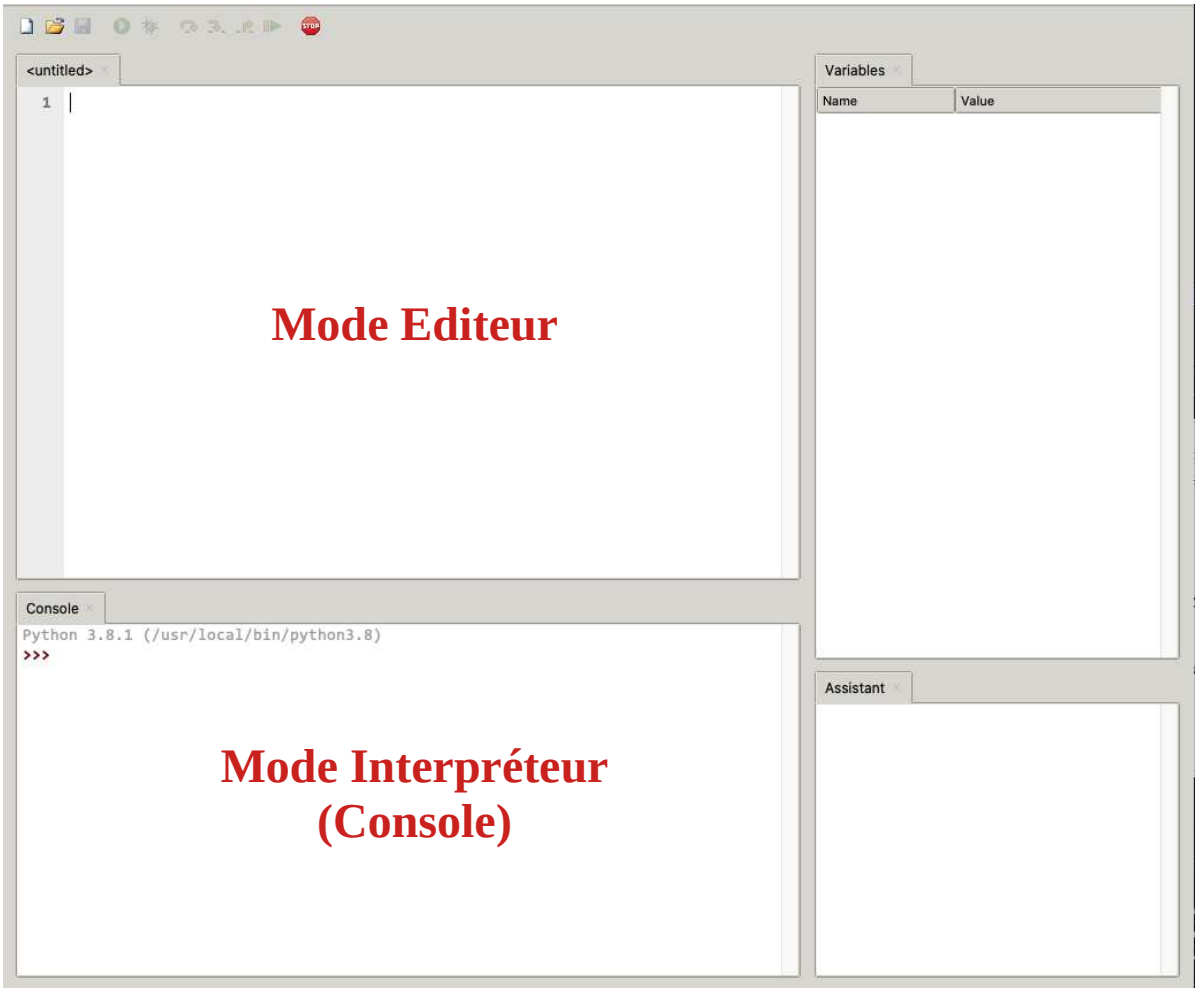
FICHE 1 - INTRODUCTION A PYTHON

1 -Programmer en Python

Il existe plusieurs « distributions » de Python. Nous allons utiliser *Thonny* qu'il va falloir télécharger à l'adresse www.thonny.org.

2 - Deux modes d'utilisation

Il existe deux modalités d'utilisation de Python : **le mode interpréteur** (ou console) et **le mode éditeur** pour écrire des scripts.



