

Programmation

Python : Introduction

La programmation est le processus qui permet d'indiquer à un ordinateur ce qu'il doit faire.

I. Principes de programmation

a) Bref historique de la programmation

L'histoire de la programmation remonte à l'antiquité, en effet, les Grecs et les Chinois ont été les premiers à automatiser les calculs à la main ce qui constitue les prémisses de la programmation.

Puis, en 820, le savant et mathématicien Al Khawarizmi a publié à Bagdad un traité intitulé "La science de l'élimination et de la réduction" qui, importé en Europe Occidentale lors des invasions arabes a eu une grande influence sur le développement des mathématiques. Son nom est indissociable de l'informatique car chaque procédure qu'il décrit pour résoudre un calcul est un enchaînement d'instructions qui manipulent les nombres.

Une des premières machines programmables (dans le sens où les actions changent quand le programme est modifié) est le métier à tisser de Jacquard. La machine utilisait une suite de cartons perforés. Les trous indiquaient le motif que le métier suivait pour réaliser un tissage ; avec des cartes différentes le métier produisait des tissages différents.

En 1840, Ada Lovelace a défini le principe des itérations successives dans l'exécution d'une opération. En l'honneur d'Al Khawarizmi elle a (probablement) nommé "algorithme" le processus logique d'exécution d'un programme. Élève puis collaboratrice de Charles Babbage, elle a réalisé en 1843 ce qui est considéré comme le premier programme informatique sur un ancêtre de l'ordinateur, la machine analytique de Charles Babbage.

Enfin en 1936, Alan Turing imagine un modèle abstrait d'un appareil mécanique de calcul, la machine de Turing, qui est toujours largement utilisé en informatique théorique, en particulier dans les domaines de la complexité algorithmique et de la calculabilité.

b) Différents domaines de la programmation

Nous sommes entourés au quotidien de systèmes programmés, on peut distinguer 5 grands domaines :

- Programmation de sites Internet
- Programmation de jeux vidéo
- Programmation d'applications pour Smartphone
- Analyse de données
- Programmation de modules industriels

c) Instructions et registres

Définition :

Une **instruction** est l'opération élémentaire que le processeur peut accomplir. Les instructions sont stockées dans la mémoire principale, en vue d'être traitée par le processeur.

Les instructions peuvent être classées en catégories dont les principales sont :

- **accès à la mémoire** : des accès à la mémoire ou transferts de données entre registres.
- **opérations arithmétiques** : opérations telles que les additions, soustractions, divisions ou multiplications.
- **opérations logiques** : opérations ET, OU, NON, NON exclusif, etc.
- **contrôle** : contrôles de séquence, branchements conditionnels, etc.

Définition :

Lorsque le processeur exécute des instructions, les données sont temporairement stockées dans de petites mémoires rapides de 8, 16, 32 ou 64 bits que l'on appelle **registres**. Suivant le type de processeur le nombre global de registres peut varier d'une dizaine à plusieurs centaines.

d) Programme

Un programme est une séquence d'instructions pour faire quelque chose par un ordinateur. Par exemple:

- faire un calcul mathématique
- trier une liste de mots
- résoudre une équation

II. Langage de programmation

Les instructions d'un processeur sont représentées avec des séquences binaires (des 0 et des 1). Mais c'est difficile pour un humain de directement comprendre un tel langage.

Rapidement, des assembleurs et des compilateurs ont été développés, permettant de spécifier un programme avec des commandes basées sur l'anglais et donc plus facilement compréhensible pour un humain.

Un langage de programmation permet à un humain de décrire ce que doit faire un ordinateur. Tous les ordinateurs, smartphones et sites web fonctionnent grâce à des programmes

On distinguera 3 types de langage de programmation :

- **Le langage machine ou langage assembleur** : les instructions sont représentées par un mot, souvent court, qui identifie l'action à effectuer

Exemples :

load a, #3 : instruction qui charge la valeur 3 dans le registre a

add a, #1 : instruction qui additionne un nombre (2) à un registre

- **Les langages de haut-niveau :** ils permettent le développement d'un programme dans un contexte de programmation beaucoup plus convivial et qui est généralement indépendant de l'architecture matérielle de l'ordinateur (abstraction).

Exemples : C, Java, Python

- **Les langages web :** Pour des sites web, trois langues sont indispensables :
 - HTML pour le contenu de la page
 - CSS pour le style et le format de la page
 - JavaScript (JS) pour programmer l'interactivité de la page

En général, un langage de programmation comporte ces 5 composants :

- une entrée (clavier, fichier)
- une sortie (écran, fichier)
- des opérations mathématiques
- des exécutions conditionnelles (if-else)
- des répétitions (boucles)

III. Environnement de développement intégré

En programmation, un « **environnement de développement** » est un logiciel ou un ensemble d'outils qui permettent d'écrire, de développer, de tester et de déboguer des programmes informatiques. On appelle cela un **IDE** (de l'anglais **I**ntegrated **D**evelopment **E**nvironment).

