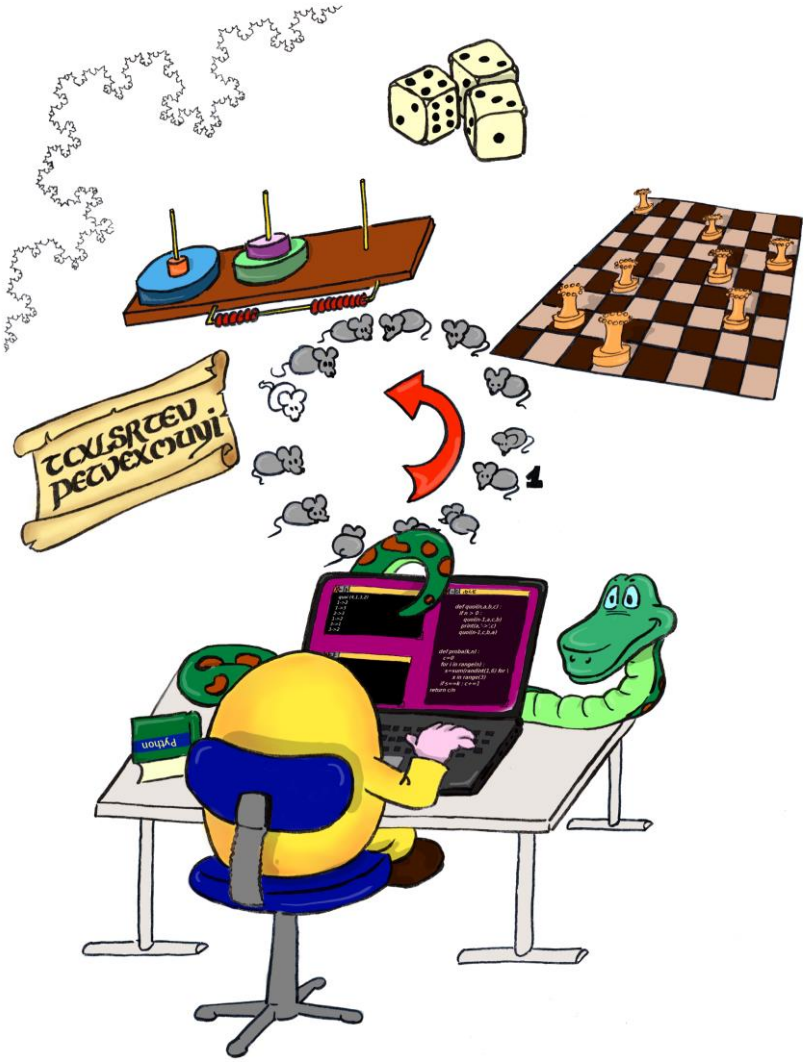
Kit de démarrage en Python



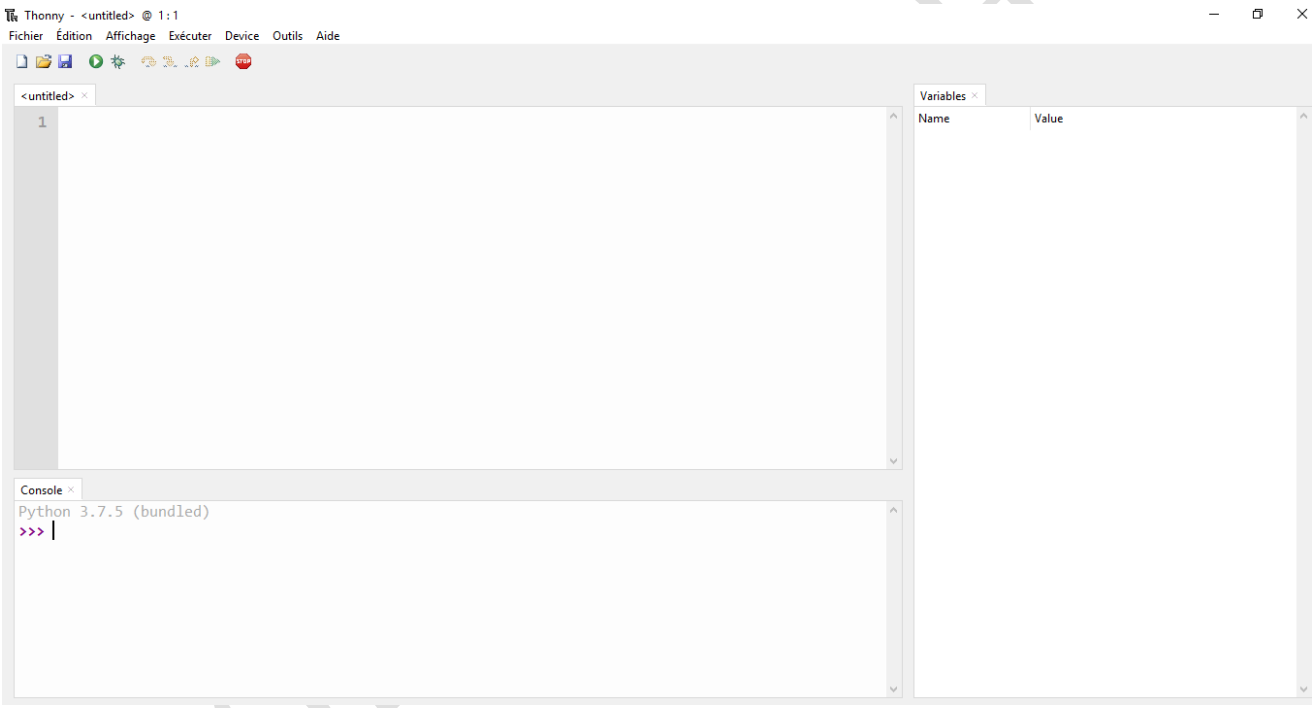
1. Un peu d'histoire

Vous allez commencer par vous rendre à l'adresse suivante :
https://www.youtube.com/watch?v=swGl-iPmYic

2. Présentation de l'interface

Python est un langage libre, et il existe de nombreux environnements de développement.
Cette année, nous allons utiliser Thonny que vous pouvez télécharger à l'adresse :
https://thonny.org/

Quand on démarre Thonny on constate que l'interface contient trois parties : une partie « interpréteur » (ou shell ou Console) qui lit du code écrit en Python et le traduit en langage machine pour que celle-ci puisse l'exécuter et vous donner le résultat ; une partie "code source" (ou Éditeur) ou l'on peut créer un fichier (extension .py) comportant des programmes écrits en Python ; une partie faisant apparaître les variables.
Une fois installé, quand vous lancez Thonny vous avez une fenêtre qui ressemble à :



Pour commencer :

- Lancer **Thonny**. On arrive alors sur « **la console Python** ».
- Il est possible de revenir en arrière à tout moment, avec les flèches du clavier ← ↑ → ↓ ou avec la souris ; les touches gauche et droite permettant de modifier les instructions et les touches haut et bas, donnant la possibilité de revenir sur une commande précédemment exécutée.

Une instruction s'exécute avec la touche « Entrée ».

3. Utiliser Python comme une calculette

La console **Python** fonctionne comme une simple calculatrice : vous pouvez y saisir une expression dont la valeur est renvoyée dès que vous pressez la touche « Entrée ».

