académie Nice
RÉGION ACADÉMIQUE
PROVENCE-ALPES-CÔTE D'AZUR
MINISTÈRE
DE L'ÉDUCATION NATIONALE
MINISTÈRE
DE L'ENSEIGNEMENT SUPÉRIEUR,
DE LA RECHERCHE
ET DE L'INNOVATION
Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

Aménagement du programme de Mathématiques de seconde 2017

Algorithme et Programmation

Guide de démarrage au langage Python

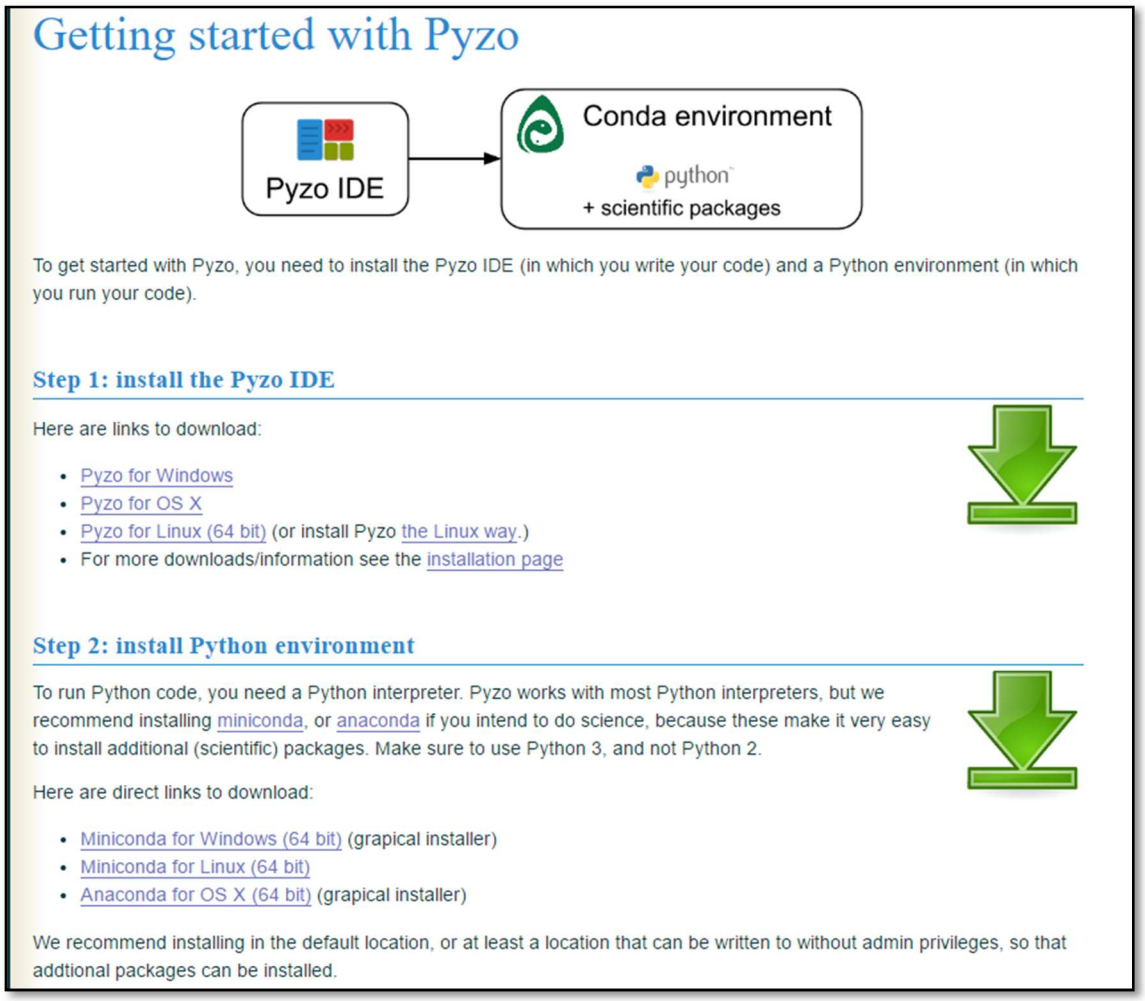
Document rédigé sous la direction des IA-IPR de l'académie de Nice, par D. Lacroix et L. Faubourg, avec l'aide des correspondants en informatique et algorithmique de L'académie de Nice.

Ce document est disponible en ligne sur : http://www.ac-nice.fr/maths/mathv2/

Le but de ce document est de permettre un démarrage rapide dans la programmation en python. Il ne constitue en aucun cas un cours sur la programmation en python. Vous trouverez des cours complets et détaillés dans le paragraphe 4 « liens et cours ».

1- Installation de la suite Pyzo :

La suite Pyzo est un logiciel qui permet d'éditer, de tester et de mettre au point des programmes écrits en python. Pour en faire l'installation, il faut se rendre sur la page http://www.pyzo.org/start.html .



Remarques pratiques :

1. Assurez-vous d'installer python 3 à l'étape 2
2. Pour débuter, **il n'est pas nécessaire d'installer les bibliothèques** supplémentaires dans un premier temps. Mais si vous le faites il est conseillé d'installer les bibliothèques *« numpy », « mathplotlib »* etc…. non pas avec *conda* mais plutôt en utilisant les instructions suivantes dans la fenêtre *shell :*
   « pip install numpy »
   « pip install mathplotlib » .....