Certains IDE sont dédiés à un seul langage de programmation : on ne pourra alors écrire/exécuter des programmes uniquement dans ce langage-là. D'autres IDE permettent de développer des programmes dans plusieurs langages différents.

L'objectif d'un IDE est d'augmenter la productivité des programmeurs, en automatisant une partie des activités et en simplifiant certaines opérations. Un IDE peut par exemple ajouter des couleurs aux mots/parenthèses/accolades qu'on écrit, afin de faciliter la lecture du code. Il peut également proposer l'auto-complétion (on écrit le début d'un mot, et l'IDE écrit la fin du mot automatiquement).

Dans tous les cas, un IDE contient au moins :

- une interface graphique et des icônes/boutons pour gérer les fichiers dans lesquels on a écrit les programmes (ouvrir, sauvegarder, créer un nouveau, ...) et pour lancer le programme
- un éditeur de texte ou éditeur de code pour le code source (script) : création et édition de programmes
- un **interpréteur** : c'est une zone où l'on peut entrer une instruction, et voir le résultat que cela produit. Dès que la touche ENTRÉE est pressée, l'IDE exécute l'instruction et le résultat est affiché à l'écran.
- des fonctions pour démarrer le compilateur ou l'éditeur de liens : transformation du code source en code objet

- un **débogueur** en ligne qui permet de lancer notre programme lentement, de le faire exécuter ligne par ligne, afin de chercher où se trouve une éventuelle erreur de programmation (bug). Le débogueur permet également d'afficher l'évolution des valeurs de chaque « variable », à tout moment du programme, ligne après ligne
- Une **console** : c'est une zone par laquelle l'IDE va pouvoir communiquer avec l'utilisateur. Il y affichera différents messages, par exemple signaler une erreur de syntaxe à la ligne n°5.

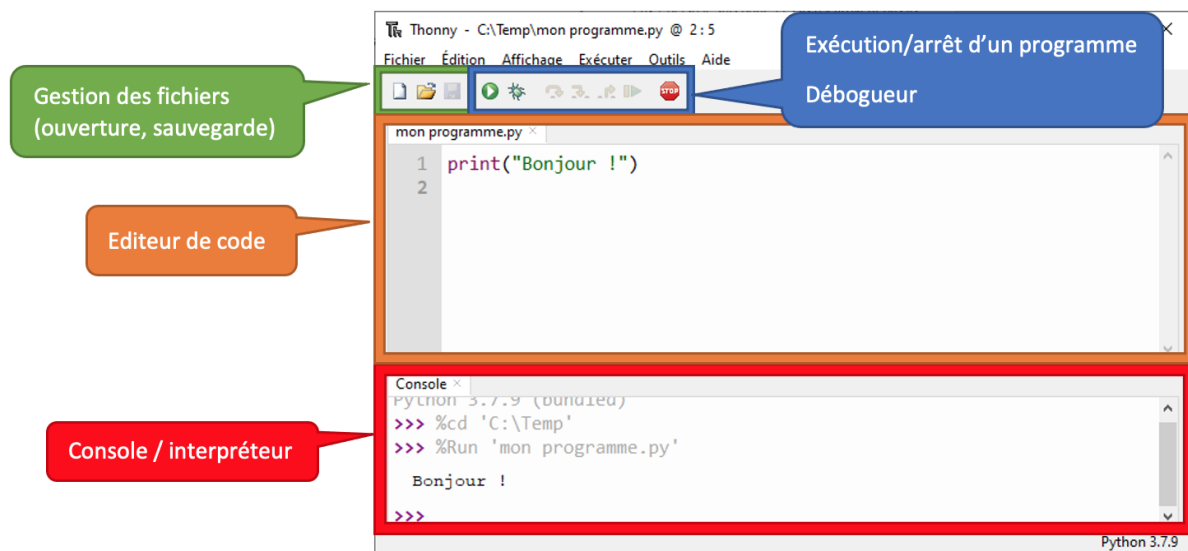


Figure 1 : L'environnement de développement intégré Thonny

On peut programmer dans différents environnements et avec n'importe quel langage de programmation.

IV. Python

Python est un langage de programmation créé en 1991 par Guido van Rossum, faisant référence au fameux groupe d'humoriste anglais, les Monty Python. Comme la plupart des langages de programmation, il est basé sur l'anglais.

