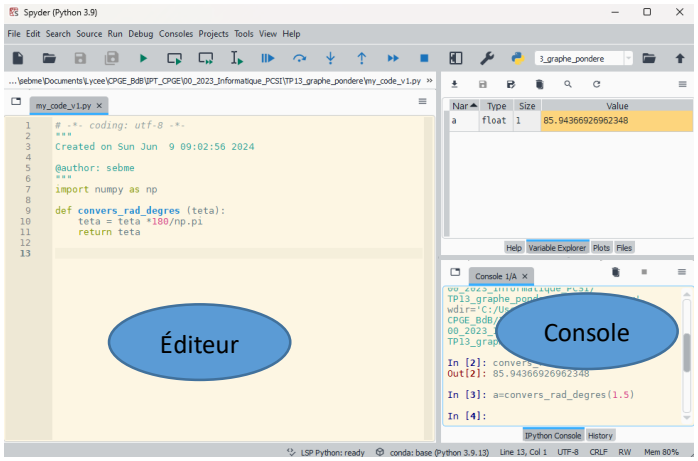
TP1 – Prise en main du langage PYTHON

Au cours de ces deux prochaines années, nous allons utiliser avec le langage de programmation **Python** dans un environnement développement intégré (IDE) **SPYDER**.

Étape 0. Dans votre espace personnel, créer un dossier *INFO_PCSI*. Dans ce dossier, créer un sous-dossier *TP1* : c'est là que vous enregistrerez les fichiers relatifs à ce TP.

1 Python en interactif



Lorsque vous ouvrez l'IDE « SPYDER », vous obtenez une plusieurs fenêtres dont :

- ✓ Éditeur : Cette fenêtre permet d'écrire nos codes.
- ✓ Console (ou interpréteur ou terminale ou shell) :
  Cette fenêtre permet d'exécuter nos codes et d'afficher les résultats.
  Elle permet aussi aussi d'utiliser Python en mode interactif. Les expressions sont rédigées à la suite de **In [1] :** puis évaluées avec la commande Entrée
- ✓ Un fenêtre qui permet :
  - d'obtenir de l'aide (help)
  - Voir les différentes varaibles
  - Voir les tracés graphiques.
  - Voir les fichiers présents dans le répertoire courant.

Exercice 1. **Calculer avec Python**

1. Ouvrir SPYDER puis réaliser les opérations suivantes dans la console.
2. $50 + 3 \times (12{,}5 - 31) =$ ..... $\frac{3}{2} =$ ..... $\frac{2}{3} =$ ..... $2^5 =$ .....
3. Évaluer les expressions suivantes : 23//3 , 23 % 3 , abs(1.3) et abs(-1.3).

Compléter le tableau ci-dessous :

Opération	Opérateur	Opération	Opérateur
Somme		Produit	
Différence		Division numérique	
	//		%
Puissance		Valeur absolue	

Les fonctions mathématiques usuelles ne sont pas disponibles immédiatement à l'ouverture de SPYDER, il faut charger une librairie (ou module) complémentaire : le module math ou numpy. Nous verrons d'autres modules importants au cours de l'année.

Exercice 2. Exécuter les commandes ci-dessous.

```
In[1] : sin(pi)
In[2] : import numpy as np
In[3] : np.sin(np.pi)
```

La commande import donne accès à une nouvelle librairie. Dans ce cas, il faut écrire **np.** avant chaque objet de la bibliothèque que nous voulons utiliser.

Exercice 3. Calculer (des valeurs approchées) des expressions suivantes.

$\ln(10) =$ ..... $\cos\left(\frac{\pi}{8}\right) =$ ..... $\frac{\sqrt{2+\sqrt{2}}}{2} =$ .....

2 Variables et affectations

Pour manipuler des données, il peut être utile de manipuler des *variables* qui représentent ces données.

