1. Qu'est-ce qu'un langage de programmation

Programmer c'est écrire une suite d'instructions qui seront exécutées par une machine. Le langage machine étant composé de 0 et de 1 il serait bien trop complexe ou long d'écrire nos programmes avec des 1 ou des 0, c'est pourquoi les informaticiens ont inventé des langages un peu plus intuitifs.

On distingue 2 grands types de langage de programmation :

- Les langages de bas niveau : qui sont assez proche du langage machine et permette de gérer très précisément ce que va faire la machine. (ex : Assembleur)
- Les langages de haut niveau : qui sont beaucoup plus intuitifs mais aussi plus éloigné du langage machine et donc moins facile à optimiser. (ex : C++, Java, Python)

Une fois que l'on a écrit les instructions en langage de programmation il faut encore les traduire en langage machine, là encore nous distinguerons 2 grandes catégories :

- Les langages compilés : Un compilateur propre à chaque machine écrit les instructions machines pour cette machine spécifique (ex : C, C++, C#)
- Les langages interprétés : L'interpréteur permet d'interpréter directement les instructions par la machine, l'interpréteur est spécifique à la machine et permet d'interpréter les instructions toujours de la même façon d'une machine à l'autre. (Ex : Python)

2. Les variables en python

Contrairement au langage C, les variables en Python sont non typées. C'est-à-dire qu'il n'est pas nécessaire de préciser s'il s'agit de int, char, float, etc.

On définit soit même une variable en lui donnant un nom. On peut donner le nom que l'on veut à une variable. Mais en général il est préférable de la nommer en rapport avec son utilité.

Lorsque l'on veut utiliser plusieurs mots dans un nom on utilisera la convention snake case c'est-à-dire que chaque nouveau mot sera séparé par un tiret du bas : _

Exemple : `age` OU `mon_age` OU `age_du_capitaine`

Pour déclarer et affecter une valeur à une variable il suffit de la nommer puis faire suivre son nom du signe = puis de la valeur souhaitée.

Exemple : `age = 17`

3. Les opérations

Une fois que l'on a déclaré toutes les variables dont on a besoin, il est possible de les manipuler avec tous les outils mathématiques dont on dispose.

Il existe un certain nombre d'opérateur préenregistré en python :

- Les opérateurs de calculs : Pour les 4 opérations de bases on utilise les signes usuels : +, -, *, / *(attention ils n'auront pas le même effet s'ils sont utilisés avec du texte)*

- Il existe aussi 2 opérateurs pour les opérations suivante :
  - Division euclidienne (division entière) : //
  - Modulo (reste de la division entière) : %

- Les opérateurs logiques :

ET	and
OU	or
NON	not

- Les opérateurs de comparaison :

Supérieur	>
Supérieur ou égal	>=
Inférieur	<
Inférieur ou égal	<=
Égal	==
Différent	!=

Remarque : les opérateurs de comparaison effectuent un test, ils renvoient donc un 1 si le test est vrai et un 0 s'il est faux.

4. L'indentation

L'indentation consiste à décaler à droite les lignes pour montrer qu'elles appartiennent à une instruction. C'est comme ça que python détermine les composants du programme. **Il est donc impératif de respecter une syntaxe stricte.**

Python n'utilise pas les parenthèses ou les accolades comme la plupart des autres langages, à la place il utilise l'indentation.

5. Les conditions et boucles

Les structures de conditions sont globalement les mêmes qu'en C, avec quelques détails de syntaxe :

- La structure **if….else…**

```
if note>=10 :      #si la variable note est supérieur ou égal à 10
  moyenne = 1      #alors la variable moyenne vaut 1
  revision +=1         # on augmente la variable revision de 1

else :               #Sinon
  moyenne = 0 ;      #la variable moyenne vaut 0
```

On remarque ici que l'on emploi le symbole « : » pour montrer que ce qui suit est l'action à exécuter. Puis on indente à droite pour délimiter le bloc d'instruction.

- if elif else

En Python on remplace le else if par le mot elif :

```
if note=>15 :         #si la variable note est supérieur ou égal à 15
  felicitation = 1    #alors la variable felicitation vaut 1

elif note =>10 and note<15 : #Sinon, si note est supérieur à 10 et inférieur à 15
  encouragement = 1      # la variable encouragement vaut 1

else :                  # sinon
  felicitation, encouragement = 0  # les variables valent 0
```

N.B. : En cas de besoin on peut utiliser le mot clef « break » pour sortir immédiatement d'une boucle, quelle qu'elle soit.