Les avantages de Python :

- Python est multiplateforme (Windows, Mac, Linux)
- Python a une syntaxe simple et visuelle
- Python est interprété et donne un résultat immédiat

a) Premier programme

```
print('bonjour')
```

```
bonjour
```

Ce premier programme affichera le mot *bonjour* sur la console.

Le programme est constitué :

- d'une fonction print qui affiche son argument
- des parenthèses () qui entourent l'argument
- des guillemets '...' qui délimitent une chaîne de texte
- du texte, bonjour, qui est délimité par des guillemets

b) Calculs de base

Une des premières fonctionnalités d'un interpréteur est de faire des calculs :

- Addition

```
123 + 456
```

```
579
```

- Multiplication

```
123 * 456
```

```
56088
```

- Division

```
123 / 456
```

```
0.26973684210526316
```

La division utilise l'opérateur / et retourne un nombre à virgule flottante comme résultat.

En Python, les virgules pour les nombres doivent prendre la forme du point

- L'exponentiation

```
123 ** 456
```

```
992500687720988567008314620574696326372959408198869005198162988813828671047493990779211286614261446
```

Ceci montre que les résultats des calculs en nombres entiers en Python ne sont pas limités en termes de nombre de chiffres nécessaires à la représentation de ces entiers : le nombre ci-dessus, par exemple, s'écrit avec 953 chiffres.

- Signes particuliers

En Python, le signe % correspond au reste d'une division euclidienne (division entière) et le signe // correspond à la division euclidienne.

Si on veut faire d'autres calculs plus évolués, il faudra **importer des fonctions depuis un module**. Par exemple une fonction qui calcule la racine carrée d'un nombre, la valeur absolue, etc...

De nombreuses fonctions sont mises à disposition dans un fichier que l'on appelle module.

c) La fonction print()

La fonction **print()** affiche sur la sortie («imprime») ce qui se trouve entre les parenthèses. Cette fois, tous les résultats sont affichés.

```
print(12 + 34)
print(12 / 34)
print(12 * 34)
```

```
46
0.35294117647058826
408
```

La fonction print() peut aussi afficher du texte.

```
print('bonjour')
```

```
bonjour
```

```
print("au revoir")
```

```
au revoir
```

En programmation, un tel morceau de texte s'appelle une *chaîne de caractères*. En Python, on peut indiquer une chaîne de caractères aussi bien avec les guillemets anglais simples ('...'), aussi appelés apostrophes) que les guillemets anglais doubles ("...").

On utilise les guillemets simples quand la chaîne contient des guillemets doubles.

```
print('le langage "Python" est puissant')
```

```
le langage "Python" est puissant
```

On utilise les guillemets doubles quand la chaîne contient des guillemets simples ou des apostrophes.

```
print("c'est intéressant")
```

```
c'est intéressant
```

Que faire si une chaîne contient à la fois des apostrophes et des guillemets doubles?

Si la chaîne est délimitée par des guillemets simples, il faut utiliser le symbole d'échappement \ (la barre oblique inversée, aussi appelée *backslash*) devant une apostrophe qui apparaît à l'intérieur.

```
print('c\'est "très" cool')
```

```
c'est "très" cool
```

Si une sous-partie du texte est délimitée par des guillemets doubles, il faut utiliser le symbole d'échappement \ devant ces guillemets.

```
print("c'est \"très\" cool")
```

```
c'est "très" cool
```

Lorsque utilisé avec du texte, l'opérateur + crée une nouvelle chaîne de caractères formée à partir des deux chaînes indiquées — une opération appelée *concaténation*.

```
'bon' + 'jour'
```

```
'bonjour'
```

```
'ha' * 10
```

```
'hahahahahahahahaha'
```

```
'*' * 30
```

```
'123'
```

```
'123'
```

Une opération mathématique représentée comme chaîne n'est pas exécutée. Elle est retournée telle quelle : en effet, le caractère + est inclus à l'intérieur des guillemets et il est considéré comme faisant partie de ce texte plutôt que comme étant l'opérateur + représentant l'addition en Python.

```
'123 + 123'
```

```
'123 + 123'
```

Ces deux nombres (représentés sous forme de chaîne) sont concaténés et non pas additionnés.

'123' + '123'

'123123'

Ce nombre (représenté sous forme de chaîne) est répété et non pas multiplié.

'123' * 12

```
'123123123123123123123123123123123123123123'
```

d) Comparaison

Python connaît 6 types de comparaison :

- plus petit (<)
- plus petit ou égal (<=)
- égal (==)
- différent (!=)
- plus grand (>)
- plus grand ou égal (>=)

Le résultat d'une comparaison est une valeur qui est soit vraie, soit fausse. C'est une valeur dite *booléenne*. En Python, ces deux valeurs sont représentées avec les mots clés *True* et *False*.