3 - Mode interpréteur (ou console Python)

Le symbole >>> vous invite à entrer des instructions ou des calculs.

Entrez les instructions suivantes (sans les >>>), en validant à chaque fois et en observant les réponses de Python :

```
>>> 5
>>> 7 - 3
>>> 1/3
>>> 17//3    # Partie Entière d'une division
>>> 17%3    # Reste d'une division entière
>>> 5**3
>>> 3.14 + 1.05
```

Remarques : *Python comprend les mathématiques ;*

- *La virgule décimale est ici remplacée par un point (à l'anglo-saxonne) ;*
- *Python ne fait que du calcul arrondi avec parfois des résultats surprenants… ;*
- *La puissance se note ***

4 – Notion de variable

Entrez les instructions suivantes :

```
>>> valeur1 = 5.3
>>> valeur2 = 2.1
>>> valeur1
>>> valeur1*valeur2
```

On a utilisé des variables (nommées ici « `valeur1` » et « `valeur2` ») pour stocker des valeurs.

Parlons maintenant de texte à afficher, écrire:

```
>>> bonjour
```

Le nom « bonjour » est vu comme celui d'une variable non définie, l'interpréteur ne comprend pas.

Tapez maintenant :

```
>>> print("bonjour")
>>> print(valeur1)
```

Le mot `print` désigne une fonction de Python, qui sert à afficher un texte ou le contenu d'une variable. Les guillemets (") permettent d'identifier qu'il s'agit d'un texte.

Pour bien comprendre la différence entre du texte et des noms de variables, tapez ceci :

```
>>> a = "bonjour"
>>> print("a")
>>> print(a)
>>> b = "tout le monde"
>>> c = a+b
>>> print(c)
```

Ce type de variable s'appelle **une chaîne de caractères** (on parlera aussi de texte). Si on ajoute deux chaînes, on obtient une nouvelle chaîne.

Tapez maintenant ceci :

```
>>> a = 3
>>> print(2*"a")
>>> print(2*a)
```

Attention : ne pas confondre le texte « `a` » et la variable nommée `a`.

On peut mélanger du texte et des nombres :

```
>>> print("1+1 font ", 1+1)
>>> a = 7
>>> print("Le carré de", a, "est", a**2)
```

La virgule sert à séparer les éléments à afficher.

5 – Mode éditeur

L'interpréteur est pratique pour tester des instructions. Le problème est que quand vous quittez l'interpréteur, toutes vos instructions sont perdues.

Dans le mode éditeur, on va pouvoir écrire des programmes longs puis ensuite les exécuter. Le résultat s'affichera dans la console.

Pensez à sauvegarder vos programmes de sorte qu'il ne soient pas perdus quand vous quittez Thonny.

Il y a des différences de fonctionnement avec le mode interpréteur. Par exemple, tapez dans l'éditeur : `3+5` puis lancez l'exécution avec le bouton **`run`**  .

Contrairement au mode interpréteur, le calcul `3+5` se fait **mais** rien ne s'affiche. Par contre, `print(3+5)` donnera le résultat attendu.

L'instruction `input()`

Quand l'utilisateur du programme doit saisir une valeur on utilise l'instruction `input()`.

Dans les parenthèses, on écrit le message qui s'affiche. Par exemple, tapez maintenant ceci :

```
a = input("Saisir a : ")
print("a vaut", a)
```

En fait cet algorithme cache un piège puisque la variable saisie n'est pas considérée comme un nombre par Python mais comme une chaîne de caractères.

On peut voir cela en modifiant le script :

```
a = input("Saisir a : ")
print("Le double de a vaut", 2*a)
```

Ce script ne donne pas le résultat espéré !

Pour résoudre le problème, on convertit la chaîne de caractères saisie en nombre entier ou à virgule (on dit **flottant)**.

a est un entier

```
a = int(input("Saisir a : "))
print("Le double de a vaut", 2*a)
```

a est un nombre à virgule

```
a = float(input("Saisir a : "))
print("Le double de a vaut", 2*a)
```

FICHE 2 - LES TESTS

1 - Les tests avec Python

On va parler ici des tests (ou « structures conditionnelles ») qui permettent à un programme de s'adapter à plusieurs situations. L'instruction clé pour les tests est **`if`** (en français : **`Si`**).

