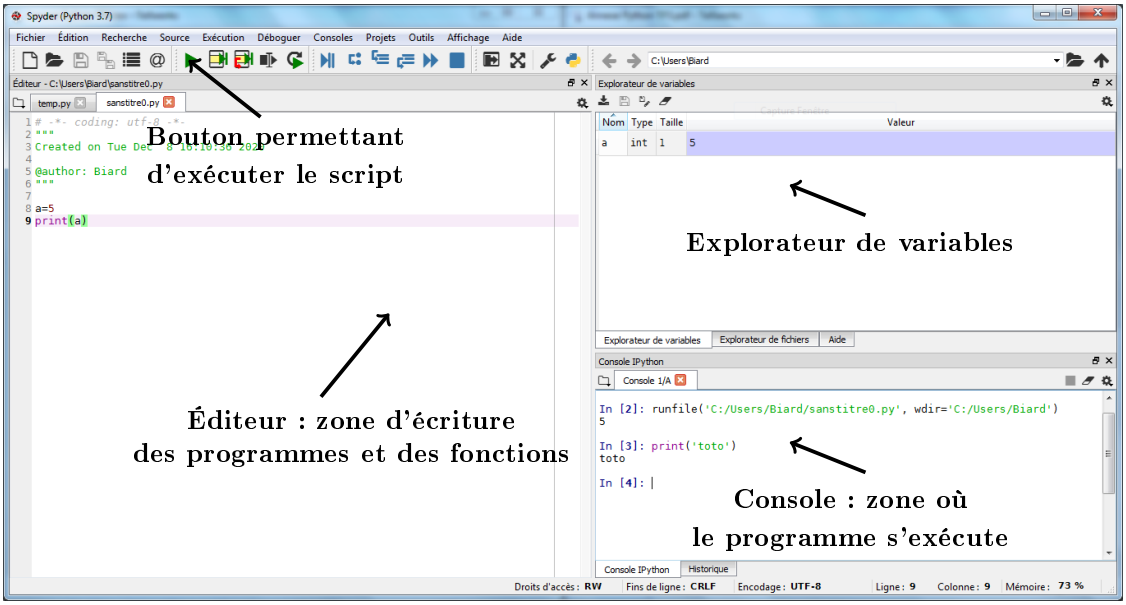
Annexe : Résumé du Langage Python

1- Le Logiciel Spyder et le Langage Python

Python est un langage de programmation qui peut être utilisé dans plusieurs environnement : Edu-Python, Pyso, Spyder, ...

L'Interface du Logiciel Spyder est la suivante :



L'**éditeur** permet d'écrire des programmes. Après l'écriture d'un programme ou d'une fonction, nous l'exécutons avec l'icône ▶. Après l'exécution d'un programme, nous pouvons utiliser la console pour connaître le contenu des variables calculées.

La **console** permet de dialoguer directement avec le programme à l'aide du clavier. L'utilisateur saisit des instructions dans la console, tape sur la touche `entrée` et le programme affiche une donnée à l'écran en réponse. La console est ainsi le lieu où s'affichent les données des programmes écrits dans l'éditeur.

2 - Éléments de Base

Nous utilisons le symbole `=` pour affecter une valeur à une variable, `print()` pour afficher la valeur d'une variable et `input()` pour saisir la valeur d'une variable.

```
prix=150

print(prix)  #affiche : 150
print("Le prix est de ",prix)  #affiche : Le prix est de 150


nom=input("Nom?")  # affiche le texte "Nom?" et attend la saisie
                   # d'une valeur stockée dans la variable nom

age=int(input("Âge?"))  # valeur saisie convertie en entier.

age=float(input("Âge?"))  # valeur saisie convertie en décimal (flottant).
```

Les différents types de variables sont :

- Les **entiers** : `int` (integer).

  Exemple : quotient et reste de la division euclidienne : $5 = 2 * 2 + 1$

```
3*5          #renvoie 15
5//2         #renvoie 2
5%2          #renvoie 1
```

- Les **flottants** : `float` (floating point number), pour représenter les nombres décimaux et de ce fait approximer les réels.

Exemple :

```
5-1.5**2     #renvoie 2.75
8**(13)      #renvoie 549755813888
```

- Les **chaînes** : `str` (character string), à écrire entre guillemets " " ou apostrophes ' '.

