1 Écriture de script

1.1 Introduction

Nous avons vu comment `python` peut s'utiliser en mode interactif dans le shell ou en le lançant dans une console par la commande `python` : cela permet d'exécuter des commandes.

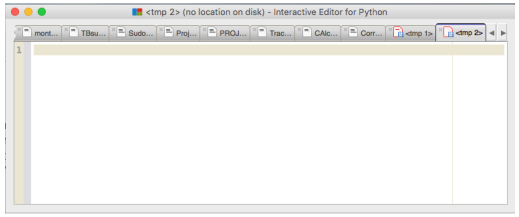
C'est pratique pour utiliser `python` comme une calculatrice ou pour programmer des algorithmes simples. Mais :

– Le mode interactif ne permet pas de développer des programmes complexes : à chaque utilisation il faut réécrire le programme. La modification d'une ligne oblige à réécrire toutes les autres lignes.

– Pour développer un programme plus complexe on saisit son code (suite d'instructions) dans l'éditeur avant de le faire exécuter par l'interpréteur : plus besoin de tout retaper pour modifier une ligne ; plus besoin de tout réécrire à chaque lancement.

L'E.D.I. permet d'écrire des programmes, de les sauvegarder, modifier, etc..., de lancer leur exécution, de la stopper, et d'avoir une fenêtre interactive qui aide au développement du programme et retourne les résultats de l'exécution.

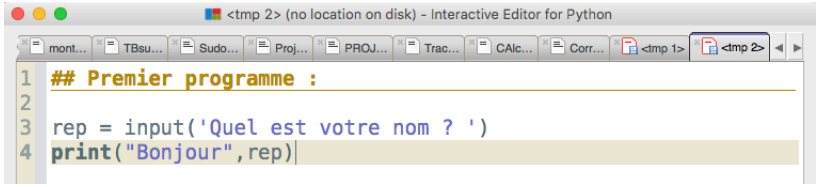
• Dans l'environnement de développement la fenêtre éditeur permettra de saisir le programme, de l'exécuter, déboguer avec une meilleure ergonomie.



• Les résultats affichés par le programme le seront dans la fenêtre "shell" (console). L'interaction avec le programme se fera à partir du shell.

1.2 Exemple - Fonctions d'entrée sortie - Chaîne de caractère

On a écrit dans l'éditeur le petit programme suivant (**script**) :



- La première ligne est un **commentaire** : elle débute par le caractère #, et est ignorée par l'interpréteur.
- La fonction **`input`** écrit dans la console son argument et attend la saisie par l'utilisateur d'une chaîne de caractère, et renvoie la valeur saisie.
- Ici la chaine de caractère saisie par l'utilisateur sera affectée à la variable que l'on a appelé **`rep`**.
- La fonction **`print`** écrira dans la console les valeurs de ses arguments (en les séparant d'un espace s'il y en a plusieurs). Ici elle a deux arguments.

Une **chaîne de caractère** est une suite de caractères alphanumériques ; pour la déclarer dans un script il faut la saisir entre deux délimiteurs : apostrophes '.' ou guillemets ".". C'est un **type** de données non numériques.

On exécute le programme en tapant :

– ctrl + e (windows) ou

– cmd + e (MAC) ; apparaît dans le shell :

```
In [1]: (executing lines 1 to 4 of "<tmp 2>")
Quel est votre nom ?
```

On saisit dans le shell notre réponse :

```
In [1]: (executing lines 2 to 4 of "<tmp 1>")
Quel est votre nom ? élève BCPST à Fénelon
```

après avoir appuyé sur entrée l'exécution du programme se poursuit :

```
In [1]: (executing lines 2 to 4 of "<tmp 1>")
Quel est votre nom ? élève BCPST à Fénelon
Bonjour élève BCPST à Fénelon

In [2]:
```

0) Ouvrir un nouveau fichier dans l'éditeur.
Y saisir le script précédent.
L'exécuter.
Sauvegarder le fichier dans votre répertoire utilisateur, dans un sous-répertoire `Informatique`, avec le nom `TP2`.

1.3 Attention : la fonction `input` renvoie une chaîne de caractère

Saisir et exécuter le script suivant :

```
n = input("Saisir un nombre entier : ")
print("Son carre est :", n**2)
```

On s'attendrait à ce que le script s'exécute en donnant le carré du nombre entier saisi. Mais ce n'est pas le cas : en effet la fonction **input** renvoie dans la variable `n` une chaîne de caractère, de sorte que dans l'instruction `print("Son carre est :", n**2)` l'interpréteur ne parvient pas à évaluer l'expression `n**2`, ce qui aboutit à une erreur. Pour y pallier on peut utiliser les fonctions de conversion **int** ou **float** :

Fonction	Action
int	renvoie son argument converti en entier
float	renvoie son argument converti en flottant

Par exemple :

```
In [1] : int("1")
Out[1]: 1

In [2] : float("1")
Out[2]: 1.0
```

1) Modifier le script précédent pour qu'il fonctionne.

2 Boucle for

• L'instruction **for ... in range(N)** permet de répéter une séquence d'instruction, en boucle, pour une variable qui décrit $0, 1, ..., N-1$.

```
for variable in range(N):
  ......   Instruction1
  ......   Instruction2
  ......   ⋮
  ......   Dernière instruction
```

meme espace — bloc d'instructions

- `N` : doit être de type entier (**int**). La séquence du bloc d'instruction est répétée `N` fois.
- `k` : la variable `k` est déclarée avec la valeur 0 à l'entrée de la boucle. Elle prend successivement les valeurs `0`, `1`, `2`, ... jusqu'à `N-1`.