2 – Inégalité

Lancez Thonny. Dans le mode interpréteur (la console), tapez ceci et voyez les réponses de Python :

```
>>> 5 > 3
>>> 4 < 4
>>> 6 >= 6
>>> a = -4
>>> a > 0
```

Explications : Python vérifie chacune des inégalités. Quand elle est vraie, il répond **`True`**, quand elle est fausse, il répond **`False`**.

3 – Egalité

Tapez maintenant ceci et observez bien les réponses de Python :

```
>>> 5 == 3
>>> 3 == 3
>>> b = 6
>>> b == 10
>>> b == 6
```

L'instruction `5 == 3` veut dire regarder si le nombre `3` est égal au nombre `5`, Python répond **`False`** ; On met enfin `6` dans la variable « `b` » puis on demande à Python de regarder si `b` est égale à `10` (ce qui est faux) puis si `b` est égale à `6`, ce qui est vrai.

Conclusion : le `==` sert à vérifier si deux choses sont égales.

Attention : ne confondez pas le `=` (pour les affectations) et le `==` (pour les tests).

4 - Non égalité

Enfin, tapez ceci :

```
>>> test = -5
>>> test == 2
>>> test != 2
```

Explications : on donne la valeur `-5` à la variable « `test` » ;

on demande à Python si la valeur de « `test` » est égale à `2`, c'est faux ;

on demande à Python si la valeur de « `test` » est différente de `2`, c'est vrai ;

le symbole **`!=`** veut dire « non égal à » (ce qu'en mathématiques, on écrirait ≠).

5 - Le test « Si … alors ... »

Dans le mode éditeur, entrez ce programme :

```
temp = float(input("Quelle température fait-il ?"))
if temp <= 0:
    print("L'eau gèle")
```

Testez-le avec plusieurs valeurs. Modifiez-le exactement comme ci-dessous puis testez de nouveau :

```
temp = float(input("Quelle température fait-il ?"))
if temp <= 0:
    print("L'eau gèle")
print("Attention au verglas")
```

Le programme ne fonctionne pas bien : il affiche toujours « Attention au verglas » quelle que soit la température…

Cela est dû au fait que l'instruction `print("Attention au verglas")` n'a pas été décalée vers la droite comme l'instruction `print("L'eau gèle")`. Elle est alors extérieure au test et s'exécute toujours. Si on veut qu'elle ne s'exécute que si `temp<0`, on fait ainsi :

```
temp = float(input("Quelle température fait-il ?"))
if temp <= 0:
    print("L'eau gèle")
    print("Attention au verglas")
```

Ce décalage vers la droite avec une tabulation s'appelle une **indentation**. En Python, il est primordial de respecter l'indentation.

6 - Le test « Si … alors … sinon ... »

On a parfois besoin de traiter le cas où la condition n'est pas vérifiée. On utilise alors `if … else …` (`else` veut dire sinon). Par exemple :

```
temp = float(input("Quelle température fait-il ?"))
if temp <= 0:
    print("L'eau gèle")
    print("Attention au verglas")
else:
    print("Aucun risque de verglas")
```

Signalons enfin une troisième forme, utilisant `if … elif … else …` (`elif` est la contraction de `else if`).

Par exemple :

```
temp = float(input("Quelle température fait-il ?"))
print("L'eau est sous la forme de : ")
if temp <= 0:
    print("glace")
elif temp > 100:
    print("vapeur")
else :
    print("liquide")
```

7 – Tests imbriqués

Il est parfois utile d'imbriquer plusieurs tests. L'exemple précédent peut ainsi s'écrire :

```
temp = float(input("Quelle température fait-il ?"))
print("L'eau est sous la forme de : ", end="")
if temp <= 0:
    print("glace")
else:
    if temp >= 100:
        print("vapeur")
    else:
        print("liquide")
```

Attention à l'indentation dans ces cas-là !

FICHE 3 - LES BOUCLES

Les boucles sont indispensables quand on veut répéter plusieurs fois des instructions.

