# Variables et opérateurs

[Mise à jour le : 6/7/2022]

En cours de rédaction

**Mots-clés**

programme, donnée, mémoire, variable, déclarer, initialiser, portée, opérateurs, arithmétique, logique, comparaison.

## 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

Une variable doit respecter quelques **règles de syntaxe**.

- Python
- C, C++ (Arduino)
- JavaScript
- PHP
- C#

En Python, le nom d'une variable ne peut être composé que de **lettres**, de **chiffres** et de "_". Le nom d'une variable ne peut pas commencer par un chiffre. Python est **sensible à la casse** (position =/= Position). L'**usage** veut qu'on se limite à des caractères minuscules underscore case !

En C, C++ (Arduino), le nom d'une variable ne peut être composé que de **lettres**, de **chiffres** et de "_". Le nom d'une variable doit commencer par une lettre. C, C++ (Arduino) sont **sensibles à la casse** (position =/= Position).

En JavaScript le nom d'une variable commence par une **lettre** ou par **$**.

*A faire*

*A faire*

### 1.2 Déclaration et initialisation

Pour **exister**, une variable doit être **déclarée**.

- **Algorithmique**

**var**
variable1 : type // *Déclaration de variable1, variable2 et variable 3*
variable2, variable3 : type
variable4 ← valeur // *Initialisation de variable4 avec valeur*

- **Programmation**

- Python
- C, C++ (Arduino)
- JavaScript
- PHP
- C#

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)**. Dans ce langage l'affectation s'effectue avec le symbole “**=**”.

*Exemple*

.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 !
```

En C, C++ (Arduino), on déclare une variable en précisant son **type**, suivi d'un **espace** puis son **nom**. La déclaration se termine par un **point-virgule**.

.cpp

```
int noteDeMaths;
double sommeArgentRecue;
unsigned int nombreDeLecteursEnTrainDeLireUnNomDeVariableUnPeuLong;
```

En JavaScript, une variable est déclarée explicitement par le mot clé **var**. Il est possible, mais pas obligatoire d’initialiser une variable (lui attribuer une valeur) lors de sa déclaration.

.js

```
var i ; // variable non initialisée
var etat = true : // type booléen
var $Somme = 400; // type entier
var maMoyenneEnTsin = 12.5 ; // type nombre réel
var monPrenom = "Lucas" ; // type chaine de caractères
```

*A faire*

A faire

### 1.3 Types

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.

- **Programmation**

- Python
- C, C++ (Arduino)
- JavaScript
- C#

- **Source** : Types natifs

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

- Les principaux types sont :
  - Les entiers (**int**)
  - Les décimaux (**float**)
  - Les nombres complexes (**complex**)
  - Les booléens (**bool**), sous-ensemble des entiers
  - Les chaînes de caractères (**str**)
  - Les listes (**list**)

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

- **Connaître** le type d'une variable avec **type**() et **isinstance**().

*Exemple*

exvar2a.py

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

```
type(4+5j) # renvoie <class 'complex'>
```

- **Convertir** une **chaîne de caractères** en un entier avec ***int()***

*Exemple*

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 ***str()***

*Exemple*

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.

A faire
A faire
A faire

### 1.4 Copie

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

- Python
- C, C++ (Arduino)
- JavaScript
- C#

*Exemple*

exvar3.py

```
var2 = var1
```

A faire
A faire
A faire

### 1.5 Permutation

> Une permutation consiste à **intervertir** les valeurs de deux variables

- Python
- C, C++ (Arduino)
- JavaScript
- C#

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

*Exemple*

exvar4.py

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

A faire
A faire
A faire

### 1.6 Variable sans valeur

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

### 1.7 Portée

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

- Python
- C, C++ (Arduino)
- JavaScript
- C#

Pour connaître la portée d'une variable on utilise la règle **LEGB** :
**L**ocalement (variable déclarée dans une fonction)
**E**nglobante (variable déclarée dans une fonction qui contient la fonction où elle est appelée)
**G**lobalement (variable déclarée globalement)
**B**uiltins (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())
```

A faire
A faire
A faire

## 2. Les opérateurs et les calculs

### 2.1 L'affectation

En algorithmique et en programmation informatique, une affectation, aussi appelée **assignation** (anglicisme), est une structure qui permet d'**attribuer une valeur à une variable**. Wikipédia

En algorithmique, on utilise le symbole flèche : ←

*Exemple*
variable ← 15

Les langages ci-dessous utilisent le signe **=** pour affecter une valeur à une variable.