Les objets manipulés par Python, au moment où ils sont créés, sont stockés dans la mémoire de l'ordinateur. Un *nom de variable* est une suite de caractères qui renvoie à une adresse mémoire où a été créé un objet.

```
In[1] : x = 1
In[2] : x
Out[2] : 1
```

Pour affecter (ou assigner) une valeur à une variable on utilise le signe *égal*.
La commande ci-contre signifie que désormais le caractère x renvoie à la valeur 1.
Le signe *égal* doit être distingué de l'égalité mathématique. On peut écrire en *pseudo-code* : $x \leftarrow 1$

(Sous Python, on teste une égalité avec le symbole == Par exemple, x==2)

Exercice 4. Examiner la série de commandes ci-dessous. Prédire le résultat puis confirmer-le à l'aide de la console de SPYDER

```
In[1] : x = 1
In[2] : y = 2
In[3] : x = x + y
In[4] : y = x**y
In[5] : y
Out[5] : ????
```

Règle. Les noms de variables autorisés sont des séquences de lettres (majuscules ou miniscules) et de chiffres qui débute toujours par une lettre. La *casse* est significative. Il existe quelques mots *réservés* ne pouvant faire office de nom de variable (def, if, while, True, False...).

La fonction print() permet également d'afficher la valeur d'une ou plusieurs variables séparées par des virgules.

```
In[1] : x = 1
In[2] : y = 1.5
In[3] : print(x,y)
Out[3]: 1 1.5
```

Affectations multiples. Le langage Pyhton offre la possibilité de réaliser en une seule instruction plusieurs affectations.

Affectations successives. La commande ci-contre signifie que **x** représente la valeur 1 et y représente la valeur x (donc 1).

```
In[1] : x=y= 1
In[2] : y
Out[2]: 1
In[3] : x
Out[3]: 1
```

Affectations parallèle. La commande ci-contre signifie que **a** représente la valeur 1 et b représente la valeur 2.

```
In[1] : a,b= 1,2
In[2] : a
Out[2]: 1
In[3] : b
Out[3]: 2
```

Remarque. La commande ci-dessous permet d'échanger deux noms de variables.

```
In[1] : a,b= b,a
```

Exercice 5. Examiner la série de commandes ci-dessous. Prédire le résultat puis confirmer-le à l'aide de la console.

```
In[1] : a,b= 1,2
In[2] : a,b= b,a+b
In[3] : c = a**b
In[4] : print(c)
Out[4] : ????
```

3 Premier programme Python

L'utilisation de Python en ligne de commande dans la console ne permet pas de sauvegarder vos lignes de calcul. Ce n'est pas très pratique si vous souhaitez relancer une même série de calculs avec des valeurs différentes pour les variables.

Pour conserver nos programmes on va donc sauvegarder les lignes de code dans un fichier texte : on parlera d'un *script* Python.

1. Ouvrir une nouvelle fenêtre : File → New Window.
2. Écrire et enregistrer le script. Le nom du fichier doit se terminer par l'extension **.py**.
3. Exécuter le script : Run → Run ▶ ou en cliquant directement sur le bouton ▶ ou en cliquant sur F5.
4. Le résultat est donné dans la console.

ATTENTION : Le répertoire de travail courant doit contenir les données (codes, fichiers sources, …) que vous souhaitez utiliser sinon, il faut indiquer le chemin dans l'arborescence de votre ordinateur qui permettra à Python d'accéder aux données.

Exercice 6. On souhaite rédiger un programme élémentaire qui calcule le prix d'une commande de livres. Les trois valeurs suivantes sont représentées par les variables :

nbr=entier désignant le nombre de livres commandés.
prix=prix unitaire d'un livre.
reduc =coefficient (entre 0 et 1) représentant la réduction dont bénéficie le client.

Le programme affiche le montant de la facture m = nbr * prix * reduc.

1. Ouvrir un nouveau fichier que vous sauvegardez sous le nom de `factureLivres.py` dans votre dossier TP1. Votre dossier TP1 doit être le répertoire de travail courant !
2. Affecter les variables comme indiqué ci-dessus dans le cas d'une commande de 27 livres dont le prix unitaire est de 30,5 euros pour un client bénéficiant de 5 % de réduction.
3. Afficher le résultat dans l'interpréteur. Pour afficher un résultat il faut faire appel à la fonction print(). Pour afficher une phrase on peut utiliser par exemple la syntaxe :

```
print('Le montant de la commande est de',m,'euros').
```

Pour aller plus loin : vous afficherez la valeur tronquée de **m** à deux chiffres après la virgule.

