

1^{ère} NSI – Thème « #3 »

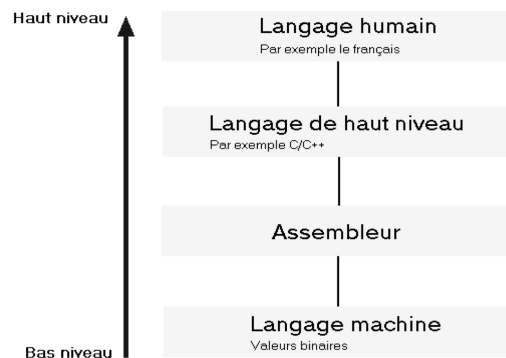
Introduction à la Programmation *(en Python)*

Marc Biver, Septembre 2023

Lycée Fustel de Coulanges, Massy

Qu'est-ce qu'un langage de programmation ?

- Programmer = demander à un ordinateur d'effectuer des actions.
- Exemple : « calcule-moi la valeur de $6 + 7$ ».
- Problème : un ordinateur ne « parle » que le binaire ; nous, les humains, pas du tout...
- Solution : on a créé des langages dits « évolués » ou « haut niveau » qui nous permettent d'exprimer de manière naturelle ce que l'on veut que l'ordinateur réalise.



- Un langage de programmation...
 - Comporte une syntaxe, du vocabulaire, une grammaire... comme une langue humaine.
 - A l'inverse d'une langue humaine ils n'ont qu'une vocation très étroite : écrire des algorithmes de manière à ce qu'un ordinateur puisse les exécuter.
 - La « traduction » vers le langage machine que comprend l'ordinateur peut se faire de deux manières :
 - La compilation : l'équivalent de la traduction d'un livre – cela produit un nouveau fichier, appelé « exécutable » qui peut être directement exécuté par l'ordinateur (comme un livre traduit peut être directement lu).

- L'interprétation : comme « dans la vraie vie » c'est une traduction au fur et à mesure que le code est écrit, sans enregistrement du code traduit dans un nouveau fichier.
- Exemples :
 - Le langage C utilise un compilateur – on dit que c'est un langage compilé.
 - Le langage Python utilise un interpréteur – on dit que c'est un langage interprété.

Le langage Python

Pourquoi l'avoir choisi ?

- A notre niveau, pour commencer :
 - Il est « open source » - gratuit et utilisable par tous.
 - Il est « portable » - utilisable sous Linux, MacOS, Windows.
 - Il a une syntaxe simple (si ! si !).
- Il a d'autres caractéristiques dont nous parlerons quand nous aborderons les thèmes qui y correspondent :
 - Il gère lui-même ses ressources mémoire.
 - Il est multi-paradigmes : impératif, fonctionnel, orienté objet.

Comment l'utiliser ?


On le comprend – taper du code dans un fichier texte ne suffit pas à créer un programme (même dans le cas du HTML qu'on a vu la semaine passée, il y avait un interpréteur – le navigateur).

On va utiliser ce qu'on appelle un IDE (« *integrated development environment* ») – il en existe littéralement des dizaines. On peut citer, pour les plus proches de nous :

- IDLE – qui est installé sur les machines NSI et que l'on va utiliser pour découvrir la console ;
- Basthon – outil en ligne sur lequel s'appuie l'application Capytale que nous allons utiliser bientôt également.
- EduPython – qui est installé sur les machines qui vous ont été fournies par la Région Ile-de-France ;
- Google Decal – outil en ligne que l'on utilisera sans doute plus tard dans l'année ;

Initiation – activités dans la console

Opérations de base

- Lancer IDLE ()
- Testez dans la console les opérations usuelles : (+ - * /) :

```
Python 3.6.1 Shell
Type "help", "copyright", "credits" or "license()" for more information
>>> 3+2
5
>>> 9/5
1.8
>>> |
```

- Testez les opérations : // % ** Devinez-vous ce qu'elles font ?

Entrées – Sorties / Affectations

Une variable est une manière de stocker une information dans la machine en la nommant.