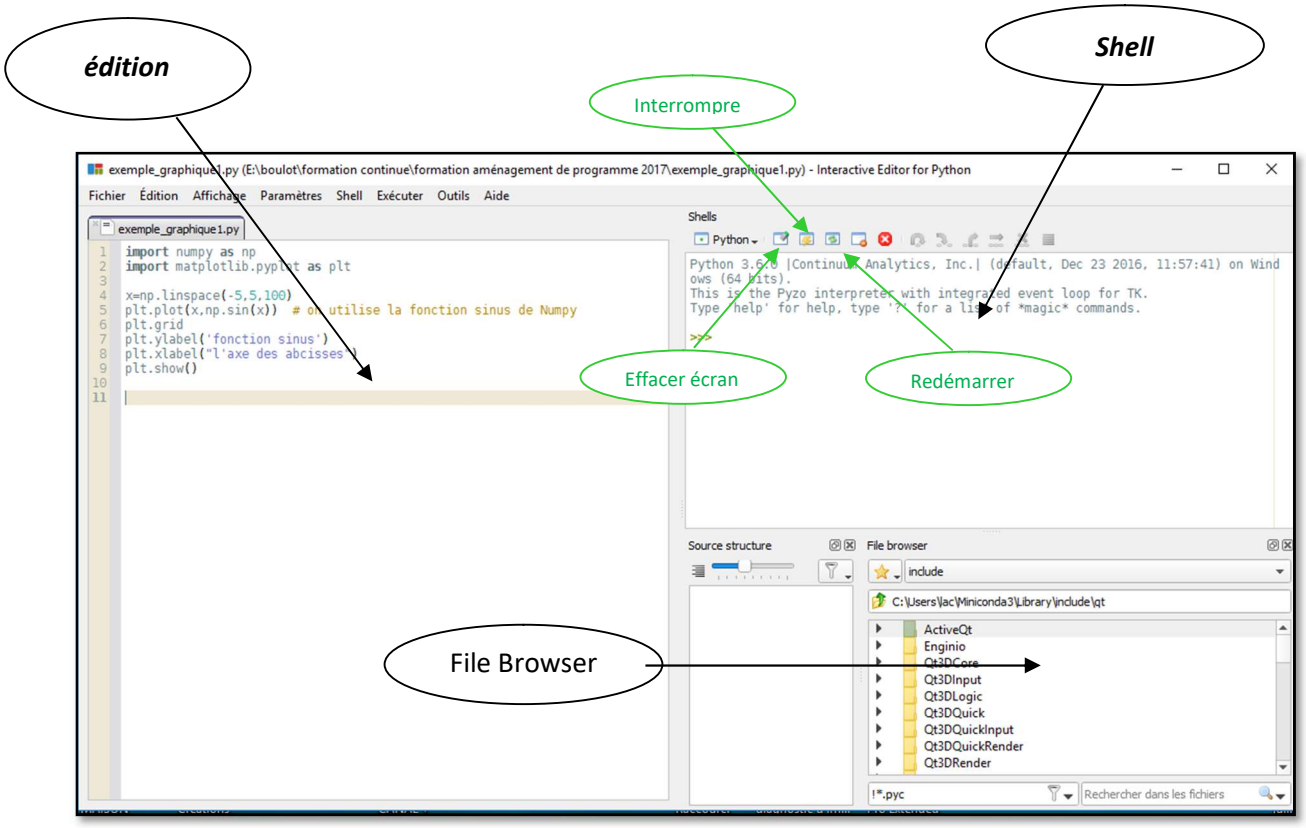
(cf la fin de ce document pour des informations plus détaillées sur les bibliothèques)

2- Description de l'environnement :

Après l'installation de Pyzo, vous pouvez lancer l'application, puis sélectionner la langue en choisissant le menu Settings-Select Language. Un redémarrage de l'application est nécessaire pour que cette modification soit prise en compte.

L'organisation de l'environnement est simple et faite autour de trois fenêtres :

- Une fenêtre *« shell »* qui a pour fonction de
  - exécuter des commandes
  - exécuter des programmes.
  - consulter les valeurs courantes des variables.
  - administrer Python (installation de nouvelles librairies)
- Une fenêtre d'édition où l'on édite, modifie et enregistre des programmes qui seront exécutés dans le « shell ».
- Une fenêtre « File browser » qui permet d'accéder à l'arborescence de ses dossiers.



La fenêtre « shell » présente quelques boutons utiles :

- Le bouton « interrompre » qui permet d'interrompre un programme en cours d'exécution (pratique lors d'une boucle infinie par exemple).
- Le bouton « Effacer écran » qui permet d'effacer la fenêtre « shell ».
- Le bouton « Redémarrer » qui réinitialise le « shell » en effaçant toutes les données.

3- Des exemples de programme en python :

3-0 Prise en main de l'environnement :

Travailler dans le « shell » : Pour commencer on peut écrire directement des commandes dans le « shell ».

