INF 1 - Introduction à Python et à l'algorithmique

Le langage utilise dans notre classe ECG Mathématiques approfondies est le Python.

Quelques mots sur le langage Python. Python est un langage de programmation sous licence libre très utilisé dans de nombreux domaines (administration de page web, science des données, etc). Puissant et complet, c'est aujourd'hui le langage le plus utilisé. De nombreuses entreprises de technologies l'utilisent pour son efficacité (et sa popularité).

Le Python en prépa ECG. Les sujets de concours à l'écrit contiennent toujours des questions de programmation. C'est aussi souvent le cas dans les épreuves orales pour les concours qui en imposent. Le programme se concentre sur des aspects d'analyse numérique liées au cours de mathématiques : résolution d'équations, simulations de variables aléatoires. Tout au long de l'année, des exercices seront données sur machine mais aussi à l'écrit. Il faudra savoir écrire un petit programme Python sur feuille dans de nombreux devoirs.

Pour installer Python sur votre ordinateur personnel, il faudra télécharger la distribution **Anaconda**.

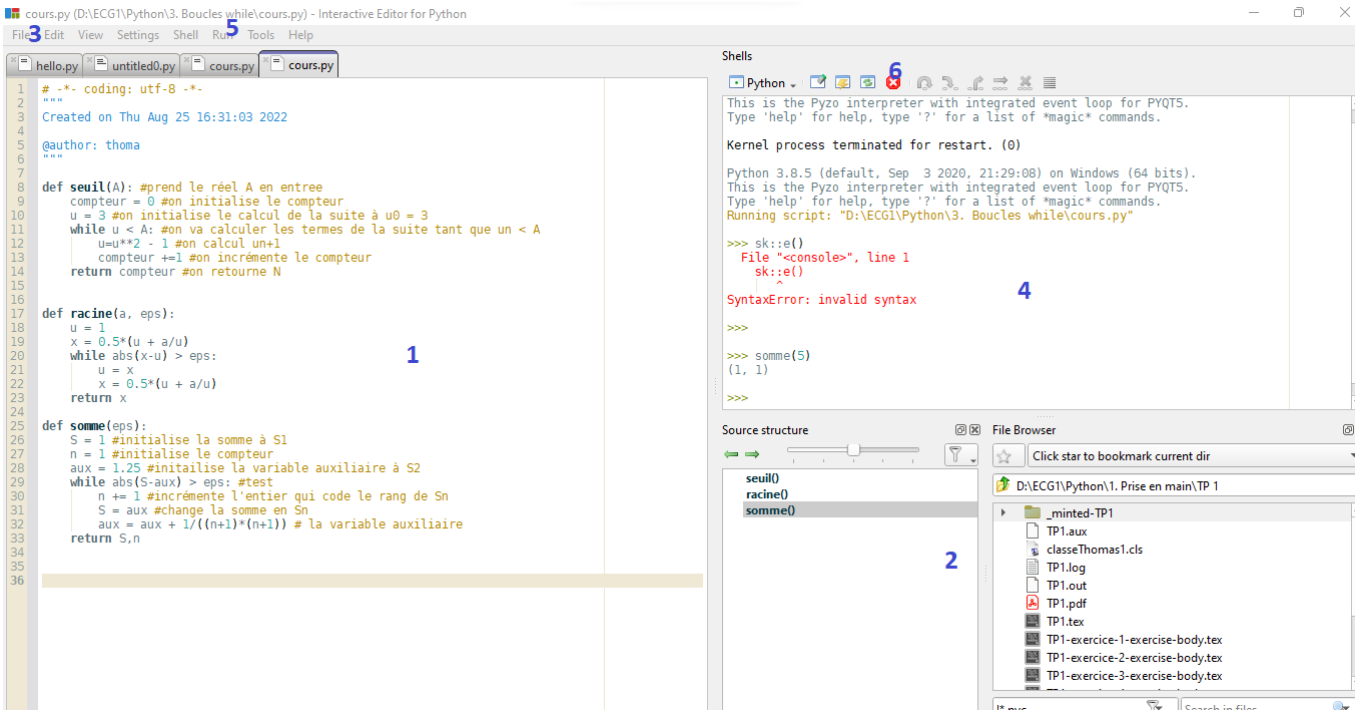
`https://www.anaconda.com/products/distribution`

Je vous recommande aussi d'installer un IDE (environnement de travail) sympathique comme Spyde ou Pyzo. Pyzo est installé sur les ordinateurs du lycée.