Faire « l'affectation » d'une variable c'est en modifier la valeur – soit en la créant si elle n'existe pas encore, soit en la changeant.

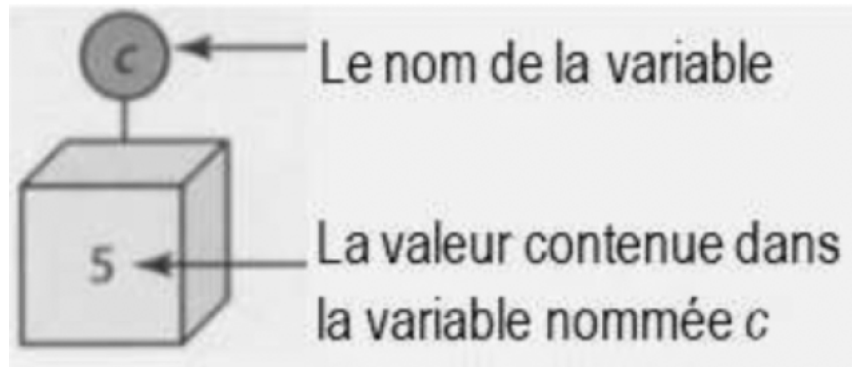
Considérons l'algorithme suivant :

1. **Saisir** a
 2. **Saisir** b
 3. $c = a + 2 \times b$
 4. $c = c + 1$
 5. **Ecrire** c
-

On a ici :

- Trois variables ;
- Deux entrées ;
- Quatre affectations (deux sur des valeurs demandées à l'utilisateur, deux calculées par l'ordinateur) ;
- Une sortie, par écriture à l'écran.

Une variable est caractérisée en premier lieu par un nom (qui permet de la localiser en mémoire) et une valeur ; elle peut être vue comme une boîte étiquetée (son nom) qui peut contenir différentes informations – dans le cas précédent, des nombres entiers. Affecter une variable, c’est modifier ce que contient la boîte.



Une variable a également un type – nombre entier, nombre réel, chaîne de caractères... – mais à l’inverse d’autres langages, il est *implicite* : il n’y a pas besoin de le déclarer.

Pour affecter une valeur à une variable en Python, on utilise le symbole =, le membre de gauche prenant la valeur de celui de droite :

```
# La ligne suivante affecte la valeur 3 à 'a'
a = 3
```

(on notera ici la syntaxe des commentaires en Python)

ATTENTION à ne pas confondre avec les maths :

- « Variable » n’a pas le même sens ;
- Le symbole « = » n’a pas le même sens.

Pour nommer une variable, il faut respecter certaines règles : caractères alphanumériques, ne pas commencer par un chiffre, aucun caractère spécial hormis l’underscore « _ » (et on évite les accents), et ne doit pas être un mot clé – int, and, break... (on y reviendra)

- Créez plusieurs variables et affectez-y différentes valeurs, cette fois de différents types (entiers, réels, texte...). (vous pouvez utiliser les exemples ci-dessous – mais inventez-en d'autres !)

```
marc = 5
Fustel = 6.578
accueil = "bonjour"
```

- Affichez la valeur de ces variables au moyen de la fonction

```
print()
```

- Demandez à l'utilisateur de renseigner certaines variables au moyen de la fonction input :

```
>>> age_prof = input("Quel age as-tu? ")
Quel age as-tu? |
```

- Vérifiez le type des variables que vous avez créé au moyen de la commande :

```
print(type(NOM_VARIABLE))
```

- On rappelle les opérations usuelles :

Instructions	Signification
$a + b$	addition
$a - b$	soustraction
$a * b$	multiplication
$a ** b$	puissance a^b
a / b	division
$a // b$	quotient de la division euclidienne de a par b
$a \% b$	reste de la division euclidienne de a par b

- Faites des tests d'opérations usuelles avec ces différentes variables – rencontrez-vous des erreurs ? Lesquelles ?