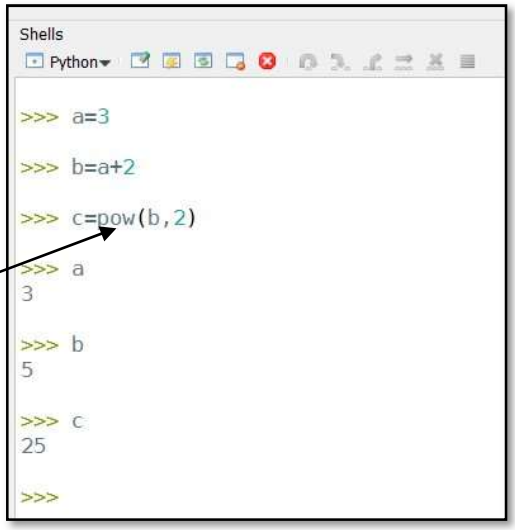
Exemple 1 :

On considère l'algorithme ci-dessous écrit en langage naturel et sa traduction en Python dans le « shell »

$a \leftarrow 3$
$b \leftarrow a + 2$
$c \leftarrow b^2$

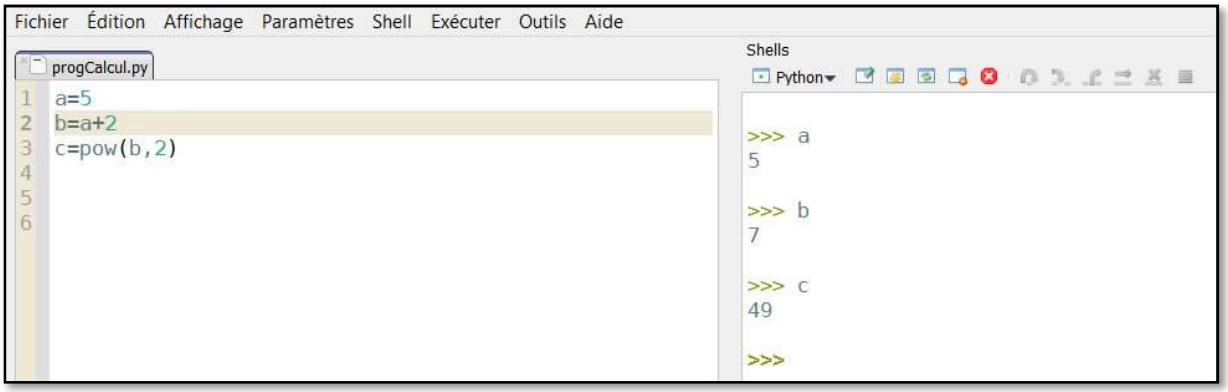
Remarquer la fonction puissance

Dans le « shell », recopier les lignes suivantes en pensant à valider chaque ligne en appuyant sur la touche entrée



Travailler en mode édition : On peut aussi écrire des programmes dans la fenêtre « édition ». Ceci permet en particulier d'enregistrer son programme.

Exemple 2 :



1) Ecrire le programme de calculs précédent dans la fenêtre « édition ».
2) Enregistrer le Fichier : menu Fichier - Enregistrer (ou CTRL S).
3) Exécuter le programme : menu Exécuter – Exécuter le contenu de l'onglet (ou CTRL E). (penser à toujours enregistrer avant d'exécuter pour que les dernières modifications soient prises en compte)
4) Vérifier les nouvelles valeurs des variables dans le « shell » en tapant a, puis b puis c.

Exemple 3 : Le programme suivant calcule le périmètre et l'aire d'un rectangle :

1) Ouvrir un nouvel onglet dans la fenêtre « édition » : menu Fichier – Nouveau (ou CTRL N)
2) Recopier puis Exécuter le programme écrit dans la fenêtre « édition »
3) Vérifier le contenu des variables « aire » et « perimetre » dans le « shell »

Remarque :

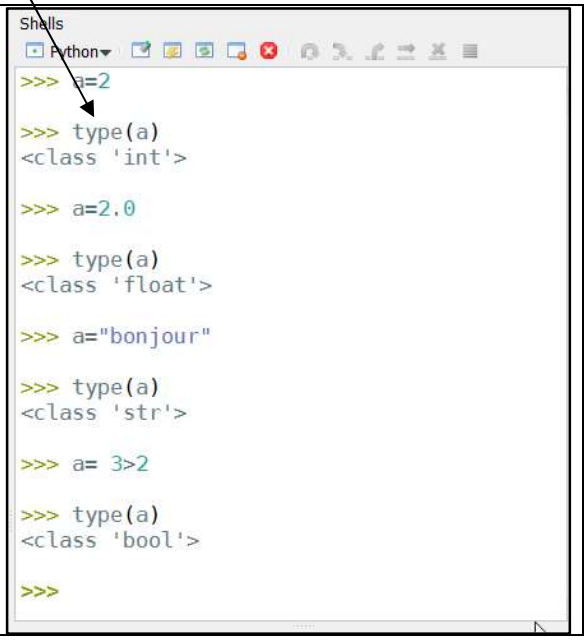
- Notez que le caractère « # » permet d'introduire des commentaires dans les programmes, ce qui constitue une bonne pratique.
- En algorithmique, les fonctions d'entrée/sortie (Lire, Afficher) étaient largement utilisées. Elles ne font pas partie de la pensée algorithmique et sont supprimées des algorithmes en langage naturel dans l'aménagement de programme et au baccalauréat. En langage Python, on préférera modifier ou accéder au contenu des variables plutôt que d'utiliser les fonctions d'entrées sorties (*print()* et *in put()*). La notion de « fonction » de la section 3-2 permet de se libérer complétement des entrées/sorties. La fonction *print()* peut, tout de même, rester utile en phase de débogages pour afficher le contenu d'une variable intermédiaire.

3-1 Les variables :

Dans le langage Python, affecter une valeur à une variable se fait à l'aide du signe « = ». Les variables ne doivent pas forcément être déclarées à l'avance. La fonction *type()* permet de consulter le type de la variable comme indiqué dans l'exemple ci-dessous.