Exemple :

```
"Résumé du "+"langage Python"   #renvoie Résumé du langage Python
```

- Les **booléens** : `bool` (boolean), prend la valeur `True` (vrai) ou `False` (faux).
- Exemple :

```
1+1==2       #renvoie True
4/3<5/4      #renvoie False
```

- Les **listes** : `list`, sont une collection d'objets à écrire entre `[]`.

Exemple :

```
[x**2 for x in[1,2,3]]      #renvoie [1,4,9]
```

3- Les Fonctions

La syntaxe en Python est la suivante :

```
def nom_de_la_fonction(parametre1,parametre2,...) :
    instructions
    return resultat
```

(uniquement si la fonction renvoie un résultat)

Exemple :

```
def f(x):
   return x**2+1
```

Pour obtenir un résultat, la fonction doit être appelée dans la console ou par une autre instruction.

```
f(-3)          #renvoie 10
f(2.1)         #renvoie 5.41
```

4 - Tests et Instructions conditionnelles

Les différents opérateurs de comparaison sont :
égalite : `==`, différent : `!=`, `<`, `<=`, `>`, `>=`. et : `and`, ou : `or`, non : `not`.

Nous pouvons écrire un test à **une seule condition** : `if`

Exemple :

```
def divSansErreur(a,b):
   if b!=0:
      return a/b
```

Nous pouvons écrire un test à **deux conditions** : `if ... else`

Exemple :

```
if x>=2:
   y=2*x+1
else:
   y=x**2
```

Nous pouvons écrire un test à **trois conditions** : `if ... elif ... else`

Exemple :

```
if x>0:
   print("positif")
elif x<0:
   print("négatif")
else:
   print("nul")
```

5- Les Boucles

Il y a la boucle bornée ou boucle **Pour** (`for`)
Une boucle `for i in range(n)` s'exécute n fois, le compteur i allant de 0 à $n-1$.

Exemple :
```
for i in range(5):
    print(i**2)
```

avec : `range(n)` sont les entiers de 0 à $n-1$.
`range(n,p)` sont les entiers de n à $p-1$.
`range(n,p,k)` sont les entiers de n à $p-1$ avec un pas de k.

Une boucle `for` permet aussi de parcourir tous les éléments d'une liste ou d'une chaîne.

Exemple :
La fonction suivante compte le nombre d'apparitions d'une lettre dans un mot.
```
def compte(mot,lettre):
    nb=0
    for c in mot:
        if c==lettre:
            nb=nb+1
    return nb
```

Par exemple, `print(compte("bonjour","o"))  #renvoie 2`
La variable c parcourt toutes les lettres du mot.

Il y a également une boucle **non bornée** ou boucle **Tant que** (`while`)
Une boucle `while` s'exécute tant que la condition donnée est vérifiée.
Exemple :
```
i=0
while i<5:
    print(i**2)
    i=i+1
```

Il faut s'assurer qu'une boucle `while` s'arrête.
Nous pouvons remplacer une boucle `for` par une boucle `while` : pour cela, penser à initialiser le compteur(`i=0`) et à l'incrémenter (`i=i+1`)

6 - Les Listes

6-1 Générer une Liste

- en extension :
  La liste vide se note : `[]`
  Créer une liste en indiquant les valeurs : `L=[2,4,6,8]`
  Créer une liste des premiers termes d'une suite arithmétique : `list(range())`

  Exemple :
  ```
  list(range(6))      # [0,1,2,3,4,5]
  ```

- par Ajouts successifs :
  Ajouter un élément à la fin d'une liste : `append()`
  Exemple :
  ```
  M=["a","b","c"]
  M.append("d")
  ```

- Concaténer deux Listes : `+`
  Exemple :
  ```
  ["a","b"]+["d","e"]
  ```

- en compréhension :
  indiquer l'instruction de construction dans la liste
```
P=[x**3 for x in range(4)]
print(P)   #renvoie [0,1,8,27]
```
  Nous pouvons ajouter une condition
```
P=[c for c in "kiara" if c!="a"]   # ['k','i','r']
```