Exercices :

On considère le programme Python suivant :

```
a = 7
b = 3
c = a + b
c = c + 1
b = a - c
```

Sur une feuille copiez le tableau suivant et complétez-le :

Contenu des variables	a	b	c
Ligne 1			
Ligne 2			
Ligne 3			
Ligne 4			
Ligne 5			

➔ QCM « Affectation de variables » sur ProNote.

Scripts

Un script n'est rien de plus qu'une succession d'instructions qui sont exécutées à la suite – dans une console chaque instruction est exécutée lorsque vous passez à la ligne suivante tandis que dans un script elles le sont toutes sans que vous n'ayez rien à faire pour passer de l'une à la suivante.

Un script peut être enregistré pour être exécuté plusieurs fois ou dans plusieurs contextes – c'est, au sens où il contient une suite d'instructions mettant en œuvre un algorithme, réellement un « programme informatique ».

- Depuis votre ENT, lancez l'application « Capytale ».
- Avant de vous lancer n'oubliez pas qu'avec Python on peut convertir (quand c'est possible) des variables d'un type à un autre :

```
# Mettre un nombre au format chaîne
a = 1984
b = str(a)
c = "Roman d'Orwell: " + b
# Ou l'inverse...
a = input("Quel est votre âge? ")
b = int(a)
print(b + 10)
# Ou encore en float...
a = input("Que vaut pi avec deux decimales?")
b = float(a)
aire2 = b * 2 ** 2
```

- Rejoignez l'activité que j'ai préparée pour la séance avec le code :

```
d00f-1846631
```

Introduction aux notebooks Jupyter

➔ Les notebooks Jupyter sont des **cahiers électroniques** qui, dans le même document, peuvent rassembler du **texte**, des **images**, des **formules mathématiques** et du **code informatique exécutable**. Ils sont manipulables interactivement dans un navigateur web.

➔ Ces notebooks sont organisés en cellules qui peuvent être soit du texte (auquel cas on peut y inclure des images ou des formules également) soit du code Python (pour ce cours en tous cas).

➔ Ces notebooks sont facilement convertibles au format PDF en ligne – ce qui permet de réellement s'en servir comme d'un bloc-notes.

Ceci est une cellule de texte

On peut y mettre des titres, des sous-titres...

Des sous-sous-titres etc... Un peu comme en HTML.

C'est d'ailleurs aussi un langage balisé, mais beaucoup plus simple, appelé "markdown" - on y reviendra.

```
In [7]: 1 # Ici on met une cellule de code – dans laquelle on peut mettre des commentaires
        2 a = 3.1416
        3 # Cette cellule ne renvoie rien – elle ne fait qu'une affectation
```

```
In [8]: 1 # Celle-ci, en revanche, envoie quelque chose à la sortie
        2 print(a * 2 ** 2)

12.5664
```

Dans la cellule précédente on a utilisé la formule de l'aire d'un cercle πR^2

```
In [3]: 1 # Celle-ci afficherait quelque chose si on l'effectuait dans la console
        2 # Dans les notebooks, ceci s'affiche sous forme de "Out"
        3 5+2
```

Out[3]: 7

➔ Tout se passe comme si les cellules de code étaient dans des terminaux les uns à la suite des autres – mais entre elles je peux mettre des instructions et vous pouvez prendre des notes !

- Depuis votre ENT, lancez l'application « Capytale ».
- Rejoignez le notebook que j'ai préparé pour la séance avec le code :

ab1d-1883830

➔ A partir de maintenant nous allons régulièrement utiliser ce format pour les TPs – alors prenez-soin de bien vous y habituer !