Exemple 4 :
Taper dans le « shell » les affectations suivantes puis vérifier le type de la variable avec la fonction *type()*.

On remarque que la variable *a* change de type en passant par un type int (entier relatif), float (réel), str (chaine de caractère), puis bool (booléen : true ou false)



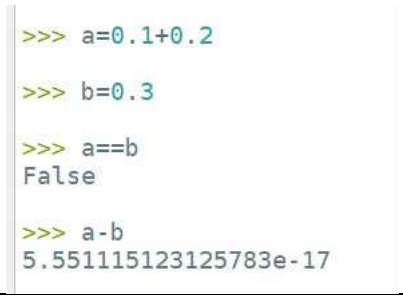
Remarque :

- Le nom d'une variable ne doit pas commencer par un chiffre. Une bonne pratique consiste aussi à ne pas utiliser les lettres accentuées.
- En langage Python, les variables utilisées ne sont pas forcément « typées » de manière explicite par le programmeur : entier (*int*), chaine de caractère (*string*)…. Le typage est dynamique en fonction des valeurs affectées à la variable et ce typage peut changer au cours d'un programme. Ainsi $a = 2$ est de type *int* et $a = 2.0$ est de type *float* .

IMPORTANT : Nous serons souvent amenés à faire des tests d'égalité de deux nombres décimaux. Or, ces tests renverront souvent des résultats faux car le développement dyadique d'un nombre décimal peut être infini incompatible avec le codage informatique fini. En effet,

$$0{,}1=(0{,}0001100110011001100110011001100110011001100110011\cdots)_2$$

La valeur stockée de 0,1 dans l'ordinateur sera donc une valeur approchée !

Exemple 5 : L'exemple ci-contre montre que le test 0,1+0,2 = 0,3 renvoie la valeur fausse sur Python.	

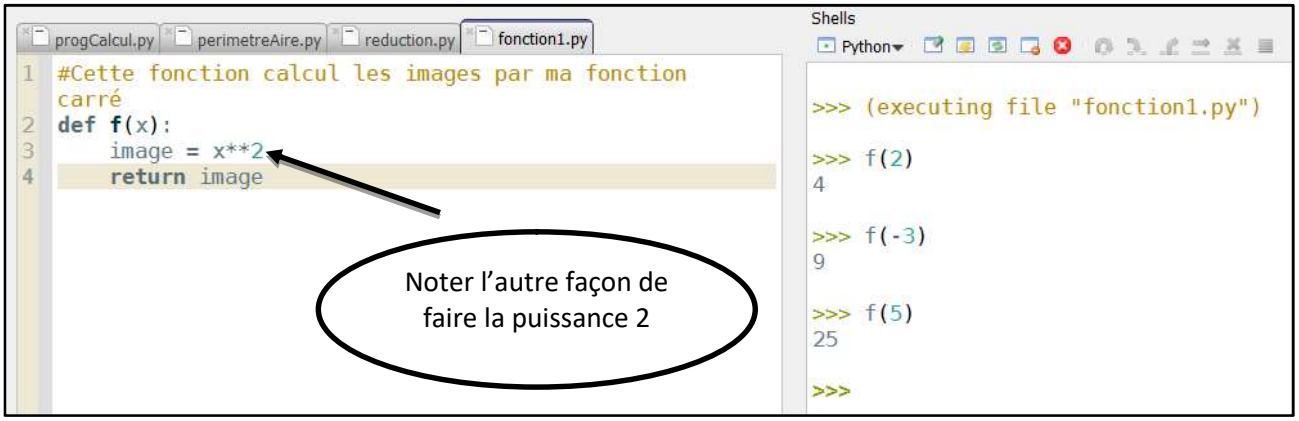
3-2 Les fonctions :

La notion de fonction en programmation est la grande nouveauté de l'aménagement de programme. Elle permet de structurer le programme en le découpant en sous-programmes indépendants. Elle permet de revisiter la notion de fonction en mathématiques avec qui elle possède un lien fort.

La structure générale d'une fonction sous Python est la suivante :

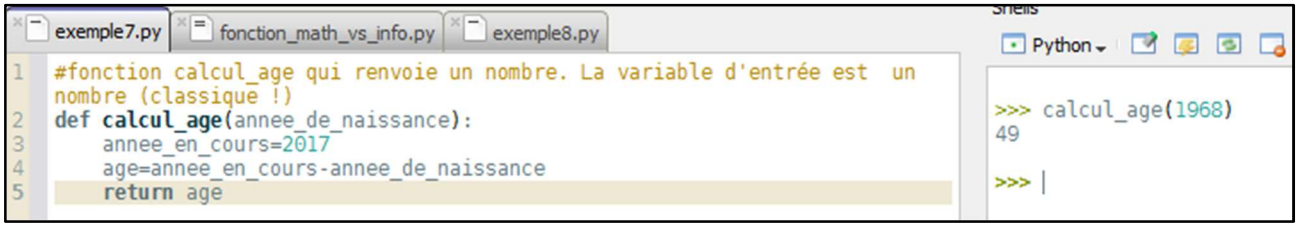
def *nomFonction*(paramètres) :
  instructions
  return valeur
	• *def* indique le début de la fonction.
	• *nomFonction* est le choix du nom de la fonction.
	• Les « : » marquent le début de la fonction
	• Les « paramètres » sont les arguments dont dépend la fonction. Ils remplacent les entrées (*input()*)
	• Toutes les instructions de la fonction sont reconnues grâce à l'indentation (décalage vers la droite). La fonction se termine lorsque l'indentation (décalage sur la droite) se termine. L'indentation remplace { } dans d'autres langages.
	• La commande « return » permet de renvoyer une ou plusieurs valeurs. Ces valeurs renvoyées remplacent les sorties (*print()*)

Exemple 6 : Le programme suivant calcule l'image par une fonction. Recopier puis exécuter le programme ci-dessus. Puis appeler la fonction dans le « shell » comme ci-dessous.



