Python	Pour démarrer avec Python

Avant de commencer

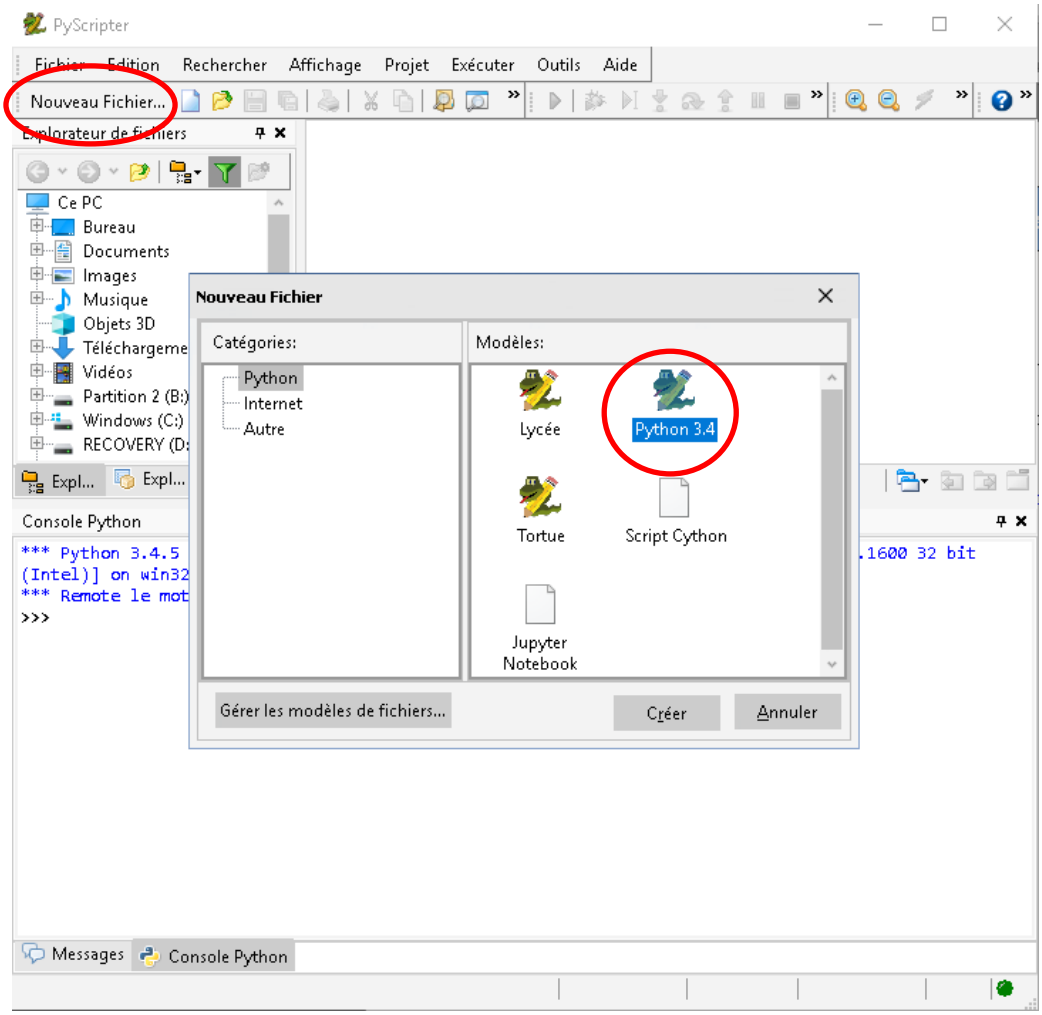
Télécharger et installer Edupython (solution clé en main pour programmer avec Python).

