Semestre: S1

Unité d'enseignement: UET 1.1.1

Matière : Programmation avancée en Python

VHS: 45h00 (Cours 1h30, TP 1h30)

Crédits: 2

Coefficient: 2

Cours :

Chapitre 1 : Rappels sur la programmation en Python

Chapitre 2 : Programmation et automatisation

Chapitre 3 : Apprentissage avancé d'Excel

Chapitre 4 : Apprentissage de GanttProject

Chapitre 4 : Programmation orientée objet avancée

Chapitre 5 : Introduction aux données pour l'IA

Travaux pratiques :

TP 01 : Maîtriser les bases de la programmation en Python

TP 02 : d'automatisation de tâches avec Python

TP 03 : Programmation ex Excel

TP 04 : organiser une réunion en Ganttproject

TP 05 : Structures avancées et organisation du code

TP 06 : Programmation orientée objet avancée en Python

TP 07 : Manipulation de fichiers et analyse de données

TP 08 : Préparation et traitement de données pour l'intelligence artificielle

Projet final :

Titre : Analyse et visualisation d'un jeu de données + modèle prédictif simple

Mode d'évaluation :

Examen 60% , CC=40%

Programme, programmation, langage de programmation, logiciel (software), matériel (hardware) :

Un programme est un ensemble d'instructions que l'ordinateur exécute afin d'accomplir une tâche. La programmation est le processus de production d'un programme. Un programme est exprimé à l'aide d'un langage de programmation. Il existe une grande variété de langages de programmation qui ont été développés tels que Java, Python, C, C++, C#, JavaScript, Swift, PHP, Scala, Go, R, etc.Le terme logiciel (software) désigne l'ensemble des programmes, par opposition au matériel (hardware) de l'ordinateur qui correspond à l'ensemble des composantes physiques.

Présentation du langage Python :

Python est un langage portable, dynamique, extensible, gratuit, qui permet (sans l'imposer) une approche modulaire et orientée objet de la programmation. Python est développé depuis 1989 par Guido van Rossum et de nombreux contributeurs bénévoles.

Python est **dynamique** (l'interpréteur peut évaluer des chaînes de caractères représentant des expressions ou des instructions Python).

Python est **extensible** : comme *Tcl* ou *Guile*, on peut facilement l'interfacer avec des bibliothèques C existantes. On peut aussi s'en servir comme d'un langage d'extension pour des systèmes logiciels complexes.

Un logiciel libre (Free Software) est avant tout un logiciel dont le code source est accessible à tous (Open Source). Souvent gratuit (ou presque), copiable et modifiable librement au gré de son acquéreur, il est généralement le produit de la collaboration bénévole de centaines de développeurs enthousiastes dispersés dans le monde entier. Son code source étant « épluché » par de très nombreux spécialistes (étudiants et professeurs universitaires), un logiciel libre se caractérise la plupart du temps par un très haut niveau de qualité technique. Le plus célèbre des logiciels libres est le système d'exploitation GNU/Linux, dont la popularité ne cesse de s'accroître de jour en jour.

Une approche modulaire est une méthode qui décompose un système complexe en éléments plus petits, indépendants et réutilisables appelés modules, afin de faciliter la gestion, le développement, la maintenance et l'évolutivité. Cette approche est applicable dans de nombreux domaines comme l'ingénierie, l'informatique (logiciels, microservices), la construction, l'enseignement ou même l'analyse de discours.

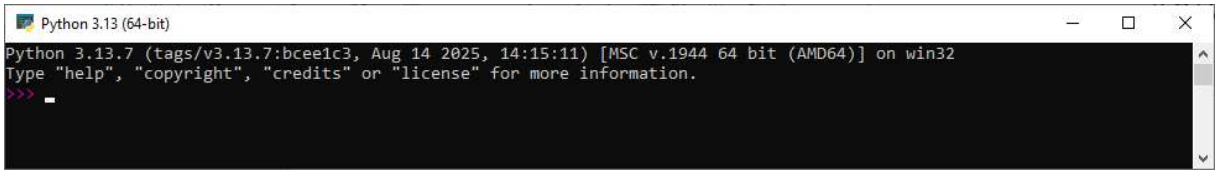
La programmation orientée objet (POO) est un paradigme de programmation qui structure le code autour d'objets, des entités autonomes qui regroupent des données (attributs) et des méthodes (fonctions) agissant sur ces données. Cette approche favorise la création de code modulaire, réutilisable et plus facile à maintenir, en organisant les programmes en ensembles d'objets qui interagissent entre eux.