Exemple 7 : Un autre exemple avec un paramètre (le calcul d'âge)

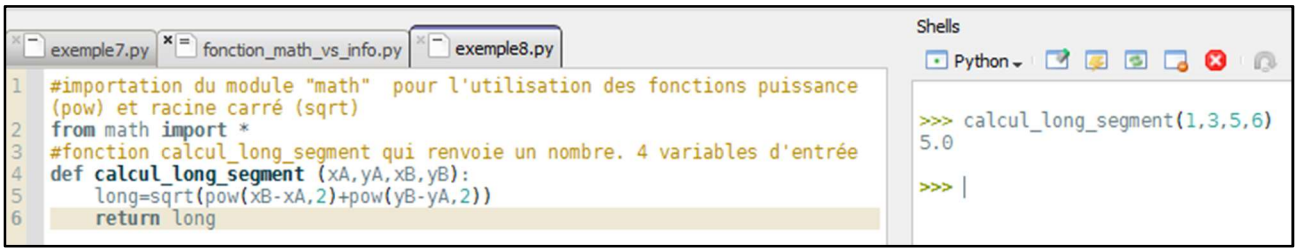
Recopier puis exécuter le programme ci-dessous



Exemple 8 : Un exemple avec plusieurs paramètres.

La fonction *calc_long_segment()* calcule le longueur d'un segment à partir des coordonnées des points.

Recopier puis exécuter le programme ci-dessous. Puis appeler la fonction dans le « shell ».

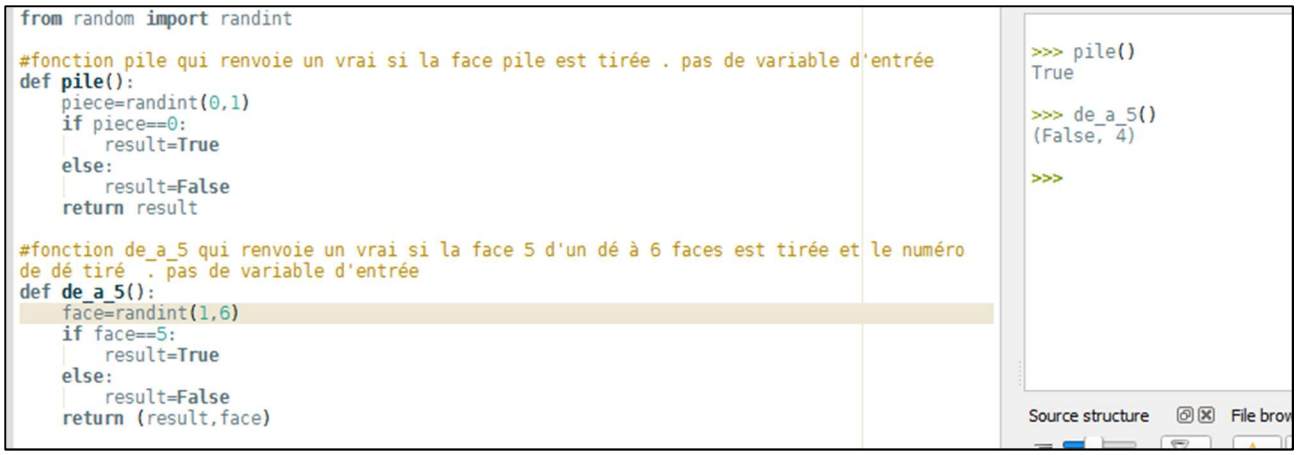


Remarques :

- La fonction utilise plusieurs paramètres et renvoie un nombre.
- Dans cet exemple, la bibliothèque *math* est appelée (aucun besoin de l'installer, elle est présente par défaut dans la distribution Pyzo. Il est néanmoins nécessaire de la charger). Dans le paragraphe 3-4, nous verrons qu'il existe d'autres bibliothèques qui nécessitent parfois une installation supplémentaire.

Exemple 9 : Deux exemples sans paramètres avec une ou plusieurs valeurs renvoyées.

Recopier puis exécuter le programme ci-dessous. Puis appeler les deux fonctions dans le « shell ».



```
from random import randint

#fonction pile qui renvoie un vrai si la face pile est tirée . pas de variable d'entrée
def pile():
    piece=randint(0,1)
    if piece==0:
        result=True
    else:
        result=False
    return result

#fonction de_a_5 qui renvoie un vrai si la face 5 d'un dé à 6 faces est tirée et le numéro
de dé tiré  . pas de variable d'entrée
def de_a_5():
    face=randint(1,6)
    if face==5:
        result=True
    else:
        result=False
    return (result,face)
```

```
>>> pile()
True

>>> de_a_5()
(False, 4)

>>>
```

Source structure File brow

Remarque :

