

1 Python

1.1 Contexte

Python n'est pas juste un langage d'enseignement. Il est très utilisé dans plusieurs domaines. Un exemple parmi d'autres, le groupe EDF recrute des mathématiciens pour construire divers algorithmes qui sont ensuite codés en Python car simple et rapide d'apprentissage.

1.2 Syntaxe

En Python, c'est *l'indentation* qui délimite les blocs de code. Il est d'usage d'indenter par quatre espaces.

```
1 temperature = 8
2 if temperature < 5:
3     print("Mets une écharpe!")
4     print("Et un bonnet!")
```

Code 1 – Deux lignes dans le bloc

```
1 temperature = 8
2 if temperature < 5:
3     print("Mets une écharpe!")
4 print("Et un bonnet!")
```

Code 2 – Une ligne dans le bloc

Activité 1 : Écrire les codes 1 et 2 et observer les différences d'exécution.

Erreur courante : Les EDI (*Environnement de Développement Intégré*) autorise l'utilisation de la tabulation pour créer l'indentation. Cependant en copiant-collant du code externe (depuis le web par exemple), il est possible de mélanger accidentellement espaces et tabulations et de provoquer une erreur pas toujours évidente à décoder.

1.3 Usages

Les usages décrits ci-après n'ont aucun caractère obligatoire. Ils assurent cependant une cohérence dans les pratiques.

Même s'il est possible de le faire en Python (encodage UTF-8), il est conseillé de ne pas utiliser de caractère accentué dans les noms de variables.

Pour écrire le nom d'une variable, il est d'usage en Python d'utiliser le *snake_case* plutôt que le *CamelCase* (code 4).

```

1 température = 8 # possible mais non conseillé
2 temperature = 8 # préférable

```

Code 3 – Éviter les caractères accentués

```

1 ma_temperature = 8 # snake_case: pas de majuscule et tiret bas
2 MaTemperature = 8 # CamelCase: majuscule pour chaque mot

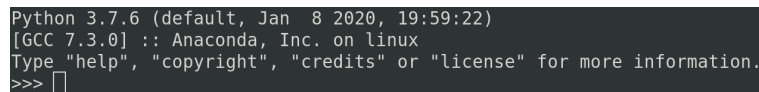
```

Code 4 – Les casses

2 Constructions élémentaires

2.1 Variable

Python est un *langage interprété* c'est à dire que les instructions sont traduites en langage machine à la volée. C'est l'*interpréteur* (figure 1) qui joue ce rôle.



```

Python 3.7.6 (default, Jan 8 2020, 19:59:22)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 

```

FIGURE 1 – Console Python

Activité 2 :

1. Dans le dossier *Maths* ou *NSI* du bureau, ouvrir une console Python : *Python 3.6.8*
2. Entrer les instructions ci -après (valider après chaque ligne) :

```

1 a = 12
2 b = 17
3 c = a + b
4 c

```

Erreur courante

Le signe `=` n'est pas à comprendre au sens mathématique. C'est un signe d'affectation. Ainsi l'instruction

```
1 12 = a
```

est juste mathématiquement mais ne signifie rien en Python.

En première approche, nous pouvons considérer une *variable* comme une boîte qui contient une information Cette information peut être de plusieurs natures (nombre, texte, tableau...)

2.2 Types de données

2.2.1 Typage

Les données utilisées dans un programme peuvent être de différentes natures. Une *chaîne de caractère* permettra l'affichage d'un message à l'écran alors qu'un *entier* sera permettra d'effectuer un

calcul.

2.2.2 Typage dynamique

Dans un programme, une variable possède un type déterminé par son contenu. La fonction Python `type()` donne le type d'une variable.

```
1 a = 42
2 type(a)
```

Ce code retourne `int` pour *integer* (*entier*).

En Python il est possible de changer le type d'une variable.