Les fonctions Python – une brève introduction

Présentation

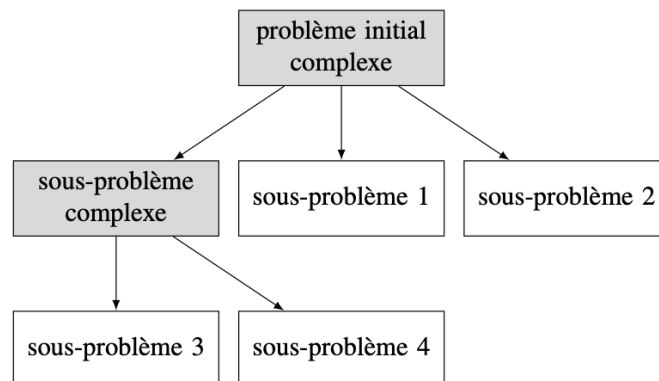
Attention pour commencer : comme pour le « = » et pour la notion de variable, le mot « fonction » n'a pas le même sens en **mathématiques** et en **informatique** : dans le premier cas c'est une **relation entre deux ensembles**, dans le second un **procédé de calcul**.

Une **fonction** :

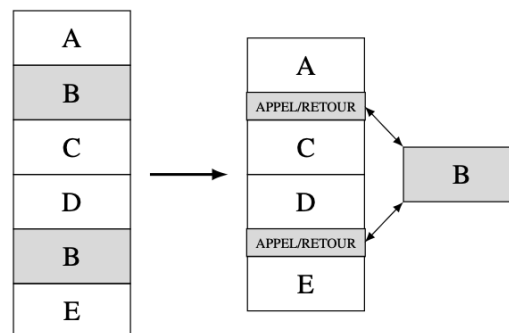
- Est définie par **un nom** ;
- Correspond à une **séquence d'instructions** réalisant **une tâche précise** ;
- Utilise aucun, un, ou plusieurs arguments. Ces arguments sont les paramètres de la fonction.
- La fonction renvoie généralement un résultat ; si elle ne le fait pas, on parle **d'une procédure**.

Les fonctions ont plusieurs utilités :

- Décomposition d'une tâche complexe :



- Réutilisation de code / éviter les répétitions :



- En conséquence : rendre le code plus court et lisible.

Syntaxe en Python

```
# Définition de la fonction
def nom_fonction(liste_paramètres):
    bloc d'instructions
    return resultat # Si fonction
```

```
# Appel d'une fonction dans le corps du programme
resultat = nom_fonction(paramètres)
```

```
# Appel d'une procédure dans le corps du programme
nom_procedure(paramètres)
```

Attention à l'indentation !!

```
def conv_minutes(heures, minutes):  
    minutes = minutes + heures * 60  
    return minutes  
  
resultat = conv_minutes(2, 30)  
  
print("2 heures 30 font ", resultat, " minutes.")
```

```
>>> # script executed  
2 heures 30 font 150 minutes.
```

Note : Lorsqu'une fonction est appelée, Python réserve pour elle (dans la mémoire de l'ordinateur) un **espace de noms**. Cet espace de noms local à la fonction est différent de l'espace de noms global où se trouvent les variables du programme principal. Dans l'espace de noms local, nous aurons des variables qui ne sont accessibles qu'au sein de la fonction – on parle alors de **variables locales**.

- Rendez-vous sur <https://pythontutor.com>
- Cliquer sur « Python ».
- Écrire le code suivant

```
def conv_minutes(heures, minutes):  
    minutes = minutes + heures * 60  
    return minutes  
  
heures = 2  
minutes = 30  
resultat = conv_minutes(heures, minutes)  
  
print("2 heures 30 font ", resultat, "  
minutes.")
```

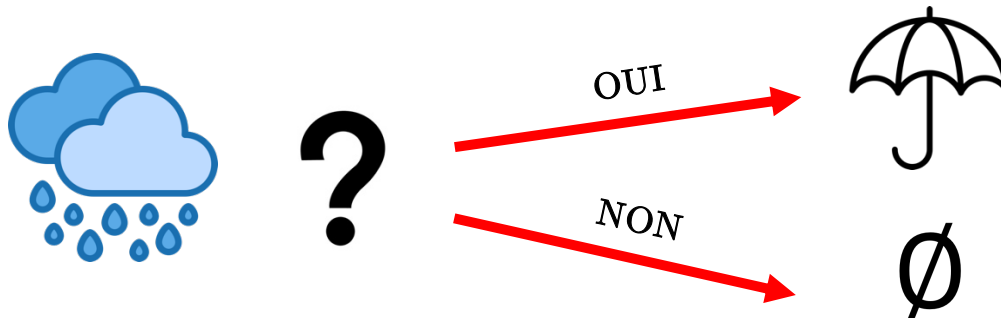
- Cliquer sur Visualize Execution et exécuter la fonction pas à pas.
- Ouvrez IDLE.
- Écrivez une fonction `perimetre_rectangle` qui prend en argument deux paramètres – une longueur et une largeur – et qui renvoie le périmètre du rectangle.
- Exécutez-la plusieurs fois et observez les sorties :

```
perimetre_rectangle  
perimetre_rectangle()  
perimetre_rectangle(3, 5)  
perimetre_rectangle(10, 2)  
perimetre_rectangle(6)
```