Remarques.

1. Vous pouvez afficher des commentaires dans votre script en les rédigeant précédés d'un symbole **#**; ils ne seront pas exécutés.
2. La fonction **`input()`** permet d'interagir avec l'utilisateur : elle interrompt le programme jusqu'à ce que l'utilisateur rentre une valeur et appuie sur *Entrée*. Cela nous impose d'en apprendre un peu plus sur les *types* des variables...

4 Premiers types simples

Dans les parties précédentes nous avons manipulé des objets de nature numérique (entiers, décimaux). Pour désigner la nature d'un objet manipulé par Python, on parle de **type**. Voici les types les plus simples :

1. int : les entiers relatifs
2. float : les nombres en virgule flottante.
3. str : les chaines de caractères.
   Une *chaine de caractères* est une séquence de caractères les uns à la suite des autres. Pour les définir il faut les écrire entre *quotes* (simples ou doubles). Les éléments d'une chaine sont numérotés à partir de 0. Pour extraire un caractère, il suffit d'accoler à la chaine (ou la variable qui la représente) son indice entre crochet.
   Si cela a un sens, les commandes **`int`** et **`float`** convertissent une chaine de caractères en une donnée numérique.

```
In[1] : message = 'bonjour'
In[2] : print(message)
bonjour
In[3] : message[1]
Out[3]: 'o'
```

Voici une liste non exhaustive d'opérations autorisées sur les chaines de caractères.

Opération	Opérateur	Opération	Opérateur
Concaténation	+	Répétition	*
Extraction de fragment	*Chaine*[n:m]	Longueur	len(*Chaine*)

4. bool : les booléens. Il s'agit d'expressions dont la valeur est True ou False.

```
In[1] : 1>0
True
In[2] : 1==0
False
```

Les booléens peuvent être construits avec les opérateurs suivants :

Opération	Opérateur	Opération	Opérateur
Égal	==	Différent	!=
Strictement supérieur	>	Strictement inférieur	<
Supérieur ou égal	>=	Inférieur ou égal	<=
ou	or	et	and
non	not		

La fonction `bool()` évalue l'argument comme un booléen selon la règle suivante : si la donnée est booléenne elle retourne sa valeur, si la donnée est 1 elle retourne True sinon elle retourne False

Exercice 7. Prédire la valeur des booléens suivants et confirmer votre résultat avec l'interpréteur Python.

A = not(1>0 and 0>1) ..... B = 1!=0 or 1==1 .....

5. tuples : les *tuples*. Il s'agit de séquences d'objet séparés par des virgules. Pour extraire un objet, il suffit d'accoler à la chaine (ou la variable qui la représente) son indice entre crochet (on commence à 0).

```
In[1] : x=('coucou',1,1>0)
In[2] : x[2]
Out[2] : True
```

La fonction `type()` permet d'obtenir le type d'un objet.

Sous Python, il n'est pas nécessaire de définir le type d'une variable avant de l'utiliser : on dit que le langage est à *typage dynamique*.

Exercice 8. Reprendre l'exercice 6 en utilisant la fonction input pour interroger l'utilisateur et lui retourner le montant de la facture.

Indication. La fonction `imput()` récupère les entrées de l'utilisateur sous forme de *chaine de caractère*.

```
In[1] : X = input('Quel âge avez-vous ?')
Quel âge avez-vous ?   17
In[2] : X,type(X)
Out[3]: ('17',str)
```

Exercice 09.

Réaliser un programme `horaire.py` qui demande à l'utilisateur : l'horaire de départ de son train, l'horaire d'arrivée de son train et qui affiche la durée du trajet en minutes.

On suppose que l'utilisateur rentre les horaires sous le format : XXhYY (exemples : 19h51, 08h01).
Tester votre programme pour les entrées : départ 17h51; arrivée 19h38. Dans votre console, votre résultat ressemble à ça :

```
In[1] :
Horaire de départ ? 17h51
Horaire d'arrivée ? 19h38
Votre trajet dure 107 minutes
```

Quels sont les inconvénients de votre programme ?