Exercice 1 : Taper les calculs suivants dans le *shell* (regarder ce qui se passe dans la fenêtre *Workspace* au fur et à mesure).

```
5+3
2-9
7+3*4
(7+3)*4
2*5+6-(100+3)
2**7 # pour l'exponentiation (et non pas 2^7 !)
3**0.5
7/2
7/3
34//5   # quotient de la division euclidiennede 34 par 5
34%5    # reste de la division euclidiennede 34 par 5
```

Remarque : le symbole # (dièse) permet de placer des commentaires dans les lignes de commande. Tout ce qui est situé à droite du symbole # (jusqu'au changement de ligne) est purement et simplement ignoré par l'interpréteur.

4. Variables et affectation

En pratique une **affectation** s'effectue à l'aide du symbole « = ».

- Les commandes suivantes définissent et utilisent des variables.

```
x = 10
x = x + 1
largeur = 20
hauteur = 5 * 9.3
v = largeur * hauteur
r ,pi = 12, 3.14159
s = pi * r ** 2
```

- Noms de variables et mots réservés

Les noms de variables sont des noms que vous choisissez vous-mêmes assez librement. Efforcez-vous cependant de bien les choisir : de préférence assez courts, mais aussi explicites que possible, de manière à exprimer clairement ce que la variable est censée contenir.
Par exemple, des noms de variables tels que « *altitude* », « *altit* » ou « *alt* » conviennent mieux que x pour exprimer une altitude. ***Un bon programmeur doit veiller à ce que ses lignes d'instructions soient faciles à lire.***
Sous **Python**, les noms de variables doivent en outre obéir a quelques règles simples :

• Un nom de variable est une séquence de lettres (a → z , A → Z) et de chiffres (0 → 9), qui doit toujours commencer par une lettre.