6-2 Manipuler les Éléments d'une Liste

- Ajouter, supprimer

  Insérer un élément à une position donnée : `insert()`
```
M=['a','b',2,3]
M.insert(3,'c')
M  # ['a','b',2,'c',3]
```

  Supprimer la première apparition d'une valeur : `remove()`
```
M.remove(2)
M  # ['a','b','c',3]
```

  Supprimer un terme connaissant son indice :
```
del M[3]
M  # ['a','b','c']
```

- Utiliser les indices

  Parcourir une liste

  de gauche à droite, utiliser des indices positifs : `L[0], L[1], L[2] L[3], ...`

  de droite à gauche, utiliser des indices négatifs :`L[-1], L[-2], L[-3] L[-4], ...`
```
L=["a","b","c","d"]
print(L[0],L(1],L(2],L[3])  #a b c d
print(L[-1],L(-2],L(-3],L[-4])  #d c b a
```

6-3 Longueur d'une Liste
```
len([2,3,5,7])  #4
```

7- Les Bibliothèques de Python

- Certaines fonctions spécifiques au langage Python sont rangées dans des bibliothèques. Pour pouvoir les utiliser, nous pouvons importer entièrement la bibliothèque ou seulement la ou les fonction(s) souhaitée(s).

L'étoile * permet d'importer toutes les fonctions d'une bibliothèque.

Exemple : `from math import *` importe toutes les fonctions de la bibliothèque `math`.

Nous pouvons importer d'une bibliothèque seulement les fonctions dont nous avons besoin.

Exemple : `from math import sqrt` importe la fonction racine carrée de la bibliothèque `math`.

Lorsque le nom de la bibliothèque est très long, nous pouvons lui définir un alias en ajoutant `as` suivi de quelques lettres formant l'alias. Nous pouvons ensuite utiliser toutes les fonctions de la bibliothèque en faisant précéder leur nom de l'alias.

Exemple : `import matplotlib.pyplot as plt`

L'instruction `plt.plot(x,y)` permet d'importer la fonction plot de la bibliothèque mathplotlib.pyplot en utilisant l'alias plt.

Quelques bibliothèques utilisées sont :

- `math` : définit des fonctions mathématiques (`sqrt` pour racine carrée, `sin`, `cos`, ..), ainsi que certaines constantes (`pi`,...)
- `random` (`random`, `randint`, ...) gère l'aléatoire en Python.
- `matplotlib`(`plot`, `show`...) gère le graphisme plan en Python.
- `scipy`(`signal`, ...) gère tout ce qui est calcul scientifique

8- L'Aléatoire en Python

Le module `random` est un module qui regroupe des fonctions permettant de simuler le hasard.
Pour cela, il est nécessaire d'importer la bibliothèque adéquate.

```
import random as rd
n=rd.randint(a,b) # entier choisi au hasard entre a et b compris
a=rd.random() # réel choisi au hasard entre 0 et 1 non compris
b=rd.uniform(a,b) # réel choisi au hasard entre a et b compris
```

9- Les Tableaux

La bibliothèque `numpy` est importée dans Python avec la syntaxe :
`import numpy as np # bibliothèque numpy renommée en np`
On renomme très souvent la bibliothèque `numpy` en `np` pour raccourcir l'écriture des programmes Python.

```
tab1=np.array([8, -5, -2.1, 7]) # création d'un tableau tab1 à 4 éléments

tab2=np.array([]) # création d'un tableau tab2 vide
tab2=np.append(tab2,'test') # ajout de l'élément 'test' dans le tableau tab2

tab3=np.zeros(N) # création d'un tableau tab3 contenant N valeurs égales à 0
```

Comme pour les listes, on peut modifier la valeur d'un élément d'indice `i` du tableau `tab3` avec la syntaxe `tab3[i]`.

La syntaxe `np.arange(2, 9.1, 0.5)` permet de créer un tableau dont les valeurs sont comprises entre 2 inclus et 9,1 exclu avec un pas égal à 0,5.

La fonction `linspace(mini, maxi, nbr_pts)` permet de créer un tableau contenant `nbr_pts` éléments régulièrement espacés avec des valeurs comprises entre mini inclus et maxi inclus.

```
T=np.linspace(0, 1.5, 10) # Tableau T de 10 valeurs entre 0 inclus et 1,5 inclus
```