Conditions & Embranchements

Introduction

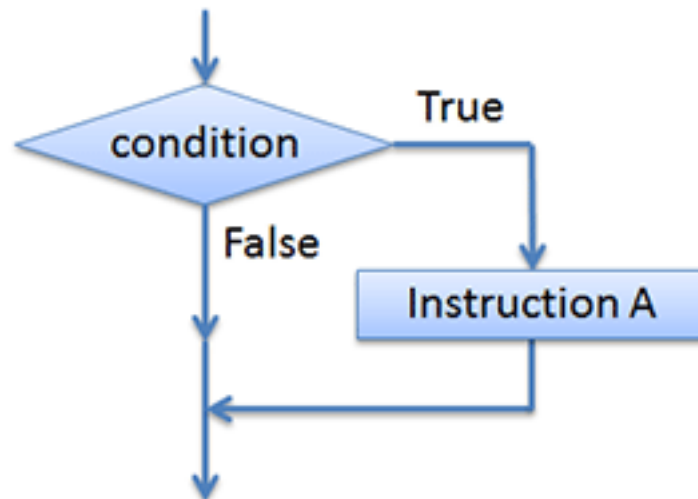
Situations classiques de la vie courante – *je dois sortir...*



On a mis en place une condition (« s'il pleut ») et une structure d'embranchement (ce que je fais dans un cas ; ce que je fais sinon).

Remarque : une structure d'embranchement, dans la vie courante comme en programmation, une fois résolue, revient à la poursuite « normale » des actions. Ici :

- S'il pleut – *alors* je prends mon parapluie, *puis* je sors.
- S'il ne pleut pas – *alors* je ne fais rien de spécial, *puis* je sors.



Conditions et tests

Une condition est :

- Un énoncé qui peut prendre deux valeurs et seulement deux valeurs : Vrai ou Faux.
- Sa résolution est donc une variable booléenne (True / False).
- Une combinaison de tests reliés par des opérateurs logiques.

Ces tests sont la plupart du temps réalisés à l'aide d'opérateurs de comparaison dont les plus fréquents sont :

==	(égal)
!=	(différent de)
<=	(inférieur ou égal)
>=	(supérieur ou égal)
<	(strictement inférieur)
>	(strictement supérieur)