• Seules les lettres ordinaires sont autorisées. Les lettres accentuées, les cédilles, les espaces, les caractères spéciaux tels que $, #, @, etc. sont interdits, a l'exception du caractère _ (souligne).

• La casse est significative (les caractères majuscules et minuscules sont distingués).

Attention : *Joseph, joseph, JOSEPH sont donc des variables différentes. Soyez attentifs !*

Prenez l'habitude d'écrire l'essentiel des noms de variables en caractères minuscules (y compris la première lettre). Il s'agit d'une simple convention, mais elle est largement respectée. N'utilisez les majuscules qu'à l'intérieur même du nom, pour en augmenter éventuellement la lisibilité, comme dans « *tableDesMatieres* ».

En plus de ces règles, il faut encore ajouter que vous ne pouvez pas utiliser comme nom de variables les 33 ≪ mots réservés ≫ ci-dessous (ils sont utilisés par le langage lui-même) :

and, as, assert, break, class, continue, def, del ,elif ,else ,except, false, finally, for, from global, if, import, in, is , lambda, none, nonlocal, not, or, pass, raise, return, true , try ,while with, yield

5. Les types numériques

Il existe quatre types numériques en Python :

- Les *int* qui représentent les nombres entiers
- Les *float* représentent les nombres décimaux
- Les *bool* qui représentent les booléens, un sous ensemble des nombres entiers

La fonction *type( )* permet de vérifier le type d'une variable.

Les entiers en Python se manipulent comme dans une calculatrice.

Les entiers en Python ont la caractéristique d'être des objets de précision illimitée. On n'a aucune perte de précision quand on les manipule.

Les *float* en Python ont une précision limitée. En général, ils sont codés sur quinze chiffres significatifs.

On peut faire des opérations sur n'importe quel type numérique. Python se charge de convertir. Ce qui peut engendrer une perte de précision.

En Python, on peut convertir un *float* vers un *int* à l'aide de l'instruction *int(4.3)* par exemple. La conversion est une troncation. De même, on peut convertir un *int* vers un *float* à l'aide de l'instruction *float(4)* par exemple.

Exercice 2 : Taper les instructions suivantes dans le *shell*.

$i = 123456789101112131415$
$type(i)$
$f = 4.3$
$type(f)$
$c = f + i$
$type(c)$

Que remarquez-vous ?

6. Instruction d'écriture et de lecture

Exercice 3 : ***Un premier programme : volume d'un cylindre***

a. Aller dans le menu Fichier puis choisir Nouveau (ou faire taper Ctrl+N)

b. Immédiatement, enregistrer le fichier avec Ctrl+S.

c. Taper le programme suivant dans l'éditeur :

```
from math import pi
r = float(input(« Entrez le rayon du cylindre : »))
h = float(input(« Entrez la hauteur du cylindre : »))
v = pi * r ** 2 * h
```

d. Enregistrer le programme puis le lancer avec F5.
e. Que remarquez-vous ?
f. Compléter le programme précédent en ajoutant l'instruction :

```
from math import pi
r = float(input("Entrez le rayon du cylindre : "))
h = float(input("Entrez la hauteur du cylindre : "))
v = pi * r ** 2 * h
print(« le volume du cylindre est = », v)
```

g. Lancer le programme. Que constatez-vous ?
h. Que font les commandes ***input*** et ***print*** ?
i. Supprimer la ligne ***from math import pi*** et lancer à nouveau le programme. Que se passe-t-il ?

Conclusion : il est nécessaire de rajouter la fonction ***print( )*** s'il on souhaite voir s'afficher la valeur d'une valeur. L'une des grandes forces d'un langage de programmation de haut niveau est qu'il permet de construire des instructions complexes par assemblage de fragments divers. Ainsi par exemple, si vous savez comment additionner deux nombres et comment afficher une valeur, vous pouvez combiner ces deux instructions en une seule :

```
print(17 + 3)
```

Cela n'a l'air de rien, mais cette fonctionnalité qui parait si évidente va vous permettre de programmer des algorithmes complexes de façon claire et concise.

Exemple :

```
h, m, s = 15, 27, 34
print(« nombre de secondes écoulées depuis minuit = », h * 3600 + m * 60 + s)
```

L'instruction ***input( )*** permet de demander à l'utilisateur de fournir des données au programme en cours d'exécution.

Exemple :

```
prenom = input(« Entrez votre prénom : »)
print(« Bonjour, », prenom)
```

Remarque : La fonction **input( )** renvoie toujours une chaîne de caractères. Si vous souhaitez que l'utilisateur entre une valeur numérique, vous devrez donc convertir la valeur entrée (qui sera donc de toute façon de type string) en une valeur numérique du type qui vous convient, par l'intermédiaire des fonctions intégrées ***int***( ) (si vous attendez un entier) ou ***float***( ) (si vous attendez un réel).
En pratique : Pour obtenir tous les opérateurs mathématiques, on a besoin de la bibliothèque *math.* Pour cela, il faut donc ajouter au dénut du programme la commande ***from math import*,*** si on veut les utiliser.