- Dans cet exemple, la bibliothèque *random* est appelée (aucun besoin de l'installer, elle est présente par défaut dans python). Dans cette méthode d'appel, seul la fonction *randint()* est chargée. Cela évite de charger toute la bibliothèque pour n'utiliser qu'une seule fonction.
- La fonction *randint()* permet de tirer un nombre entier aléatoire entre deux bornes comprises.
- Les deux fonctions *pile()* et *de_a_5()* n'ont pas de paramètres d'entrée . *pile()* renvoie un booléen et *de_a_5()* renvoie un entier et un booléen.
- Plusieurs fonctions peuvent coexister dans le même script.
- La structure « if (condition) : Else : » est utilisée ici. Sa syntaxe sera détaillée au 3-3-1

3-3 Les structures algorithmiques :

3-3-1 instructions conditionnelles :

Si … alors … sinon …

```
from random import randint
#fonction de_a_5 qui renvoie un vrai si la face 5 d'un dé à 6 faces est tirée et le numéro
de dé tiré  . pas de variable d'entrée
def de_a_5():
    face=randint(1,6)
    if face==5:
        result=True
    else:
        result=False
    return (result,face)
```

Structure du test conditionnel :	
if (condition) : instructions else : Instructions	• On peut remarquer les « : » qui indiquent le début des instructions • Les indentations indiquent les instructions soumises au *if* puis au *else*. • L'indentation remplace les « { } » présentes dans d'autres langages

Remarques :

- Au niveau des conditions, il faut être attentif test d'égalité :
  - « face=5 » est une affectation et signifie « face prend la valeur 5 »
  - « face==5 » est un test qui renvoie vrai si « face est égal à 5 »

C'est donc bien « == » que l'on doit écrire dans un test d'égalité ;

- « a!= 3» est le test qui renvoie vrai si a est différent de 3.
- Attention au test d'égalité sur les nombres décimaux :

Le codage des nombres décimaux impose d'éviter les tests d'égalité sur les nombres décimaux.

Et de les remplacer par un test d'inégalité avec une précision.

```
>>> 0.1+0.2==0.3
False

>>> 0.1+0.2
0.30000000000000004

>>> 0.1+0.2-0.3
5.551115123125783e-17

>>> 0.1+0.2-0.3 <0.000000000001
True

>>>
```

Exemple 10 :

Un entrepreneur vend 2 types des produits A et B. Le prix unitaire de chaque pièce dépend du nombre de pièces achetées. Le détail des prix figure dans le tableau ci-dessous.

Type de pièces	Quantité <50	Quantité ≥ 50
Pièce de type A	90	70
Pièce de type B	40	30

L'entrepreneur souhaite un outil logiciel lui permettant de répondre rapidement à un client qui lui demande un devis pour une quantité nA de pièces A et nB de pièces B.

Une solution :

On peut remarquer que toutes les instructions à réaliser dans la fonction sont indentées (décalées vers la droite) à partir du « : ». La fin des instructions de la fonction est liée à la fin de l'indentation.

```
#Exemple 10

def calcul_prix_partiel(n,p1,p2):
    if n<50 :
        prix = n*p1
    else :
        prix = n*p2
    return prix

def calcul_prix_total(na,nb):
    prix_total = calcul_prix_partiel(na,90,70)+calcul_prix_partiel(nb,40,30)
    return prix_total
```

Recopier puis exécuter le programme ci-dessus. Remarquons que l'utilisation des paramètres « na » et « nb » permettent de remplacer les fonctions d'entrée par un appel de la fonction dans le shell avec les paramètres souhaités.

Exemple : « calcul_prix_total(1,1) » renvoie « 130 »

Si … alors … sinon si… sinon…

Cette instruction permet la discrimination à plus de deux cas. Elle est structurée de la manière suivante :

```
if (condition) :
        instruction1
elif :
        Instruction2
elif :
        Instruction3
else :
        Instruction4
```

```
# tirage d'un dé à 6 faces. gagné dans le cas où la face tirée est 1,5
ou 6

from random import randint
d = randint(1, 6)


if d == 1:
    resultat = True
    chaine= "gagn"
elif d == 5:
    resultat = True
    chaine= "gagn"
elif d == 6:
    resultat = True
    chaine= "gagn"
else:
    resultat=  False
    chaine= "perdu"
```

Exemple 11 :

Reprenons l'exercice précédent. L'entrepreneur décide en fait de modifier la structure de ces prix selon le tableau suivant. Adapter votre programme à la nouvelle situation.

Type de pièces	Quantité <= 50	50<Quantité <= 100	100 < Quantité
Pièce de type A	90	70	60
Pièce de type B	40	30	25

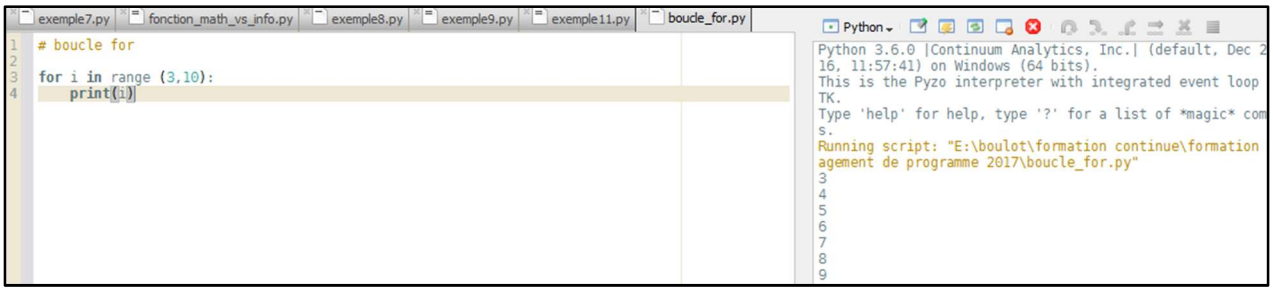
Une solution

```
#Exemple 11

def calcul_prix_partiel(n,p1,p2,p3):
    if n<=50 :
        prix = n*p1
    elif n<=100:
        prix = n*p2
    else :
        prix = n*p3
    return prix

def calcul_prix_total(na,nb):
    prix_total = calcul_prix_partiel(na,90,70,60)+calcul_prix_partiel(nb,40,30,25)
    return prix_total
```