Exemple : calcul (et affichage) de la somme des entiers de 0 à 100 :

```
S = 0
for k in range(101):
    S = S + k
print(S)
```

La variable `k` est déclarée à l'entrée de la boucle, et initialisée par la valeur `0`. Après chaque passage dans la boucle, sa valeur est incrémentée. Ici il y a 101 passages.

2) Que produira le script suivant :

```
S = 0
for k in range(101):
    S = S + k
    print(S)
```

Expliquer.

3) Écrire un script, qui demande à l'utilisateur de saisir son prénom et l'affiche 100 fois dans la console, et l'exécuter.

4) Écrire un script, qui demande à l'utilisateur de saisir un nombre entier `n`, puis affiche `n` fois votre prénom dans la console.

5) Calcul de sommes.

a) Écrire un script qui calcule et affiche dans la console la valeur de :

$$\sum_{k=0}^{100} k^2 = 0^2 + 1^2 + 2^2 + \cdots + 99^2 + 100^2.$$

b) Même question avec $\sum_{k=0}^{100} k^3$.

6) Algorithme de Babylone.

Soit la suite (u_n) définie par $u_0 = 2$ et pour tout $n \in \mathbb{N}$:

$$u_{n+1} = \frac{u_n}{2} + \frac{1}{u_n}$$

Écrire un script qui calcule et affiche dans la console les 15 premiers termes de la suite (u_n).

7) Somme de termes consécutifs d'une suite.

Soit la suite (u_n) définie par $u_0 = 1$ et pour tout $n \in \mathbb{N}^*$, $u_n = 2u_n^2 - 1$.

Écrire un script qui calcule et affiche dans la console la somme $\sum_{k=0}^{10} u_k$.

8) Suite de Fibonacci.

La suite de Fibonacci est la suite $(F_n)_n$ définie par :

$$F_0 = 0,\ F_1 = 1,\ \forall n \in \mathbb{N},\ F_{n+2} = F_n + F_{n+1}$$

a) Écrire un script qui calcule et affiche dans la console les 15 premiers termes de la suite de Fibonacci.

b) Écrire un script qui calcule et affiche dans la console la somme des 15 premiers termes de la suite de Fibonacci.

```
1
11
121
1331
14641
```

3 Opérations des chaines de caractères

Deux opérations peuvent être appliquées aux chaines de caractères :

- La concaténation `+` :
  Ex : `'aaa'+ 'bbb'` a pour résultat la chaine de caractère `'aaabbb'`.
  Les deux opérandes doivent être de type chaine de caractères.
- La duplication $*$:
  Ex : `'abc'* 3` a pour résultat la chaine de caractère `'abcabcabc'`.
  Un opérande doit être une chaine de caractère, l'autre doit être de type entier

Attention : `'abc'* 3.0` produira une erreur ! Le nombre doit être de type entier (**`int`**).

Opérations sur les chaîne de caractères	
`+`	Concaténation : `'aa' + 'bb'` produit `'aabb'`.
`*`	Duplication : `'ab' * 2` et `2 * 'ab'` produisent `'abab'`.

Exercice 9. à l'aide d'une boucle **`for`**, de la fonction **`print`** et des opérations des chaines de caractères, écrire 3 scripts pour afficher dans la console :

```
a)

o
oo
ooo
oooo
ooooo
```

```
b)

    o
   oo
  ooo
 oooo
ooooo
```

```
c)

    o
   ooo
  ooooo
 ooooooo
ooooooooo
```

Exercice 10. Triangle de Pascal

On peut remarquer que dans le triangle de Pascal, les cinq premières lignes donnent les puissances successives du nombre 11.
En appliquant cette remarque, écrire un script qui affiche les 5 premières lignes du triangle de Pascal, à l'aide d'une boucle **`for`** :

4 D'autres boucles `for`

```
for k in range(N):
  ......   Bloc d'instructions
```
meme espace / bloc d'instructions

`k` varie de `0` à `N`−`1`.

```
for k in range(M,N):
  ......   Bloc d'instructions
```
meme espace / bloc d'instructions

`k` varie de `M` à `N`−`1`.

```
for k in range(M,N,P):
  ......   Bloc d'instructions
```
meme espace / bloc d'instructions

`k` varie de `M` à `N`−`1` par pas de `P`.

`M`, `N` et `P` doivent être de type entier.
Ils peuvent être négatifs ;
`P` doit être non nul.

Exercice 11. Que calculent et affichent les 4 scripts suivants ?

```
a)

S = 0
for k in range(1,101):
    S += k**2
print(S)
```

```
b)

S = 0
for k in range(0,101,2):
    S += k**2
print(S)
```

```
c)

S = 0
for i in range(1,101):
    for j in range(1,101):
        S += (i+j)
```

```
c)

S = 0
for i in range(1,101):
    for j in range(1,i+1):
        S += (i+j)
```

Exercice 12. Écrire un script qui affiche dans la console : (indication utiliser deux boucles **for** imbriquées, la première faisant varier `i` de 1 à 4, la deuxième faisant varier `j` de 0 à `i`).

```
    o
   ooo
    o
   ooo
  ooooo
    o
   ooo
  ooooo
 ooooooo
    o
   ooo
  ooooo
 ooooooo
ooooooooo
```