écrire : on a un triangle isocèle.

Sinon si il a 2 cotés égaux et qu'il vérifie le théorème de Pythagore : il est isocèle rectangle

Sinon si le carré d'un côté est égal à la somme des carrés des 2 autres côtés, écrire : on a un triangle rectangle.

Dans tous les autres cas écrire : le triangle est quelconque.

Programme Python :

```python
# A quel type de triangle a-t-on affaire ?
# On commence par demander les 3 longueurs
a=float(input("Ecrire la longueur de premier coté : "))
b=float(input("Ecrire la longueur de deuxième coté : "))
c=float(input("Ecrire la longueur de troisième coté : "))
# Test triangle équilatéral
if a==b and b==c:
    print("Les 3 cotés sont égaux ; Ce triangle est équilatéral")
# Test isocèle rectangle
elif a==b or b==c or a==c and a**2+b**2==c**2 or a**2+c**2==b**2 or a**2==b**2+c**2:
    print("C'est un triangle isocèle rectangle")
# Test isocèle
elif a==b or b==c or a==c:
    print("Avec 2 cotés egaux, ce triangle est isocèle")
# Test rectangle
elif a**2+b**2==c**2 or a**2+c**2==b**2 or a**2==b**2+c**2:
    print("C'est un triangle rectangle")
# Si aucun des tests n'est validé, le triangle est quelconque
else:
    print("C triangle est quelconque")
```

Dans ce programme, on a des variables numériques et des tests utilisant des booléens.

On remarque que le mélange des expressions et des tests est tout à fait possible.