3-3-2 structures itératives

Structures itératives Bornées : la boucle for

L'instruction *for* est une boucle qui se répète lorsqu'un index parcours les éléments d'une liste :



Structure du *for* :	• La fonction *range(a,b)* renvoie une liste contenant les nombres entiers entre a et b. • La borne a est incluse et la borne b est exclue. • L'indentation détermine les instructions réalisées dans la boucle.
for i in range(a,b) : Instructions	

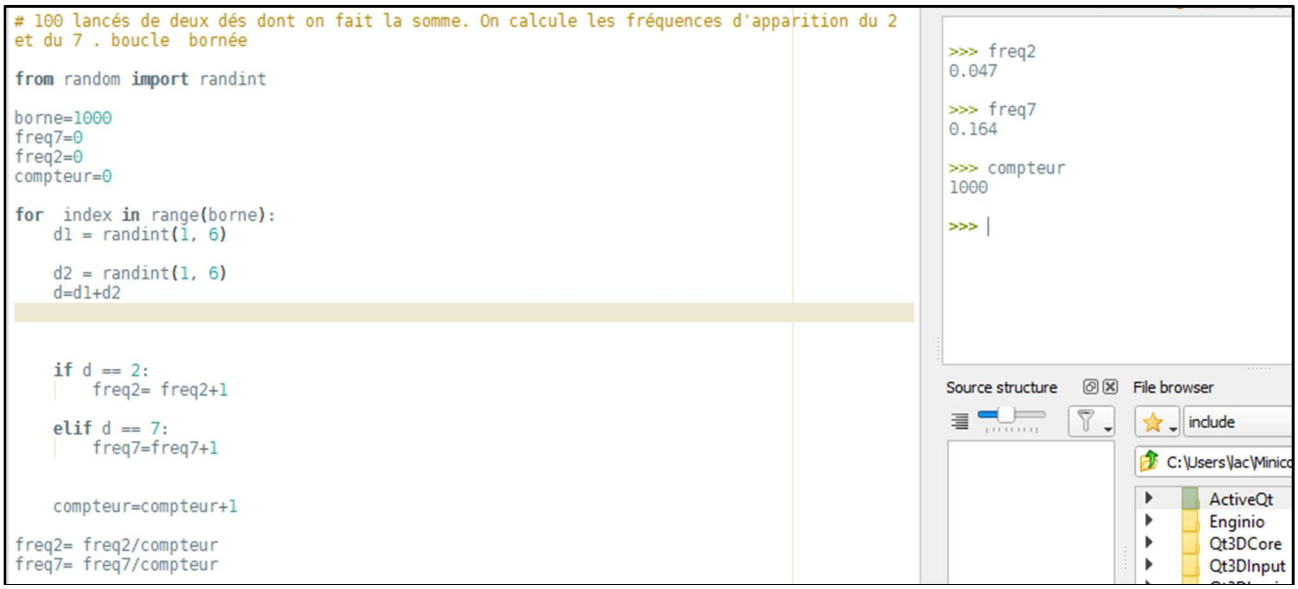
Remarques :

- Si l'index démarre à 0, il n'est pas nécéssaire de l'indiquer et l'instruction devient « ***for i in range(a)*** ».

Exemple 12 :

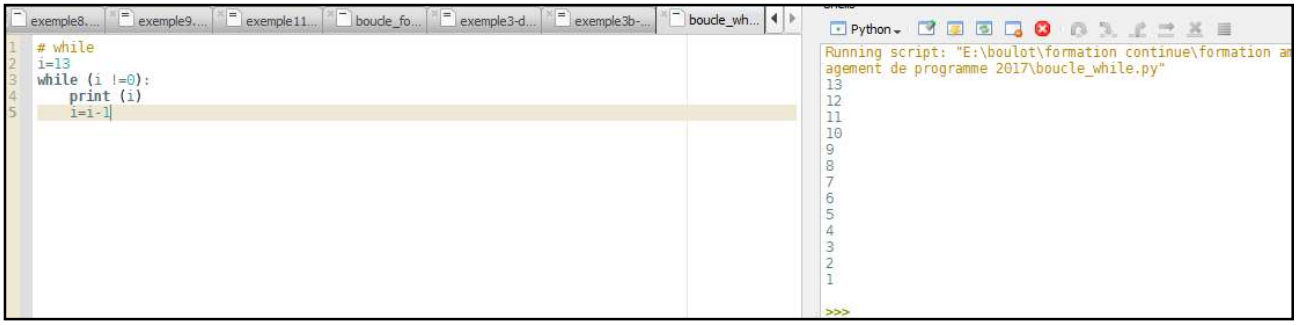
Simuler 1000 lancers de deux dés dont on calcule la somme. Le programme détermine la fréquence d'apparition du 2 et du 7.