Voici quelques exemples:

 $2 > 3$

False

 $2 < 3$

True

 $2 \neq 3$

On peut aussi comparer des chaînes de caractères. C'est l'ordre alphabétique qui détermine le résultat.

'antilope' < 'zèbre'

True

L'ordre des mots est celui utilisé dans les dictionnaires.

```
'art' < 'artiste'
```

```
True
```

e) Commentaires

Un commentaire commence par un dièse (#) et se termine naturellement à la fin d'une ligne. Il permet de documenter un programme et n'est pas interprété comme du code.

```
# ceci est un commentaire  
print('bonjour')
```

```
bonjour
```

Le symbole # peut être utilisé pour signaler à la machine une instruction à ne pas exécuter.

```
# print('bonjour')  
print('bonsoir')
```

```
bonsoir
```

Un commentaire peut aussi être placé après une instruction pour donner une explication supplémentaire.

```
print('bonjour') # utilisé le matin  
print('bonsoir') # utilisé le soir
```

```
bonjour  
bonsoir
```

f) Principaux types

Voici les principaux types que nous utiliserons dans cette séquence :

- **int** : nombre entier, abréviation de **integer** qui veut dire entier en anglais
Ex : 4
- **float** : nombre à virgule, la virgule peut se déplacer comme si elle flottait. D'où le nom **float** en anglais
Ex : 4.124

- **str** : chaîne de caractères. Elles sont généralement délimitées par des guillemets. Abréviation de string en anglais. Une ficelle sur laquelle on a accroché les caractères...

Ex : "Bonjour"

- **bool** : booléen qui prend les valeurs True ou False

Ex : `2 == 4` donnera False

- **list** : liste. C'est une collection d'éléments de différents types séparés par des virgules.

Ex : ["lundi", "mardi", "mercredi", "jeudi", "vendredi", "samedi", "dimanche"]

[0, 1, 2, 3, 4, 5, 6]

Remarques :

- pour la 1^{ère} liste on a mis des guillemets car c'était du texte. Pour la 2^e liste on n'en a pas mis car c'étaient des nombres.
- pour déterminer de quel type est l'élément que l'on considère, on peut interroger Python en demandant : **type()**
- on peut convertir un type de données en un autre type en utilisant la même syntaxe que si on appelait une fonction en procédant comme suit :

int() convertit en un nombre entier

float() convertit en un nombre à virgule

str() convertit en une chaîne de caractères

- nous verrons ces différents types de données petit à petit... mais pour mieux en comprendre le fonctionnement, il va nous falloir encore passer par une ou deux étapes de préparation. La première consiste à définir ce que c'est qu'une variable. La deuxième consistera à abandonner la console au profit de l'éditeur.

g) Variables et expressions

Les données qu'un ordinateur manipule se trouvent dans sa mémoire. On peut s'imaginer la mémoire comme un grand espace divisé en octets. Chaque octet a une adresse numérique, souvent exprimée avec un nombre hexadécimal.

Par exemple la valeur 26 (ou 0x1A en format hexadécimal) pourrait se trouver à l'adresse 0x1200A.

Pour ne pas avoir besoin de gérer l'adresse exacte en mémoire, Python utilise le concept de variable. Une variable est une étiquette qui permet de réserver un espace mémoire pour stocker une donnée.

Une variable permet de

- sauvegarder une valeur dans la mémoire
- lire cette valeur depuis la mémoire

- utiliser cette valeur dans une expression

On peut notamment stocker dans des variables:

- des nombres
- des chaînes de caractères
- des valeurs booléennes (True ou False)

Affectation¶

On appelle affectation l'action d'associer une valeur à une variable. La forme générale d'une affectation est `var = valeur`.

ATTENTION! '=' est l'affectation qui n'est pas pareil que '==' qui est la comparaison!

- le nom de la variable se trouve à gauche (var)
- le symbole d'affectation (=) se trouve au milieu
- la valeur à affecter à la variable se trouve à droite (valeur)

Dans l'exemple suivant nous affectons trois valeurs aux variables a, b et c. Ensuite nous pouvons afficher ces variables avec la fonction `print()`.

```
1 a = 123
2
3 print(a + 2)
4 print(a * 2)
5 print(a ** 2)
```

Par la suite, nous pouvons utiliser la variable a dans une expression arithmétique.

```
1 a = 123
2
3 print(a + 2)
4 print(a * 2)
5 print(a ** 2)
```

