Python - Variables, types numériques et entrées / sorties dans la console

[Mise à jour le : 19/7/2021]

- **Sources**
  - **Documentation** sur Python.org : référence du langage, analyse lexicale, fonctions natives (built-in)
- **Lectures connexes**
  - **Real Python**
    - Basic Data Types in Python
    - How to Use the Python or Operator
    - How to Use sorted() and sort() in Python
  - **Developpez.com** - Variables locales et globales
- **Mots-clés** : programme, donnée, mémoire, variable, déclarer, initialiser

Les mots ci-dessous sont dits “réservés”. Ils ne peuvent pas être utilisés comme nom de variable. Les mots soulignés sont une nouveauté de Python 3. Les mots en **gras** sont utilisés dans cette page.

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

- Fonctions natives **(built-in)**[1] utilisées dans les exemples : **type**(), **isinstance**(), **float**(), **print**().

---

1. Les variables

Pendant l'exécution d'un **programme**, les **données** qu'il manipule sont stockées en **mémoire**. Les **variables** permettent de manipuler ces données sans se préoccuper de leur position. Pour cela, il suffit de leur donner un nom (les **déclarer**).

1.1 Nommage des variables

- Une variable doit respecter quelques règles de syntaxe :
  1. Le nom d'une variable ne peut être composé que de **lettres**, de **chiffres** et de "_".
  2. Le nom d'une variable ne peut pas commencer par un chiffre.
  3. Python est **sensible à la casse** (position =/= Position)
  4. L'**usage** veut que l'on se limite à des caractères minuscules (underscore case) !

1.2 Déclaration et initialisation

En Python, la simple **déclaration** d'une variable ne suffit pas à la créer. Après avoir choisi son nom, il est nécessaire de lui **affecter** une **valeur initiale** (**initialisation**). En Python l'**affectation** s'effectue avec le symbole "**=**".

exvar1.py

```
position_x = 10 # variable de type entier nommée selon la convention inderscore case
pi = 1.14159 # variable de type réel
hello = "Hello world !" # variable de type chaîne de caractère
position_x, pi, hello = 10,1.14159,"Hello world !" # C'est possible !
```

1.3 Le type des variables

- **Source** : Types natifs

Le **type d'une variable** identifie la **nature de son contenu** : nombre entier, nombre décimal, texte, etc.. Ces différents types d'information peuvent être placés dans une variable.

En Python, il n'est pas nécessaire de préciser le type d'une variable. Python est dit **typé *dynamiquement***.

- On dispose de quatre **types numériques** en Python :
  - Les entiers (**int**)
  - Les décimaux (**float**)
  - Les nombres complexes (**complex**)
  - Les booléens (**bool**) sous ensemble des entiers

Les **entiers** sont donnés **sans perte de précision**.

- **Connaître le type d'une variable** : pour cela, on utilise les fonctions natives ***type()*** et ***isinstance()***.

exvar2a.py

```
nom_variable = 15
type(nom_variable) # renvoie <class 'int'> dans la console
isinstance(15,int) # renvoie true
type(4+5j) # renvoie <class 'complex'>
```

- **Convertir le type d'une variable**
  - Convertir une **chaîne de caractères** en un entier avec la fonction ***int()***

    exvar2b.py

    ```
    annee = "2019" # renvoie la chaîne de caractères '2019' dans la console
    annee = int(annee) # renvoie le nombre 2019 dans la console
    ```

  - Convertir un **entier** en une chaîne de caractères avec la fonction ***str()***

    exvar2c.py

    ```
    annee = 2019 # renvoie le nombre 2019 dans la console
    annee = str(annee) # renvoie la chaîne de caractères '2019'
    ```

  - **float()** : permet la transformation en flottant.
  - **long()** : transforme une valeur en long.