Activité 3 : Tester le code :

```
1 a = 42
2 type(a)
3 a = "test"
4 type(a)
```

2.2.3 Les types de base

- **int** : *integer* ; nombre entier (42)
- **str** : *string* ; chaîne de caractère ("test")
- **bool** : *boolean* ; valeur booléenne (True ou False)
- **float** : *flottant* ; nombre à virgule flottante (4.2)

Erreurs courantes

- La souplesse de Python qui permet de changer le type d'une variable, est perçue par certains comme une source d'erreur potentielle. D'une manière générale on évitera cette pratique.
- La représentation en mémoire des nombres flottants est source de confusion. Dans la console, tester le code :

```
1 0.1 + 0.2
```

Le résultat peut paraître déroutant. Ainsi la comparaison

```
1 0.1 + 0.2 == 0.3
```

renvoie `False`. **D'une manière générale, il ne faut pas comparer deux nombres flottants.**

Activité 4 : Tester les instructions ci-après et expliquer ce qu'elles renvoient.

```
1 20/3
2 20//3
3 20%3
```

2.3 Input/Output

2.3.1 Entrée

L'instruction

```
1 input()
```

demande une valeur à l'utilisateur. Cependant la valeur est inaccessible. Il faut donc la stocker dans une variable en mémoire :

```
1 age = input()
```

Pour être plus explicite, il est possible d'ajouter un *prompt*.

```
1 age = input("Entrez votre âge: ")
```

Erreur courante

La fonction `input()` renvoie systématiquement une chaîne de caractère même si un nombre est entré. Pour convertir la chaîne de caractère en entier (`int`) ou réel (`float`), il faut utiliser respectivement les fonction `int()` et `float()`.

```
1 age = int(input("Entrez votre âge: "))
```

2.3.2 Sortie

L'instruction

```
1 print("mon texte")
```

affiche *mon texte* à l'écran.

Il est possible d'afficher le contenu d'une variable :

```
1 print(age)
```

Il ne faut alors pas mettre de guillemets. Pour combiner texte et variable il existe plusieurs syntaxes.

```
1 # Concaténation
2 print("Vous avez"+"15 ans")
3 # La fonction format remplace les accolades par les variables.
4 print("Vous avez {} ans".format(age))
5
6 # Une autre possibilité (pour Python > 3.6). Il faut remarquer le f en d
  ébut.
7 print(f"Vous avez {age} ans")
```

Code 5 – Associer chaîne de caractère et variable

Erreur courante

Le code 6 renvoie une erreur.

```
1 age = 25
2 print("Vous avez "+ age +" ans")
```

Code 6 – Erreur de concaténation

Il n'est pas possible de concaténer un *String* avec un autre type de données. Il faut d'abord convertir la variable avec la fonction *str()*.

```
1 age = 25
2 print("Vous avez "+ str(age) +" ans")
```

Code 7 – Conversion en String

Activité 5 : Écrire un programme qui demande l'âge de l'utilisateur, calcule l'année de naissance et affiche cette année.

2.4 Utilisation d'un EDI

Il peut rapidement être fastidieux d'écrire un programme dans la console Python. Un *Environnement de Développement Intégré* permettra d'écrire plusieurs lignes de code puis se chargera d'envoyer toutes ces lignes à l'interpréteur. De plus, il sera possible d'enregistrer le programme dans un fichier.

Activité 6 :

1. Créer un dossier *formation-python* dans l'espace personnel de l'ordinateur.
2. Ouvrir un EDI au choix : Spyder, Pyzo, EduPython. Les descriptions ci-après se feront sur Spyder.
3. Écrire le programme de l'activité 4 dans la partie gauche de l'EDI.
4. Enregistrer le programme dans le dossier *formation-python* sous le nom *naissance.py*
5. Exécuter le programme en cliquant sur la flèche verte (figure 2) ou en appuyant sur la touche *F5*. Le code est exécuté dans la console en bas à droite.



FIGURE 2 – Exécuter un programme

2.5 Structure conditionnelle

L'instruction

```
1 a == b
```

renvoie *True* si les variables *a* et *b* sont égales, *False* sinon.

L'instruction

```
1 if a == b:
2     print(a)
```

compare *a* et *b* et affiche *a* si *a* == *b*.

Erreur courante

Il faut noter l'emploi du *double égal* pour ne pas confondre avec le signe d'affectation.

