TD : A la découverte de Python – 2e partie

1) Ordre de priorité des opérateurs

Question 1. Définir l'ordre de priorité entre les différents opérateurs arithmétiques et logiques disponibles en Python.

2) Interactions avec l'utilisateur

a) I/O

La plupart des programmes permettent à utilisateur d'interagir avec la machine. On peut distinguer deux grands types d'interactions :

- les entrées : l'utilisateur donne des informations à la machine
- les sorties : la machine affiche des informations à l'utilisateur

On parle donc souvent des entrées/sorties, en anglais I/O (pour input/output)[1].

b) Les entrées/sorties en Python

Concernant les sorties, nous avons déjà vu la fonction `print`, qui permet d'afficher un texte dans la console. Il existe d'autres sorties possibles (par exemple une fenêtre graphique, ou l'écriture dans un fichier texte) : nous en verrons certaines lors des prochaines séances.

Concernant les entrées, la méthode la plus simple est sans doute d'utiliser la fonction `input`. Cette fonction permet à l'utilisateur de saisir une chaîne de caractères dans la console.

Cette chaîne de caractères peut-être stockée dans une variable avec la syntaxe suivante :

```
chaine_saisie = input()
```

Remarques :

- Le nom de la variable (ici `chaine_saisie`) est libre : vous pouvez utiliser le nom que vous voulez, mais il est recommandé de le choisir intelligemment
- La fonction `input` ne prend aucun argument, c'est la raison pour laquelle il n'y a rien entre les parenthèses.
- Pour valider sa saisie, l'utilisateur doit appuyer sur la touche Entrée

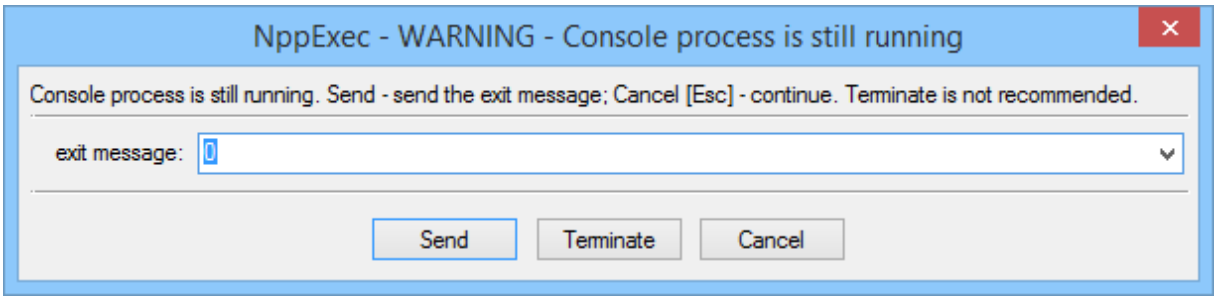
[1] La grande conférence annuelle de Google s'appelle d'ailleurs Google I/O

Question 2. Ecrire une séquence d'instructions effectuant les opérations suivantes :

- Afficher "Entrez votre prénom : "
- Lire la chaîne de caractères saisie par l'utilisateur et la stocker dans une variable
- Afficher "Bonjour *[prénom]* !" (ou *[prénom]* est la valeur saisie par l'utilisateur)

Remarque : Attention, l'appel à la fonction `input` bloque temporairement l'exécution du programme : tant que l'utilisateur n'a pas validée sa saisie, le programme continue à tourner.

Si vous essayez de relancer votre programme alors qu'il est encore en train de tourner, vous obtiendrez l'avertissement suivant :



Le cas échéant, il vous suffit de vérifier que le champ `exit message` contient bien 0, puis de cliquer sur `Send`. Cela aura pour effet de forcer la fermeture de la précédente exécution du programme, et de le relancer.

Remarque : Par défaut, la fonction `print` se contente d'afficher un texte et de passer à la ligne. Pour afficher plusieurs éléments avant de passer à la ligne, plusieurs solutions sont possibles :

- Utiliser la syntaxe `print(x, end = "")` (le deuxième argument indique qu'il ne faut pas passer à la ligne après avoir affiché la valeur `x`)
- Utiliser la syntaxe `print(x1, x2, …, xN)` : les valeurs `x1`, `x2`, … et `xN` seront affichées avant de passer à la ligne

Question 3. Ecrire une séquence d'instructions effectuant les opérations suivantes :

- Afficher "----------" (10 tirets)
- Afficher "Entrez votre prénom : "
- Lire la chaîne de caractères saisie par l'utilisateur et la stocker dans une variable
- Afficher "Bonjour *[prénom]* !"
- Afficher "----------" (10 tirets)
- Afficher "Quel âge avez-vous ?"
- Lire la chaîne de caractères saisie par l'utilisateur et la stocker dans une variable
- Afficher "Vous avez [age] ans."
- Afficher "----------" (10 tirets)

3) Premières fonctions

a) Faire plusieurs fois la même chose

Comme vous avez pu le constater dans l'exemple précédent, il arrive qu'un même programme doive effectuer plusieurs fois les mêmes opérations.