Guido van Rossum, né le 31 janvier 1956 à La Haye aux Pays-Bas, est un développeur connu pour être le créateur et leader du projet du langage de programmation Python.

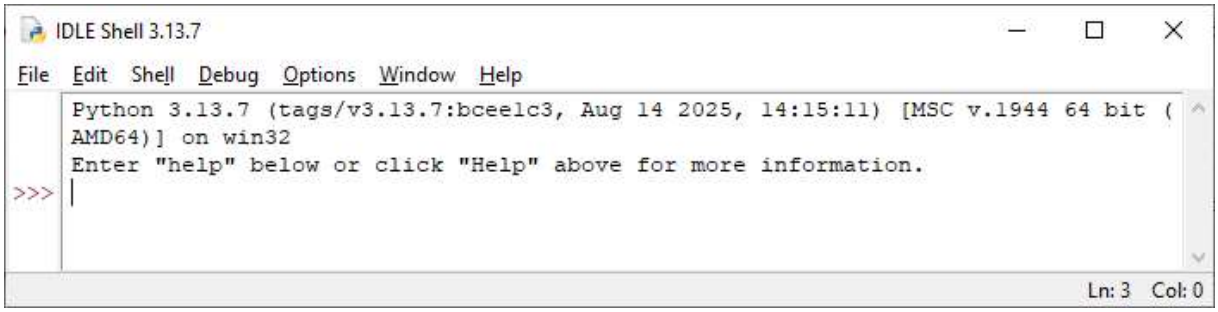
Calculer avec Python

Python présente la particularité de pouvoir être utilisé de plusieurs manières différentes. Vous allez d'abord l'utiliser en mode interactif, c'est-à-dire de manière à dialoguer avec lui directement depuis le clavier. Cela vous permettra de découvrir très vite un grand nombre de fonctionnalités du langage. Dans un second temps, vous apprendrez comment créer vos premiers programmes (scripts) et les sauvegarder sur disque.

L'interpréteur peut être lancé directement depuis la ligne de commande dans une fenêtre DOS sous Windows) : il suffit d'y taper la commande python3.



Si vous utilisez une interface graphique telle que Windows, vous préférerez vraisemblablement travailler dans une « fenêtre de terminal », ou encore dans un environnement de travail spécialisé tel que IDLE. Voici par exemple ce qui apparaît dans une fenêtre de terminal Windows:



Noms de variables et mots réservés

Les noms de variables sont des noms que vous choisissez vous-même assez librement. Efforcez-vous cependant de bien les choisir : de préférence assez courts, mais aussi explicites que possible, de manière à exprimer clairement ce que la variable est censée contenir. Par exemple, des noms de variables tels que altitude, altit ou alt conviennent mieux que x pour exprimer une altitude.

Un bon programmeur doit veiller à ce que ses lignes d'instructions soient faciles à lire.

Sous Python, les noms de variables doivent en outre obéir à quelques règles simples :

• Un nom de variable est une séquence de lettres (**a** → **z** , **A** → **Z**) et de chiffres (**0** → **9**), qui doit toujours commencer par une lettre.

• Seules les lettres ordinaires sont autorisées. Les lettres accentuées, les cédilles, les espaces, les caractères spéciaux tels que **$**, **#**, **@**, etc. sont interdits, à l'exception du caractère **_** (souligné).

• La casse est significative (les caractères majuscules et minuscules sont distingués).

Attention : Joseph, joseph, JOSEPH sont donc des variables différentes. Soyez attentifs !

Prenez l'habitude d'écrire l'essentiel des noms de variables en caractères minuscules (y compris la première lettre7). Il s'agit d'une simple convention, mais elle est largement respectée. N'utilisez les majuscules qu'à l'intérieur même du nom, pour en augmenter éventuellement la lisibilité, comme dans tableDesMatieres.