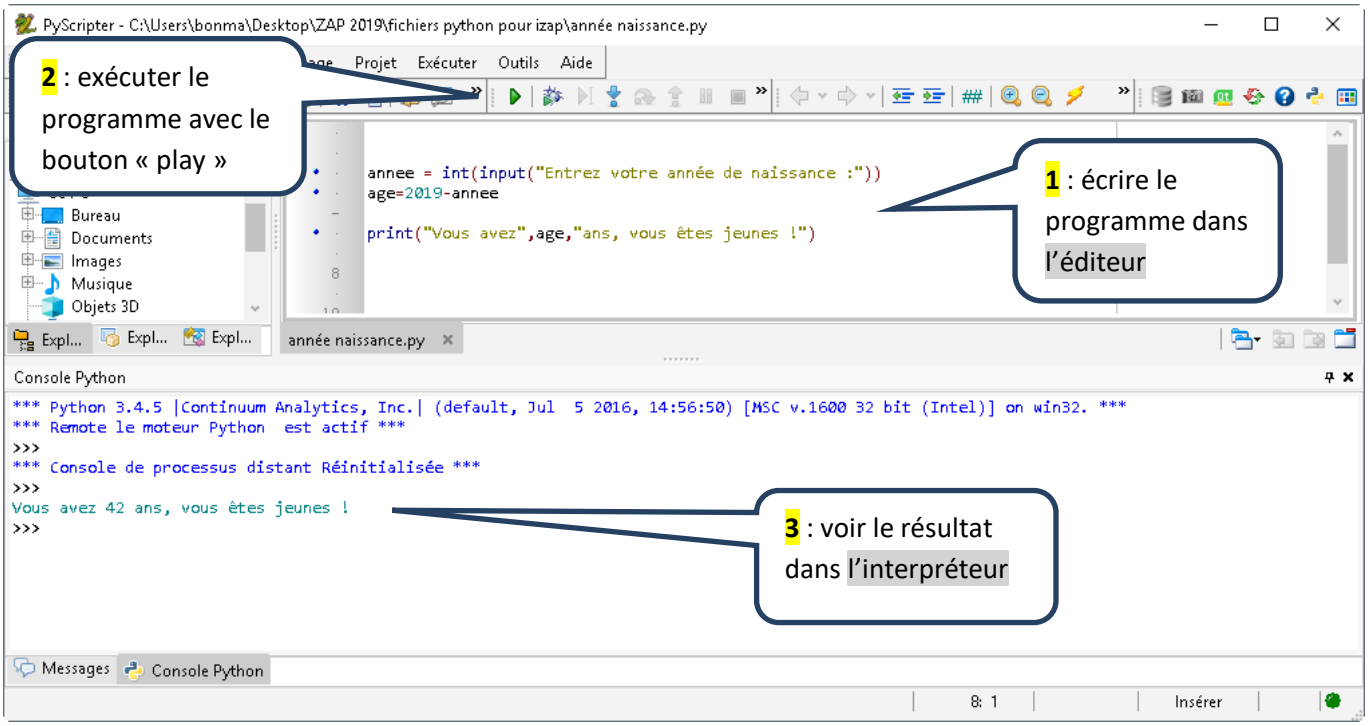
Autre environnement de développement intégré (IDE en anglais, IDLE pour les versions light) : PyCharm qui nécessite d'installer au préalable l'interpréteur Python 3.

Pour les plus curieux : https://www.supinfo.com/articles/single/7899-top-10-ide-developpeurs-python.

Découverte de l'IDE Pyscripter fourni avec Edupython

Après l'ouverture du logiciel, créer un nouveau fichier :





annee et **age** sont des variables. Les variables ne doivent jamais être accentuées.

La fonction **input()** permet de demander à l'utilisateur d'entrer une donnée. Ici l'année de naissance.

La fonction **int()** transforme la donnée entrée en entier (integer en anglais).

Pourtant j'ai bien entré un entier ! Oui mais Python l'a interprété comme du texte. Attention avec les input().

La fonction **print()** permet d'afficher le résultat. Ici l'âge de l'utilisateur.

Le texte entre guillemets est une **chaîne de caractères** (string en anglais). On aurait pu utiliser des apostrophes.

Ex1 : En s'appuyant sur l'exemple précédent, écrire un programme qui demande un entier n et renvoie le $n^{ième}$ terme de la suite arithmétique définie par $u_1 = 5$ et $r = 1{,}5$.

Par exemple, si on entre 12, le programme renvoie : >>> Le terme de rang 12 est 21.5

Attention : en programmation, le séparateur décimal est le point (.).

Si on a besoin de faire des calculs nécessitant π, un cosinus, un logarithme, …, il faut l'**importer du module math.** Voir ce lien pour en savoir plus.