1 - Les boucles `Pour`

En général, quand on sait combien de fois doit avoir lieu la répétition, on utilise une boucle **`for`**.

```
for i in range(10):
    print(i)
```

Pour … allant de … à …

Attention, observez-bien quelles sont les bornes du compteur !

```
for i in range(3, 10):
    print(i)
```

La fonction `range(a,b)` renvoie la séquence de nombres entiers a, a+ 1, a+2, …, b-2, b-1.

```
for v in range(3, 9, 2):
    print(v)
```

```
for v in range(13, 2, -3):
    print(v)
```

Avec la fonction `range(a, b, k)`, `k` est utilisé comme valeur de pas : Le premier élément de la séquence est `a`. puis `a+k`, `a+2k`, ... `b` est la limite. Le dernier élément de la séquence doit être inférieur strictement à `b`.

2 - Les boucles `Tant Que`

La syntaxe de la boucle « `Tant Que` » est **`while`**:

```
while condition:
    instruction_dans_boucle1
    instruction_dans_boucle2
    …
instruction_hors_boucle1
instruction_hors_boucle2
…
```

Tant que la condition est vérifiée, on effectue les instructions 1, 2 et 3 (etc., celles qui ont été indentées).

Exemple 1 : la machine doit afficher le texte « `Bonjour` » un certains nombre de fois (souvent indéterminé).

```
i=int(input("Saisir i inférieur à 50"))
while i <= 50:
    i = i+1
    print("Bonjour")
```

Un problème apparaît parfois : la boucle ne s'arrête jamais (on parle de boucle infinie)

```
i = 1
while i > 0:
    print("Bonjour")
    i = i+1
```

Il faut donc faire attention que la condition de boucle finisse par être réalisée.

3 – Comparaison de boucles

Exemple 2 : la machine doit afficher les carrés de tous les entiers de 1 à 100.

Avec la boucle « Tant Que » :

```
i = 1
while i <= 100:
    print("Le carré de", i, "est", i**2)
    i+=1
```

Remarque : *i+=1 est identique à i=i+1*

Avec la boucle « Pour » :

```
for i in range(1, 101):
    print("Le carré de", i, "est", i**2)
```

La boucle « Pour » est plus compacte, mais la boucle « Tant Que » est plus universelle car on n'a pas besoin de savoir au départ combien de répétitions la machine va faire.

```
i=int(input("Saisir i inférieur à 100"))
while i <= 100:
    print("Le carré de", i, "est", i**2)
    i+=1
```

FICHE 4 - LES FONCTIONS

1 – Les fonctions

Lorsqu'une tâche doit être réalisée plusieurs fois par un programme avec seulement des paramètres différents, on peut l'isoler au sein d'une fonction. La syntaxe Python pour la définition d'une fonction est la suivante :

```
def nom_fonction(liste de paramètres):
    bloc d'instructions
    ...
    return(résultat)
```

Remarque : Vous pouvez choisir n'importe quel nom pour la fonction que vous créez à la condition de n'utiliser aucun caractère spécial (le caractère souligné _ est permis). Comme c'est le cas pour les noms de variables, on utilise par convention des minuscules, notamment au début du nom.

2 – Fonction sans paramètre et sans sortie

On peut définir une fonction dont l'objectif est d'écrire « `Bonjour !!!` »

```
def bonjour():
    print("Bonjour !!!")

bonjour()
```

Remarques : *On appelle la fonction par : `bonjour()`*

3 – Fonction avec un paramètre et sans sortie

```
def bonjour(nom):
    print("Bienvenue à toi", nom)

bonjour("Marcel")
bonjour("Paul")
```

Remarques : *On appelle la fonction par : bonjour("Marcel")*

4 – Fonction avec un ou plusieurs paramètres et avec une ou plusieurs sorties

Un paramètre et une sortie :

```
def plus_1(a):
    return a+1

resultat=plus_1(12)
print(plus_1(12))
```

La fonction `plus_1()` retourne une valeur.

- Soit on stocke cette valeur (ici dans `resultat`)
- Soit on affiche directement ce qui est renvoyé dans un `print()`

Deux paramètres et une sortie :

```
def somme(a, b):
    return(a+b)

print(somme(12, 14))
```

Deux paramètres et deux sorties :

```
def somme_produit(a, b):
    return(a+b, a*b)

print(somme_produit(12, 14))
```

FICHE 5 - LES LISTES

Une liste en python, est une variable dans laquelle on peut mettre plusieurs variables. Ce sont des séquences, des collections ordonnées d'objets séparés par des virgules.

