L. n°1

Introduction au langage *Python*

I. Présentation du langage *Python*

Historique : À la fin des années 1980, le programmeur **Guido van Rossum**, participe au développement du langage de programmation ***ABC*** au *Centrum voor Wiskunde en Informatica* (CWI) d'Amsterdam, aux Pays-Bas.

En 1989, profitant d'une semaine de vacances durant les fêtes de Noël, il utilise son ordinateur personnel pour écrire la première version du langage. Fan de la série télévisée *Monty Python's Flying Circus*, il décide de baptiser ce projet ***Python***. Il s'est principalement inspiré d'***ABC***, par exemple pour l'indentation comme syntaxe.

La première version de ***Python*** est sortie en 1991. Ce langage a voyagé du Macintosh de son créateur jusqu'à se voir associé à une organisation à but non lucratif, la Python Software Foundation, créée en 2001.

Le langage Python est :

- **entièrement gratuit**
- **portable** : un même programme s'exécute sur Linux, Windows, Mac Os . . .
- **interprété** : les instructions envoyées sont transcrites en langage machine au fur et à mesure de leur lecture.

  D'autres langages (C ou C++) sont des langages de programmation compilés car, avant de pouvoir les exécuter, un logiciel tiers doit se charger de transformer le code du programme en langage machine : c'est l'étape de la compilation.
- **orienté objet**
- **de haut niveau**
  - la syntaxe permet de programmer sans tenir compte des détails inhérents au fonctionnement de l'ordinateur.
  - Python possède un garbage collector, destruction automatique des objets créés lorsqu'ils ne sont plus utilisés.
  - structures de données complexes telles que des dictionnaires, éloignées des types numériques standards.
- **modulaire** : la définition du langage est très succincte et autour de ce noyau concis, de nombreuses librairies ou modules ont été développées.
- **à syntaxe positionnelle** : l'indentation fait partie du langage.

Inconvénients	Avantages
• vitesse d'exécution plus lente que le langage C++ • moins utilisé que le C++ ou le Java	• syntaxe plus simple que celle de Java ou du C++ ○ langage plus simple à apprendre ○ amélioration significative des temps de développement • pas de déclaration de types, de variables . . . • le code en python est : ○ 3 à 5 fois plus court que le code Java équivalent ○ 3 à 10 fois plus court que le code C++ équivalent

Python est un langage puissant, assez facile à apprendre (par rapport à d'autres langages) et très riche en possibilités. Maîtriser la langage Python c'est pouvoir :

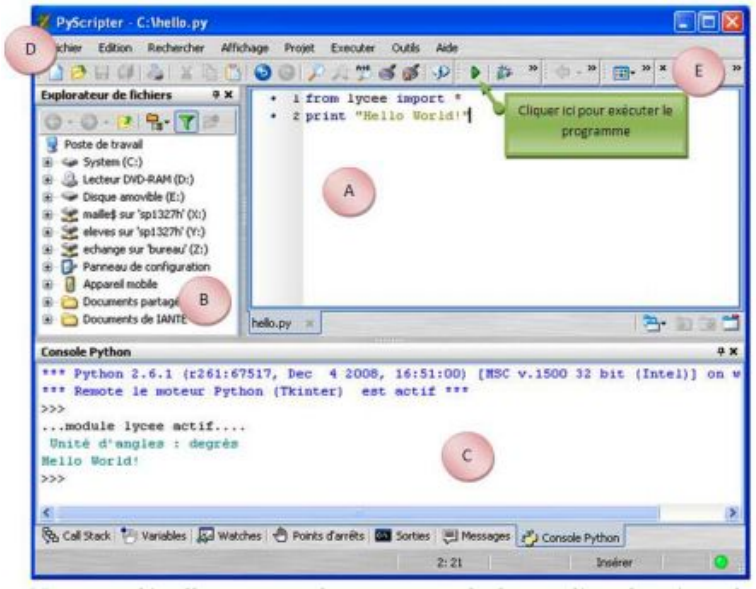
➔ écrire des petits scripts, programmes courts et très simples, chargés d'une mission très précise sur votre ordinateur ;

➔ écrire des programmes complets comme des jeux, des suites bureautiques, des logiciels multimédias, des clients de messagerie etc ;

➔ écrire des projets très complexes comme des progiciels (ensemble de plusieurs logiciels pouvant fonctionner ensemble principalement utilisés dans le monde professionnel).

II. Utilisation de Python

Lorsqu'on démarre *Edupython*, on obtient la fenêtre suivante :



La fenêtre de l'éditeur est composée de plusieurs zones :

➢ **A** : Zone de saisie du programme

➢ **B** : Zone de l'explorateur windows pour aller chercher vos fichiers

➢ **C** : Zone où le programme s'exécute, cette zone s'appelle aussi la console Python. On remarque 3 chevrons.

➢ **D** : La barre de menu

➢ **E** : La barre d'outils

a. En mode console *Application : TP 1 ex 1*

La console correspond à la zone **C** de la fenêtre active. On y voit un message d'accueil donnant quelques informations concernant la version de Python mise en route, l'architecture de l'ordinateur utilisée ainsi que les commandes à saisir pour obtenir davantage d'informations.