Activité 7 :

1. Tester les codes ci-après :

```
1 a = 5
2 b = 3
3 if a == b:
4     print("a vaut ",a)
5     print("b vaut ",b)
```

```
1 a = 5
2 b = 3
3 if a == b:
4     print("a vaut ",a)
5 print("b vaut ",b)
```

2. Noter les différences d'exécution.
3. Tester le code ci-après pour plusieurs valeurs de *a* et *b*. Bien observer l'indentation.

```
1 a = 5
2 b = 3
3 if a == b:
4     print("a et b sont égaux.")
5 else:
6     print("a et b sont différents.")
```

4. Il est possible de tester plusieurs conditions. Trouver des valeurs de *a* et *b* pour lesquelles le message « *a* est vraiment très grand. » est affiché.

```
1 a = 5
2 b = 3
3 if a == b:
4     print("a et b sont égaux.")
5 elif a > 10*b:
6     print("a est vraiment très grand.")
7 else:
8     print("a et b sont différents.")
```

5. Enregistrer le programme sous le nom *condition.py*

2.6 Répéter une instruction

2.6.1 Boucle non bornée

Une boucle *non bornée* répète une instruction *tant que* (*while*) la condition est vérifiée.

Activité 8 :

1. Tester le programme ci-après :

```
1  compteur = 10
2  while compteur > 0:
3      print("Boum dans {} secondes.".format(compteur))
4      compteur = compteur - 1
5  print("Boum")
```

2. À quelle ligne compare-t-on le compteur avec la valeur limite ?
3. Quel est le rôle de la ligne 4 ? Que se passera-t-il si cette ligne est retirée ?
4. Enregistrer le programme sous le nom *boucle-non-bornee.py*

Erreur courante

Dans une boucle *while* il faut gérer manuellement le cas où la condition ne sera plus vérifiée. Il arrive régulièrement d'oublier de modifier la valeur du *compteur*. La boucle tourne alors indéfiniment.

2.6.2 Boucle bornée

Il existe une autre manière de répéter des instructions avec un mécanisme qui varie le compteur automatiquement.

Activité 9 :

1. Tester le programme ci-après :

```
1  for compteur in range(10):
2      print("Le compteur vaut {}".format(compteur))
```

2. Lire la documentation de la fonction *range* :

<https://docs.python.org/fr/3/tutorial/controlflow.html#the-range-function>

3. Adapter le code précédent pour afficher :
 - Le compteur vaut 6.
 - Le compteur vaut 7.
 - Le compteur vaut 8.
 - Le compteur vaut 9.
4. Adapter le code précédent pour afficher :
 - Le compteur vaut 0.
 - Le compteur vaut 3.
 - Le compteur vaut 6.
 - Le compteur vaut 9.
5. Enregistrer le programme sous le nom *boucle-bornee.py*

Erreur courante

L'appel `range(10)` renvoie 10 valeurs **en commençant par 0**. La borne supérieure (10) n'est donc pas dans l'intervalle.

2.7 Types construits de données

Pour stocker plusieurs valeurs, il existe plusieurs types de structures.

2.7.1 Tuple

Un *tuple* est une séquence **ordonnée** de plusieurs éléments. En mathématiques on parle de *p-uplet*.

```
1 mon_tuple = (8, 5, 3, 9, 1, 0, 2)
```

Code 8 – Créer un tuple en Python

On ne peut modifier un tuple : on dit qu'il est *immuable* ou *non mutable*. Par contre il est possible d'accéder aux éléments individuellement.

```
1 print(mon_tuple[0]) # renvoie 8
```

Code 9 – Accéder à l'élément de rang 0

Le code 9 renvoie la première valeur du tuple. **L'indexation commence à 0.**

2.7.2 Tableaux

Un deuxième type de structure semble plus adapté lorsqu'on veut stocker des valeurs de manière séquentielle : *les tableaux*. En Python on parle de *list* pour évoquer ces structures.

```
1 mon_tableau = [8, 5, 3, 9, 1, 0, 2]
```

Code 10 – Créer un tableau

Pour accéder à un élément, la syntaxe est la même que pour les tuples.

```
1 print(mon_tableau[0]) # renvoie 8
```

Code 11 – Accéder au premier élément

Les tableaux sont *mutables*. Il est possible de modifier leur contenu, ajouter voire supprimer un élément.

```
1 mon_tableau[2] = 19
```



```
1 mon_tableau.append(12)
```

Code 13 – Ajouter un élément en fin de liste

Code 12 – Modification du troisième élément

Il existe de nombreuses autres méthodes pour manipuler les tableaux. La documentation Python présente ces outils.