`https://www.spyder-ide.org/`

`https://pyzo.org/`

1. PRÉSENTATION DE PYZO



Voici l'écran de Pyzo, on y trouve :

1. l'éditeur de fichier : c'est notre espace principal de travail. C'est ici que nous taperons nos programmes avant de les faire exécuter par la console (que nous verrons plus tard).
2. l'explorateur de fichier, de variables et de tracés correspondant à notre dossier de travail. On l'utilisera surtout quand on travaillera avec plusieurs fichiers à la fois.
3. le menu File (pour fichier) : c'est l'espace d'enregistrement d'ouverture des fichiers sur l'ordinateur.
4. la console (ou shell) : c'est la zone où on tape les commandes et on récupère les résultats. On peut s'en servir comme d'une calculatrice.
5. le bouton Run : il exécute le fichier en cours de lecture dans la console. Il est aussi intéressant de voir ce que font les boutons à cotés.
6. un bouton STOP, en cas d'urgence ou de programme qui ne s'arrête pas.

Quelques conseils de bonne rédaction :

- On donne des noms explicites aux variables, fonctions, etc.

- On commente son code : le # permet de mettre du commentaire dans son code qui ne sera pas compilé.

2. QUELQUES TYPES ET LEURS MÉTHODES

Python est un langage qui accorde de l'importance aux types. De la même façon qu'on objet mathématique peut avoir plusieurs natures (nombre, réels, fonctions ...), un objet informatique peut avoir plusieurs types. En Python, la commande **type** permet de donner le type d'un objet.

2.1. Les nombres

Il y a deux types pour les nombres en Python, les entiers et les flottants. La différence vient de la façon dont ils sont codés en machine, et les méthodes utilisés par Python pour réaliser les différentes opérations.

- le type **int** est le type entier : 5, 6 ,8 ,- 8 ou -12 sont des entiers
- le type **float** est le type flottant : 5.2, 6.6, -5.0, -1.11 sont des flottants.

Python est sympathique avec les nombres : les méthodes d'addition, multiplication, etc seront les mêmes pour les entiers et les flottants. On peut même additionner un entier et un flottant. Attention cependant à la division euclidienne. Un petit résumé des méthodes à connaître est là :

+	addition
−	soustraction
*	multiplication
**	puissance
/	division
//	division euclidienne (type **int**)
%	reste de la division euclidienne

Conversion des nombres. On peut convertir les entiers en flottants à l'aide de `float` et lorsque c'est possible les flottants en entiers avec la commande `int`. Utilisée sur un flottant non entier, la commande `int` revoie la partie entière (au type **int**).

```
1 float(3)
2 int(3.0)
```

2.2. Les booléens

Le type **bool**, ou booléen, peut prendre deux valeurs `True` ou `False`.

On peut réaliser des opérations entre booléens dont les noms sont assez explicite : `and`, `or`, `not`.

and	ET
or	OU
not	NON

On peut aussi créer des booléen en utilisant des expressions mathématiques

==	test d'égalité
!=	test de différence
<=	comparaison
>=	comparaison
<	comparaison
<	comparaison

Attention

Il ne faut pas confondre l'opérateur == avec l'opérateur = qui est l'opérateur d'affectation que l'on va découvrir dans la partie suivante.

2.3. Les uplets

A partir des types déjà connus, on peut créer des uplets, c'est à dire un ensemble ordonnée d'éléments d'un certain type. Par exemple, qu'obtient-on si on tape

```
a = (1,2,3)
```

dans le Shell ? Tapez ensuite

```
type(a)
```

Remarque 2.1 — Certains opérateurs +, * et d'autres sont définis aussi sur les uplets. Vérifiez ce que donne

```
(1,1)+(2,2)
(1,1)*(1,1)
(True,False)*(False,True)
(True,True) and (False, True)
```

Remarque 2.2 — Nous verrons d'autres types, comme les listes ou les chaines de caractère.