En plus de ces règles, il faut encore ajouter que vous ne pouvez pas utiliser comme nom de variables les 33 « mots réservés » ci-dessous (ils sont utilisés par le langage lui-même) :

and	**as**	**assert**	**break**	**class**	**continue**	**Def**
del	**elif**	**else**	**except**	**False**	**finally**	**For**
from	**global**	**if**	**import**	**in**	**is**	**lambda**
None	**nonlocal**	**not**	**or**	**pass**	**raise**	**return**
True	**try**	**while**	**with**	**yield**		

Affectation (ou assignation)

Nous savons désormais comment choisir judicieusement un nom de variable. Voyons à présent comment nous pouvons définir une variable et lui affecter une valeur. Les termes « affecter une valeur » ou « assigner une valeur » à une variable sont équivalents. Ils désignent l'opération par laquelle on établit un lien entre le nom de la variable et sa valeur (son contenu).

En Python comme dans de nombreux autres langages, l'opération d'affectation est représentée par le signe égale :

>>> n = 7	**n ← 7**	**# définir la variable n et lui donner la valeur 7**
>>> msg = "Quoi de neuf ?"	**msg ← "Quoi de neuf ?"**	**# affecter la valeur "Quoi de neuf ?" à msg**

Les deux instructions d'affectation ci-dessus ont eu pour effet chacune de réaliser plusieurs opérations dans la mémoire de l'ordinateur :

• créer et mémoriser un **nom de variable** ;

• lui attribuer un **type** bien déterminé (ce point sera explicité à la page suivante) ;

• créer et mémoriser une **valeur** particulière ;

• établir un **lien** entre le nom de la variable et l'emplacement mémoire de la valeur correspondante.

Afficher la valeur d'une variable

Pour afficher leur valeur à l'écran, il existe deux possibilités. La première consiste à entrer au clavier le nom de la variable, puis <Enter>. Python répond en affichant la valeur correspondante :

```
>>> n
7
>>> msg
'Quoi de neuf ?'
```

Il s'agit cependant là d'une fonctionnalité secondaire de l'interpréteur, qui est destinée à vous faciliter la vie lorsque vous faites de simples exercices à la ligne de commande. À l'intérieur d'un programme, vous utiliserez toujours la fonction **print()** :

```
>>> print(msg)
Quoi de neuf ?
 >>> print(n)
7
```

Remarquez la subtile différence dans les affichages obtenus avec chacune des deux méthodes. La fonction **print()** n'affiche strictement que la valeur de la variable, telle qu'elle a été encodée, alors que l'autre méthode (celle qui consiste à entrer seulement le nom de la variable) affiche aussi des apostrophes afin de vous rappeler que la variable traitée est du type « chaîne de caractères ».

Typage des variables :

Sous Python, il n'est pas nécessaire d'écrire des lignes de programme spécifiques pour définir le type des variables avant de pouvoir les utiliser. Il vous suffit en effet d'assigner une valeur à un nom de variable pour que celle-ci soit *automatiquement créée avec le type qui correspond au mieux à la valeur fournie*. Dans l'exercice précédent, par exemple, les variables **n**, **msg** et **pi** ont été créées automatiquement chacune avec un type différent (« nombre entier » pour **n**, « chaîne de caractères » pour **msg**, « nombre à virgule flottante » (ou « *float* », en anglais) pour **pi**).

Affectations multiples :

Sous Python, on peut assigner une valeur à plusieurs variables simultanément. Exemple :

```
>>> x = y = 7
 >>> x
7
 >>> y
 7
```

On peut aussi effectuer des *affectations parallèles* à l'aide d'un seul opérateur :

```
>>> a, b = 4, 8.33
 >>> a
4
>>> b
 8.33
```

Exercices :

2.1 Décrivez le plus clairement et le plus complètement possible ce qui se passe à chacune des trois lignes de l'exemple ci-dessous :

```
>>> largeur = 20
 >>> hauteur = 5 * 9.3
>>> largeur * hauteur 930
```

2.2 Assignez les valeurs respectives 3, 5, 7 à trois variables a, b, c.
Effectuez l'opération a-b//c. Interprétez le résultat obtenu.