Par exemple, pour calculer le volume d'un cylindre de rayon R et de hauteur 5 :

```
from math import pi
R=float(input("rayon du cylindre de hauteur 5 :"))
volume = round(pi*R**2*5,2)
print(volume)
```

On a besoin de π, donc on l'importe. Pour se simplifier la vie, on pourrait tout importer : **from math import ***

La fonction **float()** permet de déclarer la variable R comme un nombre décimal … sinon ce sera un string !

La fonction **round()** arrondit le calcul du volume, ici à 2 décimales. On l'utilise ainsi : round(valeur,précision).

Ex2 : Écrire un programme qui affiche au dixième près la valeur du volume d'un cône de rayon r et de hauteur h. Ces deux variables étant décimales.

Un peu d'aide : la puissance s'obtient avec deux astérisques. Par exemple 2**3 donne 8.

La boucle For

Écrivez ce programme dans l'éditeur :

```
for i in range(5):
    print(i)
```

Que fait-il ? Il affiche les entiers de 0 à 4.

i varie de 0 inclus à 5 exclu, donc on boucle jusqu'à 4. Retenir qu'en général toute borne supérieure est exclue.

En écrivant le programme, vous avez peut-être remarqué la tabulation automatique de 4 espaces, on appelle ça l'indentation et provoque une erreur à l'exécution si elle n'est pas respectée.

La fonction **range()** peux prendre jusqu'à 3 paramètres. Regardez ce que donnent range(2,10) et range(2,10,2).

Voici un autre exemple :

```
S=0
for i in range(5):
    S=S+i
print(S)
```

boucle

Que fait le programme ? ……

Il faut initialiser S à 0 car cette variable intervient dans le calcul S=S+i, qui signifie « $S_{suivant} = S_{précédent} + i$ ».

Remarquez le print en dehors de la boucle. Contrairement à l'exemple précédent, on imprime qu'une fois !

Ex3 : Modifier le programme de l'Ex1 pour afficher les n premiers termes de la suite et, à la fin, leur somme S.
Par exemple, pour n=2, le programme renvoie : >>> Le terme de rang 1 est 5
Le terme de rang 2 est 6.5
La somme des termes vaut 11.5

Allez, j'ose compliquer un peu ! Voici une autre version du programme :

```
n = int(input("entrer un entier : "))
u_n=[5+(i-1)*1.5 for i in range(1,n+1)]
print("u_n =",u_n,'\n',"La somme des termes vaut",sum(u_n))
```

u_n est une liste. Une liste est toujours définie par des crochets et sa somme s'obtient avec **sum()**.

La boucle for est écrite sur une seule ligne. On appelle ça une compréhension … de liste. Efficace, mais à éviter avec nos élèves, au moins au début.

"\n" pour forcer le retour à la ligne … mais le tout dans un seul print, ce n'est pas très pédagogique non plus.

Ex4 : Écrire un programme qui renvoie le tableau de valeurs de la fonction définie sur [-1 ; 5] par f(x) = 3x+1.
Ce tableau de valeurs se présentera sous la forme d'une liste de tuples : >>> tableau = [(-1,-2), (0, 1), …]

C'est quoi des tuples ? Ce sont des objets de la forme (a,b) ou (a,b,c). Utiles pour des coordonnées.

Un peu d'aide : initialiser une liste vide avec tableau = [] puis dans la boucle for, y ajouter les tuples avec la méthode **append()** : tableau.append((x,3x+1)). Ne pas oublier le print final !

Une version du programme en une seule ligne :

```
print("tableau = ",[(x,3*x+1)for x in range(-1,6)])
```

Pour amuser les élèves, et pour voir une autre façon d'utiliser la boucle for, regardons ce programme :

```
mot = "anticonstitutionnellement"

n=0
for c in mot :
    n=n+1
print(n)
```

Que fait-il ? ……

La variable d'incrémentation est c, pour caractère. Mais toute autre lettre convient ! Testez !

On reverra cette activité avec l'instruction conditionnelle if …. Pour compter le nombre de n dans le mot 😊.

Attention : Testez avec `mot = "Python c'est bien"` . Quel est le problème ? ……

La boucle while

Écrivez ce programme dans l'éditeur :

```
i=0
while i<5:
    print(i)
    i=i+1
```

Il renvoie … exactement la même chose qu'avec une boucle for ! En fait, Il est toujours possible de transformer une boucle for en boucle while. L'inverse est vrai à condition de connaître à l'avance le nombre d'itérations.