On remarque les triples chevrons **>>>** qui indiquent que Python est prêt à recevoir la première instruction.

Cette console va permettre de tester directement du code. On saisit une ligne d'instruction puis on appuie sur la touche **ENTRÉE** et Python nous répond. Puis on en saisit une deuxième, puis une troisième …un peu comme lorsqu'on utilise une calculatrice. Le principal inconvénient réside dans le fait que ce qui est saisi dans la console n'est pas sauvegardé.

b. En mode «éditeur» *Application : TP 1 ex 2*

Il s'agit de la zone **A** de la fenêtre active.
On peut y saisir une suite d'instructions que l'on pourra sauvegarder. L'extension du fichier est **.py**
Ce script est alors modifiable et exécutable à souhait.

III. Les variables et leurs types

Un des concepts les plus basiques en programmation est celui de **variable**.
Une variable est un mot (un identifiant) qui contient une unique valeur.

En *Python*, les noms de variables doivent obéir à quelques règles simples :

- un nom de variable doit toujours commencer par une lettre.
- seules les lettres ordinaires sont autorisées.

```
>>> n = 7                          # définir n et lui donner la valeur 7
>>> msg = "Quoi de neuf ?"         # affecter la valeur "Quoi de neuf ?" à msg
>>> pi = 3.14159                   # assigner sa valeur à la variable pi
```

Les trois 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 ;
- ➢ créer et mémoriser une valeur particulière ;
- ➢ établir un lien (par un système interne de pointeurs) entre le nom de la variable et l'emplacement mémoire de la valeur correspondante.

En Python, il n'est pas nécessaire de déclarer de type (pas besoin de dire « n est un entier compris entre . . . ») ; le typage se fait automatiquement.

```
>>> x, y, ph = 4, 15.3, "bonjour"
>>> type(x)
<type 'int'>          # x est un entier entre -2^(31)+1 et 2^(31)-1
>>> type (y)
<type 'float'>        # y est un flottant (nombre à virgule)
>>> type(ph)
<type 'str'>          # ph est une chaine de caractéres
```

En plus des 3 types cités ci-dessus, le langage python gère plusieurs types de variable :

bool (booléen) / *long* (entier non limité) / *complex* / *list* / *tuple* (n−uplet) / *dict* (dictionnaire) / *function* / . . .

a. Le type *INT*

Définition : Les `int` représentent le type le plus facilement représentable sur une architecture donnée.

Exemple :

On verra plus tard que sur une machine 32 bits, un int permet de représenter des nombres entiers compris entre $-2^{31}+1$ et $2^{31}-1$, soit entre −2 147 483 648 et 2 147 483 647.

b. Le type *FLOAT*

Définition : Pour qu'une donnée numérique soit considérée par python comme étant du type `float`, il suffit qu'elle contienne dans sa formulation un élément tel qu'un point décimal ou une puissance (un exposant) de 10.

Exemple :

```
>>> x, y, z, t, u = 3.14, 10., .001, 1e100, 3.14e-10
>>> type(x)
<type 'float'>
>>> type(y)
<type 'float'>
>>> type(z)
<type 'float'>
>>> type(t)
<type 'float'>
>>> type(u)
<type 'float'>
```

c. Le type *BOOL*

Définition : Un booléen est un type de données qui ne peut prendre que deux valeurs : vrai ou faux.

Lors de l'exécution d'un programme contenant une conditionnelle, *Python* évalue la véracité d'une condition, c'est-à-dire évalue si une expression est vraie ou fausse.

En effet :

- ❏ un ordinateur ne manipule strictement que des nombres, cela s'applique aussi à la notion de vrai/faux.
- ❏ *Python* considère que toute valeur numérique autre que zéro est « vraie » et seule la valeur zéro est « fausse ».

Le petit script suivant n'affiche « faux » que si l'on entre la valeur 0 ; pour toute autre valeur numérique, on obtient « vrai ».

```
n = int(input("Entrez un nombre entier quelconque : "))
if n:        # on reprendra cette structure plus tard
        print("vrai")
else:
        print("faux")
```

On obtient des résultats analogues avec les chaînes de caractères ; on obtient « faux » pour toute chaîne vide et « vrai » pour toute chaîne contenant au moins un caractère.

```
ch = input("Entrer une chaîne de caractères quelconque :")
if ch:
        print("vrai")
else:
        print("faux")
```

d. Le type *STR*

Définition : Une donnée de type `str` (en anglais string ou « chaîne de caractère ») peut se définir comme une suite quelconque de caractères.