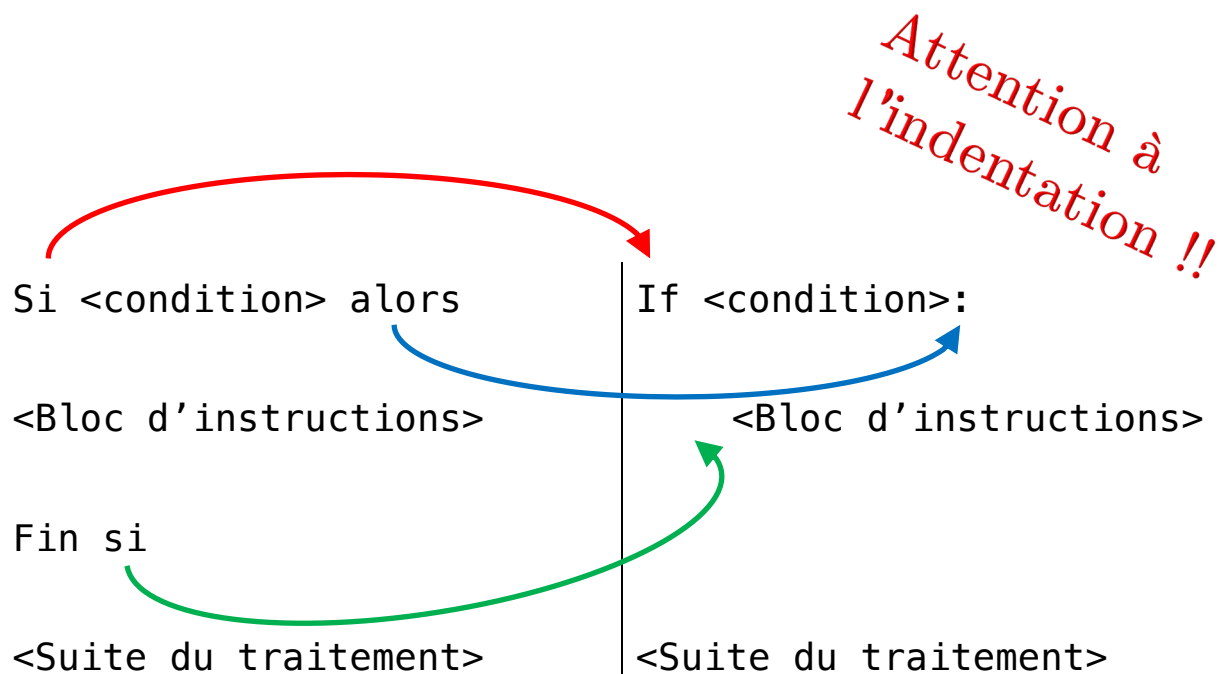
Les opérateurs logiques classiques quant à eux sont :

not	(non – négation d'une condition)
or	(ou – vrai si une de deux conditions au moins est vraie, faux sinon)
and	(et – vrai si les deux conditions sont vraies, faux sinon)


```
>>> a = 4
>>> b = 5
>>> a == 4
True
>>> a == 4 or b == 5
True
>>> a == 4 and b == 5
True
>>> a == 4 or b == 6
True
>>> a == 4 and b == 6
False
>>> not b == 5
False
```

Embranchements

La syntaxe en python pour réaliser un **embranchement** simple (tel que l'exemple de la pluie plus haut) est la suivante :



```
a = -2

if a <= 0:
    print("a négatif")
print("Fin premier test")

if a >= 0:
    print("a positif")
print("Fin second test")
```

```
>>> # script executed
a négatif
Fin premier test
Fin second test
>>>
```

Python permet également de réaliser des **embranchements** avec **alternatives** :

```
Si <condition> alors  
  
<Bloc d'instructions1>  
  
Sinon  
  
<Bloc d'instructions2>  
  
Fin si  
  
<Suite du traitement>
```

```
If <condition>:  
  
    <Bloc d'instructions1>  
  
Else:  
  
    <Bloc d'instructions2>  
  
<Suite du traitement>
```

```
a = -2  
  
if a >= 0:  
    print("a positif")  
else:  
    print("a négatif")  
print("Fin du test")
```

```
>>> # script executed  
a négatif  
Fin du test  
>>>
```

- Depuis votre ENT, lancez l'application « Capytale ».
- Rejoignez le notebook que j'ai préparé pour la séance avec le code :

5dc2-1885289

ATTENTION !!!!

Comme on l'a vu dans l'activité, les calculs algébriques et les tests avec les nombres entiers (type `int`) sont tous exacts. En revanche, **il n'en est pas de même avec les nombres flottants (type `float`)** : certains calculs sont exacts et d'autres non, de même pour les tests. Nous verrons plus loin dans le programme pourquoi il en est ainsi (c'est dû à la façon dont les nombres flottants sont représentés en binaire dans l'ordinateur). Ce qu'il faut retenir ici, c'est qu'il ne faut pas faire des tests d'égalités brutaux entre deux nombres flottants.