À n'importe quel moment, la variable peut être réaffectée, et donc changer de valeur.

```
1 a = 12
2 print(a ** 2)
3
4 a = 1234
5 print(a ** 2)
```

Affectation multiple¶

Python permet d'affecter plusieurs variables sur une même ligne.

```
1 a, b = 10, 200
2 print(a, b)
```

L'affectation multiple est une manière élégante d'échanger les valeurs de deux variables.

```
1 a, b = 10, 200
2 print(a, b)
3
4 a, b = b, a
5 print(a, b)
```

Fonction `input()`

La fonction `input()` permet de demander une valeur d'entrée (input) lorsque le programme est en cours d'exécution. Cela permet d'écrire des programmes qui utilisent des valeurs qui sont encore potentiellement inconnues lors de l'écriture du programme.

L'instruction `input('explication')` affiche le texte explication dans la console. A ce moment l'exécution du programme est interrompue et le programme attend à ce que vous entrez une valeur et confirmez avec la touche Enter.

Dès que vous appuyez sur la touche Enter, la valeur que vous avez entrée, est alors retourné par la fonction `input()`. Normalement cette valeur est affectée à une variable.

Voici un exemple de code pour vous saluer personnellement.

```
1 nom = input('Entrez votre nom (suivi par Enter): ')
2 print('Bonjour', nom)
```

Valeur et expression

Une valeur est une quantité élémentaire qui ne peut pas être évaluée vers une forme plus simple.

Voici des exemples de valeurs :

- Un nombre entier: 123
- un nombre à virgule flottante: 1.23
- une chaîne de caractères: 'hello'

Une expression est une valeur, une variable ou la combinaison de valeurs et variables avec des opérateurs mathématiques. Par exemple

- une variable: a
- une variable et une valeur lié par un opérateur: a + 4
- plusieurs valeurs liés par des opérateurs: (1 + 2) * (3 + 4)

La dernière expression contient deux sous-expressions entre parenthèses. Elles sont évaluées d'abord et donnent l'expression simplifiée 3 * 7 qui est de nouveau évaluée et qui donne 21 comme résultat final.

```
1 a = 7
2
3 print(a)
4 print(a + 4)
5 print((1 + 2) * (3 + 4))
```

Types

Les données que Python peut manipuler sont de type différents. Selon leur type, les données occupent un nombre d'octets différent dans la mémoire de l'ordinateur:

- un nombre entier (integer) occupe 4 octets
- un nombre à virgule flottante (double) occupe 8 octets

- une chaîne de caractères (string) occupe un espace mémoire variable (1 octet/caractère)

Les opérations possibles sont différentes aussi. Par exemple, nous pouvons multiplier deux nombres mais nous ne pouvons pas multiplier deux chaînes de caractères.

La fonction `type()` nous retourne le type de la valeur donnée comme argument.

```
1 print(123, type(123))
2 print(1.23, type(1.23))
3 print('123', type('123'))
```

L'expression 123 est de type `int` (integer = entier) tandis que la même expression entre guillemets '123' devient une chaîne de caractères de type `str` (string = chaîne).

Les objets 123 et '123' sont traités de façon différente dans une expression :

```
1 print(123 * 3)
2 print('123' * 3)
```

Nom d'une variable ¶

Le nom d'une variable est sensible à la casse (minuscule ou majuscule). Ainsi, `a` et `A` sont deux variables différentes.

```
1 a = 1
2 A = 99
3 print(a, A)
4 print(a + A)
```

Le nom d'une variable peut être composé de

- lettres (minuscules et majuscules)
- tiret bas `_`
- chiffres (sauf pour le premier caractère)

Sont interdits:

- les espaces
- les mots réservés par Python

<i>print</i>	<i>in</i>	<i>and</i>	<i>or</i>	<i>if</i>	<i>del</i>	<i>for</i>
<i>is</i>	<i>raise</i>	<i>assert</i>	<i>elif</i>	<i>from</i>	<i>lambda</i>	<i>return</i>
<i>break</i>	<i>else</i>	<i>global</i>	<i>not</i>	<i>try</i>	<i>class</i>	<i>except</i>
<i>while</i>	<i>continue</i>	<i>exec</i>	<i>import</i>	<i>pass</i>	<i>yield</i>	<i>def</i>
<i>finally</i>						

- tout autre caractère spécial (* + % & \$ - / ?)

Ces noms de variables sont donc valides: a2, _a, speed, pos_x, POS_X

Ceux-ci ne sont pas valides:

- 2var (commence avec un chiffre)
- if (correspond à un mot-clé)
- var\$2 (contient un caractère spécial)
- mon nom (contient une espace et est interprété comme deux noms de variables)

REMARQUE: Une bonne habitude consiste à choisir des noms en **majuscule** pour les **constantes** (variables qui ne changeront jamais après leur création) et en **minuscule** pour les variables qui sont réellement **variables** (leur valeur pourra changer après leur affectation initiale)

h) Structure de contrôle

Ce sont des structures qui permettent de contrôler l'ordonnancement des actions, c'est-à-dire dans quel ordre elles doivent être exécutées.