1.4 Copie de variables

Le contenu d'une variable *var1* peut être placé dans une variable *var2*.

exvar3.py

```
var2 = var1
```

1.5 Permutation

Python propose un moyen simple pour permuter deux variables (échanger leur valeur).

exvar4.py

```
a = 5
b = 32
```

```
a,b = b,a # résultat a = 32, b = 5
```

1.6 Opérations arithmétiques et logiques

- **Source** : Opérateurs
- **Opérations arithmétiques**

addition	soustraction	multiplication	division euclidienne	division entière	reste division	puissance
+	-	*	/	/ /	%	* *

- *Exemple sur des entiers*

exvar5a.py

```
var1 = 5
var2 = 1
var3 = var1 + var2 # résultat var3 = 6
```

- *Exemple sur des flottants*

- exvar5b.py

```
var1 = 5.2
var2 = 1.4
var3 = var1 + var2 # résultat var3 = 6.6
```

- **Opérations booléennes**

conjonction (ET)	Disjonction (OU)	Négation (NON)
and	or	not

Les variables booléennes prennent les valeurs **True** et **False** ou **1** et **0**

Exemples

exvar6a.py

```
bool1, bool2 = True, False
bool3, bool4 = 1, 0
bool5 = bool1 and bool2 # résultat : bool5 = False
bool5 = bool1 or bool2  # résultat : bool6 = True
bool7 = not(bool1)      # résultat : bool7 = False
bool8 = bool3 & bool4   # résultat : bool8 = 0
```

- **Opérations au niveau bit**

ET	OU	NON	OU EXCLUSIF	Décalage à gauche	Décalage à droite
&	|	~	^	«	»

Les opérateurs de niveau bit s'appliquent sur chacun des **bits** d'un mot.
Exemple si A = a_1a_0 et B = b_1b_0 alors C = A & B ⇒ C = c_1c_0 tel que $c_1 = a_1 \& b_1$ et $c_0 = a_0 \& b_0$

Exemples

exvar6b.py

```
# 0b précise que le nombre qui suit est exprimé en binaire
var1 = 0b01010101    # résultat var1 = 85 en base 10
var2 = 0b10101010    # résultat var2 = 170 en base 10
var3 = var1 & var2   # résultat var3 = 0 en base 10, bin(var3) =
0b00000000
var4 = var1 | var2   # résultat var4 = 255 en base 10, bin(var4) =
0b11111111
var5 = ~var1         # résultat var5 = -86 en base 10 car l'ordinateur
calcule en complément à 2
                     # or 2**n + complément à 2 de x = x ici = = 8 donc
256-86 = 170 = 0b10101010 = var2
var6 = var1 ^ bool2  # résultat bin(var6) = 0b11111111
var7 = var1 << 1     # résultat var7 = 170 (décalage à gauche de 1 =>
multiplication entière par 2)
var8 = var2 >> 1     # résultat var8 = 85 (décalage à droite de 1 =>
division entière par 2)
```

- **Base 2, 10, 16**

Par défaut, les nombres entiers saisis ou affichés sont en base 10.

Pour manipuler des séquences de bits, de longueur arbitraire on utilise **0b** devant la valeur binaire ou **0x** devant la valeur hexadécimale.La fonction **bin**(**n**) renvoie la valeur de n en binaire. La fonction **hex**(**n**) renvoie la valeur de n en hexadécimale. Ces fonctions renvoient des chaînes de caractères.

Exemples

exvar6c.py

```
bin(43)          # renvoie '0b101011'
bin(0xf4)        #         '0b11110100'
hex(43)          #         '0x2b'
```

```
hex(0b11110100) #         '0xf4'
```

1.7 Variable sans valeur

On peut réinitialiser une variable en l'affectant d'une valeur **vide** avec le mot ***None***.

1.8 Portée des variables

> La **portée** d'une variable est la portion de code source où elle est accessible.

Pour connaître la portée d'une variable on utilise la règle **LEGB** :
Localement (variable déclarée dans une fonction)
Englobante (variable déclarée dans une fonction qui contient la fonction où elle est appelée)
Globalement (variable déclarée globalement)
Builtins (est une variable Built-in

- **Espace local**

Les variables définies dans un corps de fonction ou passées en paramètres sont seulement accessibles dans le corps de la fonction.