3. VARIABLES ET AFFECTATIONS

Affectation de variable. Au cours d'un algorithme, on sera amenés à devoir conserver des valeurs et à les modifier. Pour ce faire, on stocke la valeur dans une **variable**. Une variable, c'est l'association d'un nom (ou expression) à une valeur (qui peut être un entier, un flottant, un booléen, etc).

La commande d'affectation est =.

Par exemple, la commande suivante affecte l'enter 25 à la la variable **résultat**.

```
resultat = 25
```

Modification, incrémentation. Une fois la variable crée, on peut la modifier en utilisant le code suivant.

```
resultat = 20
```

Maintenant la variable résultat stocke l'entier 20. On peut aussi appeler la variable elle même pour la modifier. Par exemple

```
resultat = resultat + 5
```

augmente de 5 l'entier stocké. On aurait aussi taper

```
resultat += 5
```

Affectation simultanée. Une dernière méthode à connaître est l'affectation simultanée : on peut par exemple affecter plusieurs expressions à l'aide d'un uplet de valeur. Par exemple le code

```
(a,b) = (2,3)
```

affecte à a la valeur 2 et à b la valeur 2. Cette méthode permet aussi de récupérer des éléments d'un uplet. Réfléchir par exemple à ce que fait le code

```
uple = (1,2,3)
(a,b,c) = uple
print(b,c)
```

4. UN PREMIER EXEMPLE DE BIBLIOTHÈQUE : NUMPY

Pour de nombreux usages de Python, des fonctions ont déjà été développées et au lieu de les recoder, nous allons les utiliser. Par exemple, hors de question de recoder pi ou la fonction cosinus à chaque fois. La bibliothèque **numpy** est celle qui a été développée pour le calcul numérique. Il faut l'importer avant de pouvoir l'utiliser. Pour l'importer on tape

```
import numpy
```

et pour appeler les commandes de la bibliothèque il faut taper par exemple **numpy.pi** qui renverra pi. Pour alléger les notations, on importera **numpy** sous un autre nom en tapant

```
import numpy as np

np.sqrt(25)
```

Une première liste de fonctions **numpy** à connaître est la suivante

np.exp	exponentielle
np.log	logarithme **népérien**
np.cos	cosinus
np.sin	sinus
np.sqrt	racine carrée
np.abs	valeur absolue
np.floor	partie entière

Numpy est surtout très utile pour la gestion des matrices et des tableaux. Nous nous en servirons beaucoup dans les chapitres suivants.

5. PREMIÈRES FONCTIONS EN PYTHON

Structure d'un programme. Commande `return`. Une fonction en Python est la même chose qu'une fonction en mathématiques : elle reçoit plusieurs variables en entrées, leur fait subir une suite d'instructions, en déduit un résultat aussi appelé sortie. La syntaxe générale d'une fonction est la suivante :

```
def function(var1,var2,var3)
        blocs d instructions
        blocs d instructions
        return sortie
```

La commande `return` est la commande qui désigne la **sortie de la fonction**?

Remarque 5.1 —

1. Dans le code, on voit que c'est l'opérateur **return** qui annonce qu'on renvoie le résultat.
2. Ne pas oublier les deux points!
3. Il faut bien respecter l'indentation (qui est automatique avec Pyzo).
4. Attention, la commande **return** arrête immédiatement la fonction et le reste n'est pas lu.
5. Un programme peut ne rien renvoyer et juste afficher quelque chose. On verra cela un peu plus tard. La commande **return** peut permettre de forcer le programme.

```
   if n == 0: #test de la valeur de n pour les données initiales
      return 1 #renvoi du cas initial
   else: #sinon
      F = 1 #initialisation du resultat
      for k in range(2,n+1): #actions
         F = F * k #actions
      return F #renvoi de F
```

Exemple 1 — Le fonction suivante prend en entrée un nombre et renvoie dix fois ce nombre

```
  def foisdix (x):
    a = 10*x
    return a
```

La commande print. Dans un programme, la commande `print` affiche ce qu'elle prend en argument. Par exemple, dans le code suivant affiche tous les cinq entiers suivant celui donné en argument, avant de renvoyer 1000 (quelque soit l'entrée).

```
def inutile(n):
    print(n+1) #affichage
    print(n+2) #affichage
    print(n+3) #affichage
    print(n+4) #affichage
    print(n+5) #affichage
    return(1000) #sortie de la fonction
```