Il existe trois structures de contrôle différentes :

- La séquence : consiste à exécuter les actions les unes à la suite des autres. Chaque action est une action élémentaire ou composée.
- La répétition ou boucle : consiste à évaluer une condition et d'effectuer une ou des actions tant que la condition est VRAI (True)
- La sélection ou instruction conditionnelle : consiste à évaluer une condition et à effectuer des actions seulement dans certains cas. La condition est évaluée et son résultat ne peut être que VRAI (True) ou FAUX (False)

Un algorithme est une combinaison de séquences, de répétitions, de sélections, et d'actions propres de l'opérateur(actions élémentaires).

i) Indentation

On appelle bloc une ou plusieurs instructions qui forment un ensemble. En C ou JavaScript un bloc est délimité avec des accolades {...}. L'indentation est encouragée mais reste optionnelle.

En Python l'indentation est obligatoire. C'est la façon officielle de désigner un bloc. Ceci présente deux avantages :

- pas besoin d'accolades pour délimiter une bloc,
- la structure des blocs est claire et visuelle.

Une indentation est un retrait du code par rapport à la marge gauche de 4 caractères. Elle peut être insérée avec la touche tabulateur TAB (symbolisée par une flèche à gauche du clavier).

Une suite d'instructions indentée de la même manière forme un bloc. Ces blocs se trouvent dans :

- la définition de fonction (**def**)
- l'instruction conditionnelle (**if / else**)
- la boucle (**for, while**)

En Python le symbole ':' en fin de ligne introduit un sous-bloc qui doit être indenté. Voici des sous-blocs à la suite des mot-clés def, if, else, for :

```
def f(x):  
    return x * x  
  
if x > 0:  
    print('positif')  
else:  
    print('négatif')  
  
for i in range(3):  
    print('itération', i)
```

j) L'instruction conditionnelle

Une *condition* est une expression booléenne : elle peut-être évaluée à True ou à False

Une *condition* peut faire intervenir la valeur de variables, des valeurs constantes et des opérateurs relationnels

Opérateurs relationnels : > (est supérieur à), < (est inférieur à), >= (est supérieur ou égal), <= (est inférieur ou égal), != (est différent de), == (est égal à)

Exemple : Si A vaut 4, alors "A > 3" est évalué à True et "A == 2" est évalué à False

- L'instruction **if**

Elle permet d'exécuter un bloc si une condition est vraie, sinon le programme suit son cours sans exécuter ce bloc.

L'exemple suivant affiche majeur si l'âge est plus grand ou égal à 18.

```
1 age = 21
2
3 if age >= 18:
4     print('majeur')
```

- L'instruction ***if...else***

Elle permet de choisir entre deux blocs selon une condition. Si la condition est vraie, le bloc if est exécuté ; sinon, le bloc else est exécuté.

```
1 note = 4.5
2
3 if note >= 4:
4     print('réussi')
5 else:
6     print('echec')
```

- L'instruction ***if...elif...else***

Elle permet d'exécuter un bloc d'instructions si une condition est vraie. Si la première condition n'est pas remplie, une autre est testée.

```
1 n = -3
2 print('le nombre', n, 'est')
3
4 if n > 0:
5     print('positif')
6 elif n < 0:
7     print('négatif')
8 else:
9     print('zéro')
```

k) Opérations logiques

Les opérateurs logiques permettent de combiner deux valeurs logiques.

- et logique (***and***),
- ou logique (***or***).

Pour tester si un nombre x est dans l'intervalle [a, b] il faut combiner deux comparaisons avec une opération logique.

```
1 a = 5
2 b = 10
3
4 x = 8
5
6 if (a < x) and (x < b):
7     print(x, 'est entre', a, 'et', b)
8
9 if (x < a) or (b < x):
10    print(x, "est dehors l'interval (" , a, '...', b, ')')
```

L'opérateur **not** inverse la valeur logique.

- True devient False,
- False devient True.

Une double inversion revient à l'identité.

Dans l'exemple suivant, essayez de changer la valeur de p.

```
1 p = True
2
3 print('p =', p)
4 print('not p =', not p)
5 print('not not p =', not not p)
```

Série d'exercices n°1

A effectuer dans la console

Exercice 1

Trouvez une instruction qui affiche votre âge.