7. Structure d'une instruction conditionnelle

Il existe trois schémas possibles utilisant l'instruction *if* pour réaliser des tests.

Test simple : si alors

```
instruction normale
instruction normale
instruction normale

if condition :
        instruction
        instruction
        instruction
        …
instruction normale
instruction normale
instruction normale
…
```

Test simple : si alors autrement

```
instruction normale
instruction normale
instruction normale

if condition :
        instruction
        instruction
        instruction
        …
else :
        instruction
        instruction
        instruction

instruction normale
instruction normale
instruction normale
```

Test simple : si alors sinon…autrement

```
instruction normale
instruction normale
instruction normale

if condition :
        instruction
        instruction
        instruction
        …
elif :
        instruction
        instruction
        instruction
elif :
        instruction
        instruction
        instruction
        …
else :
        instruction
        instruction
        instruction
        …

instruction normale
instruction normale
instruction normale
…
```

Exemples :

- Entrer les deux lignes suivantes dans l'éditeur Python

```
a = 150
if (a > 100):
…
```

- Compléter l'instruction précédente par :

```
a = 150
if (a > 100):
....print (« a dépasse la centaine »)
```

Que s'affiche-t-il ?

- Recommencer le même exercice avec $a = 20$.
- Recommencer encore, en ajoutant deux lignes comme ci-dessous :

```
a = 20
if (a > 100):
....print ("a dépasse la centaine")
else :
....print ("a ne dépasse pas cent")
....
```

```
a = 0
if a > 0 :
....print ("a est positif")
elif a < 0 :
....print ("a est négatif")
else:
....print ("a est nul")
```

8. Les boucles

Dans ce paragraphe, nous allons introduire une nouvelle instruction, la boucle, qui permet d'exécuter un bloc d'instructions plusieurs fois.

Ces répétitions se classent en deux catégories :

- Les répétitions inconditionnelles : le bloc d'instructions est répété un nombre donné de fois.
- Les répétitions conditionnelles : le bloc d'instructions est répété autant donné de fois qu'une condition est vérifiée.

a. La boucle for (inconditionnelle)

C'est une répétition de la forme :

```
for i in range(n,p) :
…. bloc d'instructions
```

Remarque : comme dans le cas des tests, le corps d'une boucle doit être indenté.

Exercice 4 : Exécuter les boucles suivantes :

```
for i in range(1,11):
    print(" allo ",end = " ")
    print(« t'es où ? »)
```

```
for i in range(1,11):
    print(" allo ",end = " ")
print(« t'es où ? »)
```

```
for i in range(1,11):
    print(i)
```

```
for jour in range(1,32):
    print(jour, end = " ")
    print(" janvier ")
```

Remarque : range(n,p) renvoie l'itérateur parcourant l'intervalle [n, p-1].

b. La boucle while (conditionnelle)

C'est une répétition de la forme :

```
while condition :
    bloc d'instructions
```

Exercice 5 : Exécuter les boucles suivantes :

```
i = 1
while i <= 5:
  print(i)
  i = i + 1
```

```
i = 1
while i < 1000:
  print ("i est toujours trop petit")
  i = i + 1
```

10. Les fonctions

Dès qu'un programme devient un peu long et/ou que des blocs d'instructions ont tendances à se répéter, on a intérêt à utiliser les **fonctions** : petits sous-programmes plus ou moins indépendants du programme principal mais que l'on peut travailler de manière autonome.

Une fonction est formée de son en-tête puis de son corps.

Pour écrire l'en-tête d'une fonction, il faut :

- Choisir un nom qui indique clairement ce que fait la fonction.
- Identifier les arguments qui varient lors des différents appels de la fonction dans le programme principal. Donner un nom à ces arguments.

La syntaxe pour la définition d'une fonction est la suivante ;

```
def nomDeLaFonction(liste de paramètres):
...
  bloc d'instructions
  ...
 return resultat
```

Exemple : le volume d'un cylindre

```
from math import pi

def volume(r, h):
... v = h * pi * r ** 2
... return v
```

Attention : l'instruction ***return*** va terminer l'exécution d'une fonction, ce qui signifie qu'on placera généralement cette instruction en fin de fonction puisque le code suivant une instruction ***return*** dans une fonction ne sera jamais lu ni exécuté.