Opérateurs et expressions :

On manipule les valeurs et les variables qui les référencent en les combinant avec des *opérateurs* pour former des *expressions*. Exemple :

```
a, b = 7.3, 12
y = 3*a + b/5
```

Dans cet exemple, nous commençons par affecter aux variables **a** et **b** les valeurs **7,3** et **12**.

Comme déjà expliqué précédemment, Python assigne automatiquement le type « réel » à la variable **a**, et le type « entier » à la variable **b**.
La seconde ligne de l'exemple consiste à affecter à une nouvelle variable **y** le résultat d'une *expression* qui combine les *opérateurs* * , + et / avec les *opérandes* **a**, **b**, **3** et **5**.
Les opérateurs sont les symboles spéciaux utilisés pour représenter des opérations mathématiques simples, telles l'addition ou la multiplication. Les opérandes sont les valeurs combinées à l'aide des opérateurs.
Python évalue chaque expression qu'on lui soumet, aussi compliquée soit-elle, et le résultat de cette évaluation est toujours lui-même une valeur. À cette valeur, il attribue automatiquement un type, lequel dépend de ce qu'il y a dans l'expression. Dans l'exemple ci-dessus, **y** sera du type réel, parce que l'expression évaluée pour déterminer sa valeur contient elle-même au moins un réel
Les opérateurs Python ne sont pas seulement les quatre opérateurs mathématiques de base. Nous avons déjà signalé l'existence de l'opérateur de division entière //. Il faut encore ajouter l'opérateur ** pour l'exponentiation, ainsi qu'un certain nombre d'opérateurs logiques, des opérateurs agissant sur les chaînes de caractères, des opérateurs effectuant des tests d'identité ou d'appartenance, etc. Nous reparlerons de tout cela au fil des pages suivantes.

Signalons au passage la disponibilité de l'opérateur modulo, représenté par le caractère typographique **%**. Cet opérateur fournit le reste de la division entière d'un nombre par un autre. Essayez par exemple :

```
>>> 10 % 3        # (et prenez note de ce qui se passe !)
>>> 10 % 5
```

Cet opérateur vous sera très utile plus loin, notamment pour tester si un nombre **a** est divisible par un nombre **b**. Il suffira en effet de vérifier que **a % b** donne un résultat égal à zéro.

Exercice

2.3 Testez les lignes d'instructions suivantes. Décrivez ce qui se passe :

```
>>> r , pi = 12, 3.14159
>>> s = pi * r**2
 >>> print(s)
>>> print(type(r), type(pi), type(s))
```

Quelle est, à votre avis, l'utilité de la *fonction* **type()** ?

Priorité des opérations

Lorsqu'il y a plus d'un opérateur dans une expression, l'ordre dans lequel les opérations doivent être effectuées dépend de *règles de priorité*. Sous Python, les règles de priorité sont les mêmes que celles qui vous ont été enseignées au cours de mathématique. Vous pouvez les mémoriser aisément à l'aide d'un « truc » mnémotechnique, l'acronyme *PEMDAS* :

• *P* pour *parenthèses*. Ce sont elles qui ont la plus haute priorité. Elles vous permettent donc de « forcer » l'évaluation d'une expression dans l'ordre que vous voulez. Ainsi **2*(3-1) = 4** , et **(1+1)**(5-2) = 8**.

• *E* pour *exposants*. Les exposants sont évalués ensuite, avant les autres opérations.

Ainsi **2**1+1 = 3** (et non 4), et **3*1**10 = 3** (et non 59049 !).

M et *D* pour *multiplication* et *division*, qui ont la même priorité. Elles sont évaluées avant *l'addition A*

et *la soustraction S*, lesquelles sont donc effectuées en dernier lieu. Ainsi **2*3-1 = 5** (plutôt que 4), et **2/3-1 = -0.3333...** (plutôt que 1.0).

• Si deux opérateurs ont la même priorité, l'évaluation est effectuée de gauche à droite.

Ainsi dans l'expression **59*100//60**, la multiplication est effectuée en premier, et la machine doit donc ensuite effectuer 5900//60, ce qui donne 98. Si la division était effectuée en premier, le résultat serait 59 (rappelez-vous ici que l'opérateur // effectue une division entière, et vérifiez en effectuant 59*(100//60)).