Exercice 2

Écrivez un programme qui affiche les phrases suivantes :

Le soleil brille à nouveau à Genève.
C'est vraiment sympa ce film !
C'est un type "chelou", disait-il.

Exercice 3

Trouvez l'expression qui affiche le résultat de 54321 multiplié par 12345.

Exercice 4

Trouvez une expression qui calcule la moyenne des 5 notes suivantes : 4.0, 2.5, 6, 5 et 4.5.

Exercice 5

Trouvez une expression qui calcule la surface d'un carré de 5 cm de côté sans utiliser $5 * 5$.

Exercice 6

Trouvez une instruction qui renvoie la longueur de la chaîne «intergouvernementalisations».

Exercice 7

Amandine dispose de 54 chocolats. Elle fabrique des paquets de 4 chocolats pour Pâques.

Combien y aura-t-il de paquets de 4 chocolats ? Combien de chocolats ne finiront pas dans un paquet ?

Exercice 8

1. Quelle est l'erreur ?

```
>>> 4.2/2
```

```
2.1
```

```
>>> 4,2/2
```

```
(4, 1.0)
```

2. Que fait le signe == ?

Exercice 9

1. Tester de quel type sont les éléments suivants : 4 ; 4.965 ; "Salut, comment vas-tu ?" ; Salut

2. Effectuer "12"+4 et int("12") + 4. Conclure

Exercice 10

1. Taper ces deux instructions.

```
1 print(len("Salut"))
```

```
1 print(len("Salut, comment vas-tu ?"))
```

Que fait la fonction len() ?

2. Taper cette instruction.

```
1 print("Sabrina Smith"[3])
```

Que peut-on faire avec [...] dans une chaîne de caractères ?

3. Taper cette instruction.

```
1 print("Sabrina Smith"[:3])
```

Que peut-on faire avec [: ...] dans une chaîne de caractères ?

4. Taper ces deux instructions

```
1 print("Sabrina Smith"[3:])
```

```
1 print("Sabrina Smith"[3:10])
```

Expliquer ce que font ces deux instructions.

5. Taper ces deux instructions

```
1 print("Sabrina Smith".upper())
```

```
1 print("Sabrina Smith".lower())
```

Que font .upper() et .lower() à la fin d'une chaîne de caractères ?

Série d'exercices n°2

Faites ces exercices dans l'éditeur de script, et non pas dans la console.

Exercice 1

1. Tester le code suivant et indiquer ce qu'il affiche

```
1 x=3
2 print(x)
3 x=x+5
4 print(x)
```

2. Déclarer et initialiser une variable nommée y à 5 et une variable nommée t à 7. Afficher le contenu des 2 variables. Ajouter à chacune des variables le contenu de y en commençant par y, puis afficher les de nouveau. Que peut-on observer ?

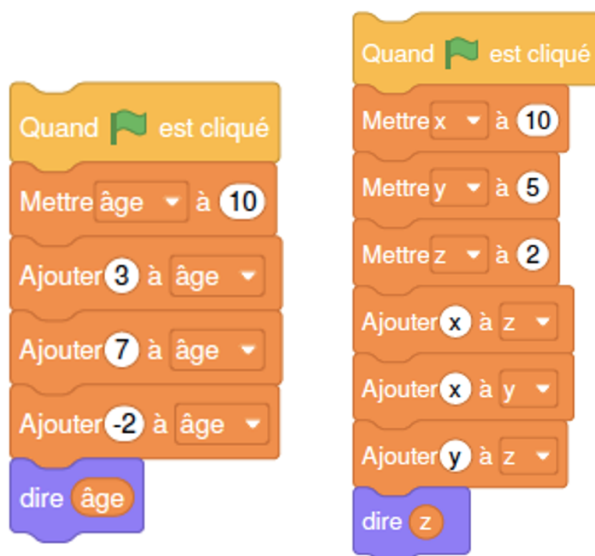
Exercice 2 - affectation

Sans utiliser l'ordinateur, déterminez ce qui va être affiché par les 5 instructions print dans ce programme.

```
a = 23
b = "hello"
c = 2
print(a + 5)
print(a - 1)
a = a + c
print(a)
c = c + a
print(c)
print(b + a)
```

Exercice 3:

Ecrire les codes Scratch suivant en code Python



Exercice 4 - input()

Faites un programme qui demande le prénom de l'utilisateur et qui affiche la phrase suivante:

```
Bonjour, _prénom_, avez-vous passé une bonne journée ?
```

Exercice 5

Définir les variables suivantes : a vaut 1, b vaut 2.3, c vaut "3" et d sera un nombre réel entré par l'utilisateur. A l'aide de la fonction print, affiche les résultats des calculs suivants :

- a+1
- a+b
- a+c
- a+d

Exercice 6 - cylindre

Faites un programme qui demande à l'utilisateur la hauteur et le diamètre d'un cylindre et retourne le volume du cylindre avec une phrase complète.