Exemple

*.py

```
# Espace local au programme
valext = 5 # variable connue dans et hors de la fonction

def func(valint): # ici valint <- valext, valint : variable connue
seulement dans la fonction
    # Espace local à la fonction func
    valint = valint * 2
    # valext est connue de func bien que sa déclaration soit à
l'extérieure
    print("Dans la fonction func, valext = ", valext)
    # REMARQUES
    # valint = valext * 2 est possible, mais déconseillé par les bonnes
pratiques de programmation
    # valext = valext * 2 est INTERDIT, car on ne peut pas modifier une
variable extérieure à l'espace local
    print("Dans la fonction func, valint * 2 = ", valint," car valint
```

```
<- valext lors de l'appel")
    return valint

print("Avant l'appel de func, valext = ", valext)
valext = func(valext) # valext est connue de func bien que sa
déclaration soit à l'extérieure
print("Après l'appel de func et l'opération valext = func(valext),
valext = ", valext)
print(valint) # valint n'est pas connue à l'extérieure de func
```

Résultat attendu

> Avant l'appel de func, valext = 5
> Dans la fonction func, valext = 5
> Dans la fonction func, valint * 2 = 10
> Après l'appel de func et l'opération valext = func(valext), valext = 10 car valint ← valext lors de l'appel
> **Une exception s'est produite : NameError**
> **name 'valint' is not defined**

Une fonction ne peut pas modifier la valeur d'une variable extérieure à son espace local par une affectation.

- **Variable globale**

Pour modifier une variable extérieure à une fonction, on la qualifie de **globale**.

Exemple

*.py

```
# Espace local au programme
valext = 5 # variable connue dans et hors de la fonction

def func(): # ici valint <- valext, valint : variable connue seulement
dans la fonction
    # Espace local à la fonction func
    global valext # A éviter sauf cas particulier
    return valext * 2

print("Après l'appel de func, valext = ", func())
```

2. Entrée / sortie dans la console

> Il est fréquent qu'un utilisateur ait besoin de saisir des informations lors de l'exécution d'un programme. Les fonctions **print()** et **input()** permettent d'interragir avec l'utilisateur dans la console.

2.1 Afficher le contenu des variables

On affiche le contenu d'une ou plusieurs variables dans console avec la fonction ***print()***.

Exemple

exvar7.py

```
nom = "Dupont"
age = 35
print("Je suis", nom, "et j'ai", age, "ans.") # Retourne : Je suis Dupont et j'ai 35 ans. dans la console
```

Par défaut l'instruction **print** provoque un retour à la ligne après l'affichage. On peut changer ce comportement en fournissant une autre chaîne de caractère à accoler à l'affichage comme ci-dessus ou même rien.

Exemple

*.py

```
print("Ce texte s'affiche", end=" ") # Retourne : Ce texte s'affiche sur une seule ligne.
print("sur une seule ligne.", end=" ")
```

2.2 Saisir une donnée utilisateur

On entre une donnée utilisateur dans la console avec la fonction ***input()***.

exvar8.py

```
nom = input("Quel est votre nom ?")
print("Bonjour",nom)
```

Résumé

- Les variables **conservent les données** du programme lors de son exécution. Leur contenu peut changer. Il faut éviter de mettre des espaces et des accents dans les noms de variable.
- Pour affecter une valeur à une variable, on utilise la syntaxe: *nomVariable* **=** *valeur*.
- Il existe différents types de variables : **entier**, **réel**, **chaîne de caractères**, etc.
- Les variables **locales**, définies avant l'appel d'une fonction, sont accessibles en lecture seule depuis l'appel de la fonction. Une variable locale définie dans une fonction est supprimée après l'exécution de la fonction.
- Les variables **globales** se définissent à l'aide du mot-clé ***globale*** suivi du nom de la variable préalablement créée. Elles peuvent être modifiées dans le corps d'une fonction.
- Pour afficher une donnée dans la console, on utilise la fonction **print()**.
- Pour entrer une donnée utilisateur dans la console, on utilise la fonction **input()**.

Quiz

- Python Variables Quiz
- Basic Data Types in Python Quiz
- Rounding Numbers in Python Quiz

Plus loin ...

- Namespaces and Scope in Python

[1)]
Fonctions toujours disponibles.