- Python
- C, C++ (Arduino)
- JavaScript
- C#

- **Source** : Assignment Operators

- * **Source** : Opérateurs d'affectation

- **Source** : Expressions et opérateurs

- **Source** : Opérateurs et expressions c# (référence C#)

### 2.2 Les opérateurs arithmétiques

Dans un algorithme, on écrira :
addition
**+**
soustraction
**-**
multiplication
* ou **x**
division euclid.
**/**
division entière
**DIV**
reste division
**%** ou **mod**
x puissance n

> **x**$^{n}$
> etc.

- Python
- C, C++ (Arduino)
- JavaScript
- C#

- **Source** : Arithmetic Operators

> En Python, on utilise :
> - * pour la multiplication,
> - ** pour la puissance,
> - // pour la division entière et
> - % pour le reste de la division.

- *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
```

- **Source** : Les opérateurs mathématiques

- **Source** : Opérateurs arithmétiques

- **Source** : Opérateurs arithmétiques (référence C#)

### 2.3 Les opérateurs booléens

Les variables booléennes prennent les valeurs **VRAI** et **FAUX** ou **1** et **0**.
Dans un algorithme, on écrira :
Conjonction
**ET**
Disjonction
**OU**
Négation
**NON**

- Python
- C, C++ (Arduino)
- JavaScript
- C#

- **Sources**
  - Opérateurs booléens
  - Using the "not" Boolean Operator in Python
  - Using the "and" Boolean Operator in Python
  - Using the "or" Boolean Operator in Python

En Python, **and** est la conjonction, **or** la disjonction et **not** la négation.

*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
```

- **Source** : Les opérateurs logiques
- **Source** : Opérateurs logiques
- **Source** : Opérateurs logiques booléens (référence C#)

### 2.4 Les opérateurs de comparaison

> Un opérateur de comparaison compare ses deux opérandes et renvoie un valeur booléenne correspondant au résultat de la comparaison (vraie ou fausse). Les opérandes peuvent être des nombres, des chaînes de caractères, des booléens ou des objets.

- Python
- C, C++ (Arduino)
- JavaScript
- C#

- **Source** : Opérateurs relationnels

- **Source** : Les opérateurs comparatifs

- **Source** : Opérateurs de comparaison

- **Source** : Opérateurs de comparaison (référence C#)

### 2.5 Les opérateurs de niveau bit

> En logique, une **opération bit à bit** est un calcul manipulant les données directement au niveau des bits, selon une arithmétique booléenne. Elles sont utiles dès qu'il s'agit de manipuler les données à bas niveau : codages, couches basses du réseau (par exemple TCP/IP), cryptographie,etc. Wikipédia

- Python
- C, C++ (Arduino)
- JavaScript
- C#

- **Source** : Bitwise Operators

ET
**&**
OU
|
NON
~

OU EXCLUSIF
^
Décalage à GAUCHE
«
Décalage à DROITE
»

*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'
```

- **Source** : Opérateurs bit à bit
- **Source** : Opérateurs binaires
- **Source** : Opérateurs au niveau du bit et opérateurs de décalage (référence C#)

## 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.

---

From:
http://webge.fr/dokuwiki/ - **WEBGE Wikis**

Permanent link:
**http://webge.fr/dokuwiki/doku.php?id=info:prog:varop&rev=1657122144**

Last update: **2022/07/06 17:42**