1 - Créer une liste en python : Pour créer une liste, rien de plus simple :

```
>>> liste = []                # liste vide
>>> liste = [1, 2, 3]         # liste pré-remplie
```

2 - Ajouter une valeur à une liste python : On peut ajouter les valeurs après la création de la liste avec la méthode **append** (qui signifie "ajouter" en anglais)

```
>>> liste = []
>>> liste
[]
>>> liste.append(1)
>>> liste
[1]
>>> liste.append("ok")
>>> liste
[1, 'ok']
```

On voit qu'il est possible de mélanger dans une même liste des variables de type différent.

3 - Afficher un item d'une liste : Pour afficher un seul élément de la liste, on précisera son index (sa position) :

```
>>> liste = ["J", "B", "D"]
>>> liste[0]
'J'
>>> liste[2]
'D'
```

Le premier item commence toujours avec l'index (la position) 0.

Il est donc possible de modifier une valeur avec son index :

```
>>> liste = ["a", "d", "m"]
>>> liste[2] = "z"
>>> liste
['a', 'd', 'z']
```

On peut aussi afficher un item à partir de sa position de fin !

Afficher le dernier item d'une liste :

```
>>> liste[-1]
"m"
```

Afficher le 3ème élément en partant de la fin :

```
>>> liste[-3]
"a"
```

4 - Compter le nombre d'items d'une liste : Il est possible de compter le nombre d'items d'une liste avec la fonction **`len`** :

```
>>> liste = [1, 2, 3, 5, 10]
>>> len(liste)
5
```

5 - Trouver un item dans une liste : Pour savoir si un élément est dans une liste, on peut utiliser le mot clé in de cette manière :

```
>>> liste = [1, 2, 3, 5, 10]
>>> 3 in liste
True
>>> 11 in liste
False
```

6 - Boucler sur une liste : Pour afficher les valeurs d'une liste, on peut utiliser une boucle :

```
>>> liste = ["a", "d", "m"]
>>> for lettre in liste:
...     print(lettre)
a
d
m
```

7 - Manipuler une liste :

Supprimer un item par son index : On utilise la méthode **`del`** :

```
>>> liste = ["a", "b", "c"]
>>> del liste[1]
>>> liste
['a', 'c']
```

Supprimer un item par sa valeur : On utilise la méthode **`remove`** :

```
>>> liste = ["a", "b", "c"]
>>> liste.remove("a")
>>> liste
['b', 'c']
```

Compter le nombre d'occurrences d'une valeur : On utilise la méthode **`count`** :

```
>>> liste = ["a", "a", "a", "b", "c", "c"]
>>> liste.count("a")
3
>>> liste.count("c")
2
```

Trouver l'index d'un item: On utilise la méthode **`index`** :

```
>>> liste = ["a", "a", "a", "b", "c", "c"]
>>> liste.index("b")
3
```

Trouver le maximum dans une liste et son rang : On utilise la méthode **`max`** :

```
>>> liste = [4, 7, 9, 4, 1, 6, 8]
>>> max(liste)
9
>>> liste.index(max(liste))
2
```

On procède de manière identique pour le minimum.

Manipulation par slicing (portion de liste) :

Afficher la troisième et quatrième valeur :

```
>>> liste = [1, 10, 100, 250, 500]
>>> liste[2:4]
[100, 250]
```

Afficher les 2 premiers éléments d'une liste :

```
>>> liste = [1, 2, 3, 4, 5, 6]
>>> liste[:2]
[1, 2]
```

Afficher les 3 derniers éléments d'une liste :

```
>>> liste[-3:]
[4, 5, 6]
```

Vider la liste :

```
>>> liste[:] = []
[]
```

Modifier la troisième et quatrième valeur :

```
>>> liste = [1, 10, 100, 250, 500]
>>> liste[2:4] = [69, 70]
>>> liste
[1, 10, 69, 70, 500]
```

8 - Opérations sur deux listes :

Fusionner deux listes :

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> x + y
[1, 2, 3, 4, 5, 6]
```

Initialiser une liste avec tous les items à `0`:

```
>>> x=[0]*5
>>> x
[0, 0, 0, 0, 0]
```