- La boucle **while**

La boucle while exécute son contenu, tant que la condition est vraie.

Exemple :

```
while note<10 :  #tant que la note est inférieur à 10 alors
   travail=1           # la variable travail vaut 1
```

Attention : Lors de l'écriture d'un programme on doit toujours s'assurer qu'il y a un moyen de sortir de la boucle while, sinon le programme bouclera de façon infinie.

- La boucle **for**

La boucle for est un peu plus compliquée elle permet d'exécuter des instructions un nombre de fois précis.

Elle contient donc 3 paramètres, la valeur de départ, la valeur d'arrivée et la méthode d'incrémentation.

La plupart du temps on ne précisera pas la valeur de départ ni la méthode de comptage car par défaut, python compte à partir de 0 et de 1 en 1.

Elle s'écrit alors de cette façon : for index in range(nombre_iteration) :

- index est une variable interne à python, je n'ai donc pas besoin de la déclarer.
- in range est une fonction qui permet d'indiquer à la boucle for le nombre de boucle à faire
- nombre_iteration est remplacé par un nombre de votre choix.

Exemple :

```
for index in range(10) :  # pour index allant de 0 à 9
     if temperature<20 :   # si temperature est inférieur à 20
          temperature=temperature+1 #alors temperature augmente de 1
          print(temperature)  # affiche la temperature
```

Subtilité : Il peut arriver que l'on souhaite faire une boucle for qui ne compte pas de 1 en 1 ou qui ne démarre pas de 0 dans ce cas on précise cela dans le range. Sous la forme range(nombre de départ, nombre d'arrivée, pas de comptage)

6. Les fonctions

Lorsque l'on écrit un programme, il est fréquent de faire plusieurs fois la même chose. *(Exemple : détecter si un prénom est présent dans une liste de prénom).*

A la place de réécrire à chaque fois toutes les lignes de codes qui permettent de faire ce que l'on souhaite, on va créer des fonctions.

Une fonction est un ensemble de lignes de codes que je vais pouvoir exécuter par l'appel du nom de la fonction. C'est un sous-programme.

Syntaxe :

Une fonction est définie par un nom unique et éventuellement un ou plusieurs résultats.

Pour déclarer une fonction, on utilise le mot clef « def » suivi du nom de la fonction. On indique ensuite les paramètres éventuels entre parenthèse. On donne ensuite les instruction après une indentation à droite.

Pour récupérer un résultat, on utilise le mot clef « return » suivi du résultat. Si on a besoin de plusieurs résultats on les indique successivement.

Voici la syntaxe complète :

```
def nom_de_la_fonction(parametre_1, parametre_2) :
    instruction_1
    instruction_2
    return resultat
```

Ensuite on appelle la fonction simplement plus loin dans le code par son nom : nom_de_la_fonction(valeur_du_parametre_1, valeur_du_parametre_2)

Exemple 1 :

Declaration de la fonction :

```
def dit_bonjour(nom, age):
    phrase = f"Bonjour {nom}, vous avez {age} ans."
    print(phrase)
```

Appel de la fonction :

```
dit_bonjour(''Bastien'', 17)
```

Explication : _____

_____

Exemple 2 :

Déclaration de la fonction :

```
def puissance(nombre, exposant):
     resultat = 1
     compteur = exposant
     if nombre == 0:
          return 0
     if compteur >= 0:
          while compteur > 0:
               resultat *= nombre
               compteur -= 1
          return resultat
     if compteur < 0:
          while compteur < 0:
               resultat *= 1 / nombre
               compteur += 1
          return resultat
```

Appel de la fonction :

```
     puissance(4,8)
```

Explication : _____

_____

_____

_____

N.B. : Attention une fonction n'a pas nécessairement de paramètre ou de résultats.

Les bonnes pratiques : Lorsque l'on fait un programme on essai de faire un maximum de fonction. C'est-à-dire que dès qu'on a un bloc d'instruction qui sert à réaliser une chose définie on créera une fonction. Cela permettra de la réutiliser facilement si besoin plus tard.

7. Les bibliothèques

L'informatique, c'est avant tout apprendre à réutiliser ce qui a déjà été fait.