<https://docs.python.org/fr/3/tutorial/datastructures.html>

2.7.3 Dictionnaires

Un dictionnaire associe une *clé* à une *valeur*. Il n'y a pas de notion d'indice comme pour les autres types construits.

```
1 dico_identite = {"nom": "Viroulaud", "prenom": "Christophe", "age": 43}
```

Code 14 – Construction d'un dictionnaire

```
1 print(dico_identite["prenom"]) # renvoie Christophe
```

Code 15 – Accéder à la valeur associée à la clé *prenom*

```
1 dico_identite["age"] = 25
```

Code 16 – Modifier une valeur

2.7.4 Parcourir un type construit de données

Les types construits sont *itérables*. Il est donc possible de *boucler* sur leurs éléments.

Activité 10 : Tester les codes ci-après :

```
1 mon_tableau = [8, 5, 3, 9, 1, 0, 2]
2 # La fonction len() renvoie la taille de la structure
3 for i in range(len(mon_tableau)):
4     print(mon_tableau[i])
```

Code 17 – Boucler en utilisation les indices

```
1 for element in mon_tableau:  
2     print(element)
```

Code 18 – Boucler directement sur les éléments du tableau

```
1 for cle, valeur in dico_identite.items():  
2     print(cle, valeur)
```

Code 19 – Boucler sur un dictionnaire

2.8 Fonction

2.8.1 Définition

Il faut rapprocher une fonction informatique de la notion de fonction mathématique : il s'agit d'une *boîte noire* qui possède des *paramètres* et qui nous renvoie un résultat (avec le mot-clef *return*).

```
1 def fonction_cube(x):  
2     return x**3
```

Code 20 – légende

Une fois la fonction créée il faut l'appeler dans le programme principal en lui passant des *arguments*.

```
1 print(fonction_cube(5)) # renvoie 125
```

Code 21 – L'argument 5 est passé à la fonction

Erreur courante

Il est tentant d'afficher le résultat directement depuis la fonction (code 22).

```
1 def est_pair(x):  
2     # si le reste de la division est nul  
3     if x%2 == 0:  
4         print(f"{x} est pair.")  
5     else:  
6         print(f"{x} est impair.")  
7  
8 # programme principal  
9 est_pair(5)
```

Code 22 – Mauvaise pratique

Ce code ne lève pas d'erreur mais il est préférable de séparer calcul et affichage. En mathématique la fonction s'occupe seulement de calculer l'image et non de placer le point correspondant dans un repère.

```
1 def est_pair(x):
2     """
3     Renvoie True si x est pair
4     """
5     if x%2 == 0:
6         return True
7     else:
8         return False
9
10 # programme principal
11 x = 5
12 if est_pair(x):
13     print(f"{x} est pair.")
14 else:
15     print(f"{x} est impair.")
```

Code 23 – L'affichage est séparé du calcul

2.8.2 Portée d'une variable

En première approche il faut considérer qu'une variable n'est visible que dans la fonction où elle est définie. À ce titre, considérons alors la programme principal comme une fonction.

Activité 11 : Tester le code suivant.

```
1 def affine(x):
2     a = 5
3     b = 3
4     return a*x+b
5
6 print(affine(10))
7 print(a)
```

Code 24 – Variable locale

a est une *variable locale*. Elle n'est visible que dans la fonction *affine*.

Erreur courante

- En réalité, dans une fonction il est possible d'accéder à une variable du programme principal (*variable globale*). C'est une mauvaise pratique. Une fonction doit être indépendante, afin par exemple d'être réutilisable dans un autre contexte. D'une manière générale nous éviterons

au maximum d'utiliser des variables globales.

- Sans rentrer dans le détail des représentations en mémoire, il faut tout de même faire la distinction entre les variables mutables ou non.

Activité 12 : Tester les codes ci-après sur le site <http://pythontutor.com/>

```
1 def modifier():
2     a = 5
3
4 a = 8
5 modifier()
6 print(a)
```

Code 25 – Variable immuable

```
1 def modifier():
2     tab[0] = 12
3
4 tab = [1, 4, 8]
5 modifier()
6 print(tab)
```

Code 26 – Variable mutable

3 Bibliothèque

3.1 Import

3.2 Bibliothèque personnelle

4 Manipulation de données

5 Représentation graphique

6 Jupyter