Une solution : Recopier puis exécuter le programme ci-dessus



Structures itératives non bornées

L'instruction *« **while** »* est une boucle qui se répète tant que la condition est vérifiée .



Structure du *whille* while (condition1) : instuction1 instruction2 instruction 3	• La portée du while se fait par indentation. • Les instructions 1 et 2 se font dans la boucle. L'instruction 3 s'execute une fois la boucle terminée.

Remarque :

ATTENTION : la boucle while est explosive ! imaginons que l'instruction $\boldsymbol{i = i - 1}$ disparaisse dans l'exemple précédent. La valeur de i n'évoluerais plus et la condition serait donc toujours vérifiées. Le programme deviendrait donc prisonnier de cette boucle et la machine tournerait indéfiniment !

Pour arrêter une boucle infinie, on peut utiliser le bouton « interrompre » du menu de la fenêtre « shell ».

Exemple 13 :

On place 1000€ à un taux de 5% sur une assurance vie. Au bout de combien d'années, la somme aura-t-elle au moins doublé ?

Une solution : Recopier et exécuter le programme ci-dessous.

```
1  #exemple 13
2  depot=1000
3  annee=0
4  while (depot<=2000):
5      depot=depot*1.05
6      annee=annee+1
7  print(annee)
```

3-4 Utilisation des bibliothèques :

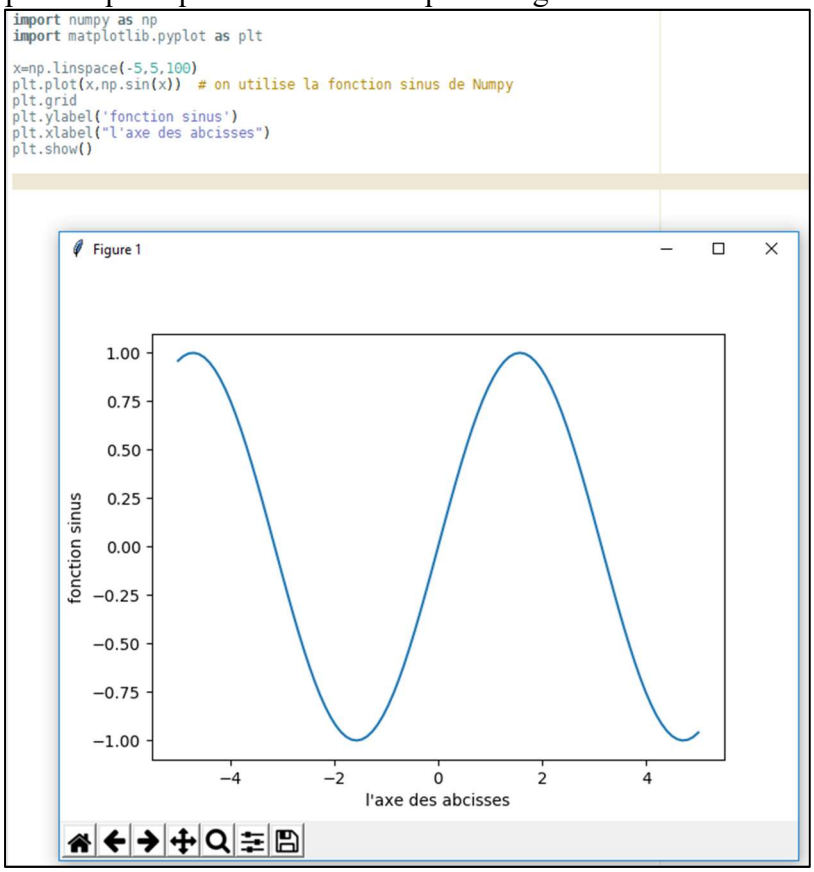
Pour utiliser des fonctions mathématiques, des fonctions aléatoires, obtenir des représentations graphiques, faire du calcul matriciel …. Il est nécessaire d'utiliser des bibliothèques Python. Nous avons vu précédemment que les bibliothèques ***math*** ou ***randint*** étaient intégrées dans Python et qu'il suffisait de les charger en début de programme. C'est différent pour les deux bibliothèques utilisées dans l'exemple ci-dessous :

Numpy et **matplotlib** nécessitent une installation qui est simple dans ***pyzo***. Il suffit d'écrire dans le shell :

```
>>> pip install numpy
>>> pip install matplotlib
```

Dans l'exemple ci-dessous les bibliothèques ***numpy*** et ***mathplotlib*** sont appelées et utilisées. La déclaration faite de la manière suivante (***Import*** *matplotlib.pyplot* ***as*** *plt)* permet d'utiliser toutes les fonctions de la bibliothèque *matplotlib.pyplot* en lui associant le préfixe *plt (plt.plot, plt.grid ….)* . On peut imaginer qu'une autre bibliothèque pourrait utiliser une fonction plot. En la déclarant ainsi, il n'y a pas d'équivoque sur la bibliothèque d'origine.

```
import numpy as np
import matplotlib.pyplot as plt

x=np.linspace(-5,5,100)
plt.plot(x,np.sin(x))  # on utilise la fonction sinus de Numpy
plt.grid
plt.ylabel('fonction sinus')
plt.xlabel("l'axe des abcisses")
plt.show()
```



Recopier puis exécuter le programme ci-dessus.

4- Liens et cours :

https://wiki.python.org/moin/
http://www.courspython.com/
http://python.lycee.free.fr/
https://openclassrooms.com/courses/apprenez-a-programmer-en-python