Exercice 7 - jeu

Le cadavre exquis est un jeu graphique ou d'écriture collectif inventé par les surréalistes, en particulier Jacques Prévert et Yves Tanguy, vers 1925.

Définition: *jeu qui consiste à faire composer une phrase, ou un dessin, par plusieurs personnes sans qu'aucune d'elles ne puisse tenir compte de la collaboration ou des collaborations précédentes.*

Faites un programme qui permet de faire un jeu d'écriture collectif (cadavre exquis).

- Le premier utilisateur doit proposer un sujet
- le deuxième, un verbe
- le troisième, un complément d'objet direct (COD)
- le quatrième, un complément d'objet indirect (COI)
- le dernier, un groupe permutable

Série d'exercices n°3

Exercice 1 : Intervalle

Vérifiez si une variable x contient une valeur qui est entre deux bornes [a, b].

Exercice 2 : question

Faites un programme qui pose une question simple à l'utilisateur. Si sa réponse est juste, affichez Bravo.

Exercice 3 : âge

Faites un programme qui demande à l'utilisateur son âge. Si l'âge est supérieur ou égal à 18, le programme doit afficher : «Vous êtes majeur, vous pouvez voter» et si l'âge est inférieur à 18, le programme doit afficher : «Vous êtes mineur, vous pourrez voter dans (*calcul de la différence*) année(s).»

Exercice 4 : jeu

Faites un programme qui demande à l'utilisateur un chiffre entre 0 et 2. Si l'utilisateur choisit :

- 0 : affichez Caillou,
- 1 : affichez Feuille,
- 2 : affichez Ciseaux.

Exercice 5 : nombre pair

Quand on calcule le reste dans la division par 2 d'un entier, il n'y a que deux cas : si le reste est 0, cet entier est pair et, sinon, il est impair.

Traduire cette alternative par une instruction conditionnelle dans un programme qui indiquera si le nombre est pair ou impair.

Exercice 6 : If else

Un étudiant de Claparède doit assister à des cours le mercredi après-midi à Lausanne. Il choisit comme moyen de transport le train.

- Le coût d'un aller simple en seconde classe est à 11,40 CHF.
- Le demi-tarif est au prix de 120.- pour les jeunes.

Créez un programme qui permet de définir en fonction du nombre d'aller-retour Genève-Lausanne si le demi-tarif sera ou pas le plus avantageux.

Exercice 7 : promotion

Créer un programme qui vérifie la promotion d'un étudiant de 1^{er} année à Claparède.

Pour rappel : (à corriger...)

- Il faut avoir la note minimale de 4.0 à chacune des disciplines d'enseignement suivies.
- La promotion est par tolérance si :
 - La moyenne générale est égale ou supérieure à 4.0

- La somme des écarts à 4.0 des notes insuffisantes (au maximum 3 notes) ne dépasse pas 1.0

On considère que les notes à entrer sont :

- Français
- 2^e langue (nationale)
- 3^e langue
- Mathématique
- Chimie (si OS ≠ BC)
- Economie / Droit (si OS ≠ ED)
- Art (visuel ou musique)
- OS
- Informatique
- Education Physique

Exercice 8 : Une entreprise européenne réalise des affaires avec des partenaires anglais et américains. Afin de convertir ses prix en dollars et en livres sterling, elle souhaite disposer d'un programme convertisseur. Établis ce programme. Cherche le cours du dollar et de la livre. Enregistre le programme dans un fichier nommé conversion.py

Exercice 9 : Un programme interactif avec l'utilisateur doit calculer l'altitude et la vitesse d'un objet lâché en chute libre. L'utilisateur sera invité à fournir successivement l'altitude initiale et le temps de chute après lequel il faut calculer altitude et vitesse. Dans une deuxième version du programme, ajoute une vitesse initiale verticale fournie par l'utilisateur. Le programme est enregistré sous le nom chuteLibre.py.

Exercice 10 : L'organisateur d'une course automobile sur circuit fermé d'une longueur de 4,6 km souhaite tenir les spectateurs régulièrement informés des performances réalisées par les différents concurrents. A cet effet, il pose le problème suivant : Étant donné le temps mis par un concurrent pour effectuer un tour du circuit (temps détaillé en minutes, secondes et centièmes de secondes), il souhaite connaître instantanément la vitesse (exprimée en km/h) à laquelle le concurrent a parcouru un tour du circuit. Enregistre le programme dans un fichier nommé automobile.py