La syntaxe pour ajouter une bibliothèque est simple, il suffit d'ajouter au début du code :

```
import nom_de_bibliotheque
```

Dans certain cas de bibliothèque particulièrement volumineuse on peut importer seulement certaines fonctionnalités.

```
from bibliotheque import fonctionnalité
```

On peut ajouter un alias si on le souhaite avec la synthaxe

```
import bibliotheque as bib
```

Cela permet par la suite de faire référence à la bibliothèque sans utiliser son nom complet mais son alias.

8. Les tuples et listes

Python permet comme les autres langages de créer des tableaux de donnée.

Il existe 2 formats principaux.

- **Les tuples :**

Un tuple est une liste immuable, c'est-à-dire qu'il n'est pas possible de modifier son contenu après initialisation.

Exemple :

```
mon_tuple = (5, 8, 6, 9)
```

On accède aux éléments du tuple par leur indice.

Ainsi `mon_tuple[3]` correspond au 4ème élément du tuple.

On peut récupérer sa valeur dans une variable en faisant `var = mon_tuple[3]`

Par contre comme le tuple est immuable, il n'est pas possible de faire `mon_tuple[3]=5`

Attention : Comme les variables en Python son non typé il est tout à fait possible de mettre des éléments de différent type dans un tuple ou une liste :

```
mon_tuple = (a, 1, 5.6, bof)
```

- **Les Listes :**

Une liste est l'équivalent du tuple mais modifiable, au lieu de se déclarer avec des parenthèses on la déclare avec des crochets :

```
ma_liste=[5, 6, 1, 9]
```

- Pour ajouter un élément à la fin de la liste on utilise la fonction `append` :

  `ma_liste.append(15)` permet d'ajouter 15 à la fin de la liste. (Elle contient donc un élément de plus qu'avant).

- La fonction `insert` permet d'ajouter un élément à une position donnée dans la liste :

  `ma_liste.insert(0, a)` permet d'ajouter l'élément a en début de liste.

- La fonction `pop` permet de supprimer un élément par son indice :

  `ma_liste.pop(1)` supprime le 2ème élément de la liste. (sans paramètre `pop` enlève le dernier élément de la liste)

- La fonction `len` permet d'obtenir le nombre d'éléments contenu dans la liste :

  ```
  taille = len(ma_liste)
  ```

NB : Il existe de très nombreuses autres fonctions applicables sur les listes.

Particularité de la boucle for sur les listes :

On peut utilise la boucle for de deux façons sur pour parcourir une liste :

- `for index in range(len(ma_liste)):`
- `for element in ma_liste :`

Les slices :

Les slices sont une syntaxe particulière qui permet d'extraire une partie d'une liste.

Exemple : Soit une liste L = [0,1,2,3,4,5,6,7,8,9]

Alors la ligne L[2 :4] permet d'obtenir la liste : [2,3,4] , il est possible de faire de nombreuses choses avec les slices, n'hésitez pas à lire des tutos !

Liste de liste :

On peut écrire une matrice avec la syntaxe suivante :

```
ma_liste = [[0, 1, 1],
            [1, 5, 3],
            [3, 7, 5]]
```

9. Les dictionnaires

Les dictionnaires sont un des outils les plus puissants de Python cela permet de stocker des données en leur affectant 2 choses : une clé et une valeur.

Voici un exemple :

```
mon_dico = {"nom": "Durand", "prenom": "Christophe", "date de naissance": "29/02/1981"}
```

Dans ce dictionnaire chaque valeur est associée à une clé, ainsi la valeur « Durand » est associée à la clé nom.

Je peux donc appeler la valeur de la clé avec la commande :

`mon_dico["nom"]` cela me renverra Durand dans ce cas.

Ainsi il est possible de donner du sens au donnée, par exemple :

```
mes_fruits = {"poire": 3, "pomme": 4, "orange": 2}
```

Ce dictionnaire me permet aisément de savoir combien de poire j'ai avec l'instruction `mes_fruits["poire"]`

Inversement si je veux connaitre les clés d'un dictionnaire, je peux utiliser la fonction keys.

Le code suivant permet d'afficher toutes les clés du dictionnaire :

```
print("liste des fruits :")
for fruit in mes_fruits.keys():
    print(fruit)
```

La fonction del permet de supprimer une paire clé/valeur :

`del mes_fruits['pomme']` permet de supprimer la clé pomme et sa valeur associé.