Pour que la boucle fonctionne, il faut initialiser la variable i (ici à 0), et l'incrémenter dans la boucle.

Dernier point, vérifiez avec Python Tutor qu'au sortir de la boucle, i = 5. Il faudra parfois être vigilent sur ce point.

Supposons que l'on veuille seulement la somme S de la suite de l'exercice 3. On pourra écrire :

```
n = int(input("entrer un entier n "))
S=0
i=1
while i<=n:
    S+=3.5+1.5*i
    i+=1
print("La somme des termes vaut",S)
```

Remarquez l'écriture des incréments : i+=1 qui est équivalent à i=i+1. Peut-être plus compréhensible !

Ex5 : Écrire et compléter le programme permettant de trouver un nombre mystère compris entre 1 et 10.

```
from random import randint
myst=randint(1,10)
N=1
A=int(input("choisissez un entier entre 1 et 10 :"))
while [redacted]
    A=int(input("Perdu, recommencez :"))
    [redacted]
print("Il vous a fallu  ", N, " essai(s) pour trouver !")
```

Un peu d'aide : en informatique le symbole ≠ s'écrit !=.

Random est un module au même titre que math et turtle, duquel on importe la fonction **randint()**.

Ex6 : Écrire un programme qui affiche la somme des n premiers entiers (n ≥ 1) de trois façons différentes :

- En utilisant la formule classique : s=n(n+1)/2 ;
- En utilisant une boucle for ;
- En utilisant une boucle while.

Ex7 : Écrire un programme qui réponde à cette question : quel est le plus grand entier vérifiant l'inéquation $2^x \leq N$, avec N entier ?

Attention, il y a un piège : il faut toujours prendre garde à la valeur de sortie des incréments.

L'instruction If

Elle permet de vérifier si une condition est remplie. Regardons cet exemple :

```
n=int(input("entrez un nombre entier :"))

if n%2==0:
    print("le nombre est pair")
else :
    print("le nombre est impair")
```

Si – *if* – le reste de la division de n par 2 est nul (modulo, noté %), alors n est pair. Sinon – *else* –, il est impair.

Il y a un double égal car il s'agit ici d'une comparaison, pas d'une affectation. Les opérateurs de comparaison sont :

< , > , <= , >= , == , !=, is, is not

Ex8 : Écrire un programme qui demande à l'utilisateur la distance D, en km, qu'il a parcourue avec une voiture de location et renvoie le prix à payer P, en €, selon deux tarifs :

- Si D < 50 km alors le tarif est de 20 cts par km,
- Si D ≥ 50 km alors le tarif est de 15 cts par km.

On peut combiner boucle et instruction conditionnelle. Regardons ce programme :

```
mot="anticonstitutionnellement"

compteur=0
for i in range(len(mot)) :
    if mot[i]=="n":
        compteur+=1
print(compteur)
```

Vous avez compris … le programme compte le nombre de n. Des explications s'imposent :

La fonction **len()** mesure la longueur du mot (length en anglais). Marche aussi pour les listes et les tuples.

i varie de 0 à 25 non compris avec mot[0] la première lettre, mot[1] la deuxième, …, mot[24] la dernière.

Dans un string, une liste ou un tuple, le rang commence à l'indice 0 et il faut des crochets.

Sauriez-vous compter le nombre de i du mot *indivisibilité* ? C'est le mot de la langue française qui en a le plus !

Ex9 : En utilisant l'instruction conditionnelle complète **if – elif – else**, écrire la boucle qui complète le programme. Il dessine, avec turtle, une ligne brisée de 10 segments orientés aléatoirement selon ce principe :

- Pour chaque segment à tracer un nombre aléatoire entre 0 et 3 est donné.
- Si la valeur entière (int) du nombre aléatoire est 0 alors avancer de 50 pixels : forward(50)
- Ou si la valeur entière du nombre aléatoire est 1 alors tourner à gauche de 60° et avancer de 50 pixels : left(60) forward(50)
- Autrement tourner à droite de 60° et avancer de 50 pixels : right(60) forward(50)

```
from turtle import *
import random

up()                    # on lève le crayon

i=0
while i<10:
    alea=random.randint(0,3)
    color("blue")
    width(3)                    # ou bien pensize() pour l'épaisseur du trait
    down()                      # on baisse le crayon
    Instruction conditionnelle
    i+=1

up()
hideturtle()    # supprime la tortue
done()          # laisse ouverte la fenêtre de dessin
```