Dans un script python, on peut délimiter une telle suite de caractères, soit par des apostrophes (simple quotes), soit par des guillemets (double quotes).

```
>>> phrase1 = 'les oeufs durs.'
>>> phrase2 = "'Oui', affirme-t-il,"
>>> phrase3 = "je mange"
>>> type(phrase2)
<type 'str'>          # même résultat avec phrase1 et phrase3
>>> print(phrase2, phrase3, phrase1)
'Oui', affirme-t-il, je mange les oeufs durs.
```

Comment gérer les sauts de ligne, les guillemets et l'apostrophe :

- La séquence \n dans une chaîne provoque un saut à la ligne.
- La séquence \' permet d'insérer une apostrophe dans une chaîne délimitée par des apostrophes.
- La séquence \" permet d'insérer des guillemets dans une chaîne délimitée elle-même par des guillemets.

```
>>> texte = "Cette\n phrase est\nlongue et voici des guillemets \"."
>>> # Attention à la position de \n pour des "blancs" significatifs
>>> print(texte)
Cette
 phrase est
longue et voici des guillemets ".
```

Comment accéder aux caractères individuels d'une chaîne :

Pour accéder à un caractère bien déterminé, on utilise le nom de la variable qui contient la chaîne et on lui accole, entre deux crochets, l'index numérique qui correspond à la position du caractère dans la chaîne.

Attention ! Les données numériques sont généralement indexées à partir de 0 et non de 1.

```
>>> texte = "Voici une chaîne"
>>> print(texte[1], texte[5], texte[10])
o   c
>>> print(2*(texte[3]+texte[1]+texte[2]+texte[7]))
coincoin        # Des opérations avec des chaînes ???
```

e. Le type *LIST*

Définition : On définit une liste comme une collection d'éléments séparés par des virgules, l'ensemble étant enfermé dans des crochets.

```
>>> mois = ["janvier", "fevrier", "mars", "avril", "mai", "juin", "juillet",
"aout", "septembre", "octobre", "novembre", "decembre"]
>>> nbjours=[31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
>>> print(type(mois),type(nbjours))
<type 'list'> <type 'list'>
```

Il est possible de faire des opérations sur les listes :

```
>>> # En reprenant les listes mois et nbjours
>>> # Accès à un élément d'une liste, comme pour les chaînes ...
>>> print("En", mois[10], "il y a", nbjours[10], "jours.")
En novembre il y a 30 jours.
>>> mois[7] = "AOUT"          # Remplacer un élément par un autre
>>> print(mois)
['janvier', 'fevrier', 'mars', 'avril', 'mai', 'juin', 'juillet', 'AOUT',
'septembre', 'octobre', 'novembre', 'decembre']
>>> nbjours.append("Toto")  # Ajouter un élément à la fin d'une liste
>>> print(nbjours)
[31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31, 'Toto']
>>> del(mois[0])             # Supprimer un élément d'une liste par son index
>>> print(mois)
['fevrier', 'mars', 'avril', 'mai', 'juin', 'juillet', 'AOUT', 'septembre',
'octobre', 'novembre', 'decembre']
>>> nbjours.remove(30)       # Supprime la première valeur 30
[31, 28, 31, 31, 30, 31, 31, 30, 31, 30, 31, 'Toto']
>>> nbjours.insert(3, 30)    # Insert l'élément 30 à l'index 3
[31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31, 'Toto']
```

f. Le type *TUPLE*

Définition : Un tuple est une séquence immuable : on ne peut pas modifier ses éléments ni lui en ajouter ou lui en enlever. On crée un tuple en écrivant ses éléments, séparés par des virgules et encadrés par des parenthèses.

En contrepartie de cette rigidité, les tuples sont très compacts (i.e. ils occupent peu de mémoire) et l'accès à leurs éléments est très rapide.

```
# En reprenant les listes mois et nbjours
# Accès à un élément d'un tuple, comme pour les chaînes ...
>>> t = 2, 'deux', 2.0, True, (1, 2, 3, 4)
>>> t
(2, 'deux', 2.0, True, (1, 2, 3, 4))
>>> t, type(t), len(t)
((2, 'deux', 2.0, True, (1, 2, 3, 4)), <type 'tuple'>, 5)
>>> t[1]
'deux'
>>> t[-1]
(1, 2, 3, 4)
```

IV. "Dialoguer" avec la machine

a. Instruction d'entrée *INPUT*

Définition : La fonction input() permet de "récupérer" une réponse donnée par l'utilisateur. input(ch) affiche ch et attend une entrée.

Remarque : *input*() retourne toujours une chaîne de caractère.

```
>>> n=input("Entrer un nombre entier : ")
Entrer un nombre entier : 5
>>> type(n)
<type 'str'>
>>> n=int(input("Entrer un nombre entier : "))
Entrer un nombre entier : 5
>>> type(n)
<type 'int'>
>>> n=int(input("Entrer un nombre entier : "))
Entrer un nombre entier : 7.7
>>> type(n)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '7.7'
```

b. Instruction d'entrée *PRINT*

Définition : La fonction `print()` permet l'affichage de données de n'importe quel type.

Remarques :

- `print` n'affiche pas les guillemets ou apostrophes "externes" ;
- une virgule entre les données à afficher permet de mettre plusieurs affichage sur une même ligne ;
- Pour écrire successivement sur la même ligne, on pourra utiliser `print(a, end = ' ' )`

```
>>> print('Toto a eu',4*6,"ans l'année dernière.")
Toto a eu 24 ans l'année dernière.
```