Ex10 : Écrire un programme qui vérifie l'égalité de Pythagore sur 3 entiers entrés par l'utilisateur. Pour détecter le plus grand, on pourra définir la liste L=[a, b, c] et la trier par ordre croissant avec L=sorted(L) ou L.sort().

Attention les deux plus petits nombres seront L[0] et L[1] et le plus grand sera au choix max(L) ou L(2].

Les réponses possibles seront : Le triangle est/n'est pas rectangle.

Les fonctions :

Les fonctions permettent de structurer un programme complexe en un ensemble de tâches plus simples. De même, si un programme doit répéter plusieurs fois la même tâche, il sera plus judicieux de l'inclure dans une fonction. On obtiendra un programme plus simple, plus fluide.

Voici deux versions d'un programme donnant la table de multiplication par n :

```
"""table de multiplication sans fonction"""

n=int(input("Vous voulez la table de ?"))

i=1
while i<11:
    print(i,"x",n,"=",n*i)
    i+=1
```

```
"""table de multiplication avec fonction"""

def ma_table(nbre):
    i=1
    while i<11:
        res = print(i,"x",nbre,"=",nbre*i)
        i+=1
    return res

n=int(input("Vous voulez la table de ?"))

ma_table(n)
```

Les deux programmes renvoient exactement la même chose. Alors pourquoi faire simple …… ! Parce que dans un programme long avec des tâches répétitives, ce sera effectivement plus simple. Mais là, ça ne se voit pas !

Une fonction commence par : **Def** nom_de_la_fonction(arguments) et se termine par **return**.

Dans notre exemple la fonction est appelée par `ma_table(n)` et renvoie -*return*- le résultat res.

nbre est l'argument de la fonction et prend la valeur n.

Une fonction peut ne pas avoir d'argument ou en avoir plusieurs : fonction() ou fonction(arg1, arg2,….).

Un détail d'importance : ajoutez print(i) à la fin des deux programmes. Que se passe-t-il ? Sans fonction on a 11, et dans l'autre une erreur. Pourquoi ?

Les variables contenues dans un programme sont des variables globales et existent tout le temps, alors que celles contenues dans une fonction sont des variables locales et cessent d'exister sitôt le return passé.

Voici un exemple de fonction donnant la liste des n premiers carrés parfaits :

```
def carre(nbre):
    L=[i**2 for i in range(1,nbre+1)]
    return print(L)

n=int(input("entrer un entier : "))

carre(n)
```

L est une liste en compréhension …. C'est juste pour créer un automatisme 😊.

Rien à ajouter sauf que n est une variable globale et nbre une variable locale….

Arrivés au terme de ce document, nous n'avons jamais vraiment utilisé l'interpréteur. Corrigeons cet oubli :

Ex11 : Écrire cette fonction dans l'interpréteur : >>> def f(x) :
Return 2*x+1
Valider jusqu'à la réapparition du triple prompt (>>>).

Entrer ensuite f(2) et valider. Tester plusieurs valeurs …. Voilà un bon début pour les élèves !

Cet exercice est tiré de cette page.

Ex12 : Modifier la fonction de la table de multiplication pour qu'elle passe en arguments le numéro de la table et la valeur maximum pour laquelle l'utilisateur souhaite cette table. En clair :

Si l'utilisateur veut la table de 8 jusqu'à 25, le programme renverra : >>> 1 x 8 = 8
2 x 8 = 16
…
…
24 x 8 = 192
25 x 8 = 200

Ne pas oublier l'input supplémentaire.

Enfin pour clore ce document d'initiation à Python, sachez qu'il existe sur internet énormément de tutos, de forums, de cours en ligne, etc., dans lesquels vous trouverez certainement les réponses à vos interrogations.

N'hésitez pas à consulter la documentation Python ainsi que son tutoriel très bien renseigné.