Pour éviter de réécrire le détail de ces opérations à chaque fois, on utilise des fonctions. Cela permet en outre de modifier plus facilement le code par la suite (il suffit de modifier le contenu de la fonction).

Question 4. Ecrire une fonction qui affiche la chaîne "----------" (10 tirets).

Remarque : Comme cette fonction ne prend pas d'arguments, on va utiliser la syntaxe
`def nom_de_la_fonction() :`

Question 4.b Modifier le code de la question 2 pour faire appel à cette fonction

Remarque : Comme vous pourrez le constater, en Python, une fonction doit être déclarée avant de pouvoir être appelée (c'est-à-dire plus haut dans le fichier).

b) Fonctions avec arguments

En pratique, la plupart des fonctions utilisent des arguments : on effectue toujours la même suite d'opérations, mais en utilisant un certain nombre de variables qui sont définies lors de l'appel de la fonction.

Question 5.c. Ecrire une fonction qui affiche la chaîne `----- Question X -----`, où `X` est un nombre entier passé en argument.

Exemple : `afficher_question(3)` affichera `----- Question 3 -----`

Question 5. b. Ecrire une fonction qui affiche une chaîne composée uniquement de tirets, avec comme argument la longueur de la chaîne à afficher.

Exemple : `afficher_tirets(3)` affichera `---` alors que `afficher_tirets(1)` affichera `-`

Question 5.b. Ecrire une fonction qui affiche une chaîne composée uniquement de symboles identiques, avec comme argument le symbole et la longueur de la chaîne à afficher.

Exemple : `afficher_symboles("#", 12)` affichera `############` alors que `afficher_symboles("*", 5)` affichera `*****`.

4) Types et conversions

a) Demander un entier à l'utilisateur

La valeur renvoyée par la fonction `input` est une chaîne de caractères : même si l'utilisateur tape un nombre entier, cette saisie est considérée comme du texte. Pour s'en convaincre, il suffit de faire quelques tests :

Question 6. Ecrire une séquence d'instructions effectuant les opérations suivantes :

- Afficher "Entrez un nombre entier : "
- Lire la chaîne de caractères saisie par l'utilisateur et la stocker dans une variable `x`
- Afficher `x + x`

Question 6.b. Ecrire une séquence d'instructions effectuant les opérations suivantes :

- Afficher "Entrez un nombre entier : "
- Lire la chaîne de caractères saisie par l'utilisateur et la stocker dans une variable `x`
- Afficher `x - 2`

b) Conversions

Pour remédier à ce problème, Python met à votre disposition plusieurs fonctions qui permettent de convertir une valeur d'un type vers un autre. Ces fonctions portent les noms de types cibles : `int`, `float` et `str`.

Par exemple, `int("3")` renvoie la valeur entière 3.

Question 7. Tester les fonctions `int`, `float` et `str` : existe-t-il des cas où la conversion est impossible ?

En utilisant ces fonctions de conversions et la fonction `input`, on peut donc demander à l'utilisateur de fournir un nombre entier ou décimal.

Remarque : Si l'utilisateur entre une chaîne qui n'est pas convertible dans le type attendu (par exemple une chaîne vide alors qu'on attend un nombre entier), le programme va planter. Il existe des mécanismes pour gérer ce type d'erreurs de saisie (on parle d'"exceptions"), mais nous ne nous en servirons pas pour le moment.

Question 8. Ecrire une fonction qui demande son âge à l'utilisateur, et affiche "Vous êtes mineur" si l'âge renseigné est strictement inférieur à 18, et "Vous êtes majeur" sinon.

Question 9. Utiliser une fonction et une boucle `while` pour écrire un programme qui effectue les opérations suivantes :

- Demander à l'utilisateur "Quel est le mot magique ?"
- Lire et stocker la chaîne saisie par l'utilisateur
    - Si l'utilisateur a tapé "Abracadabra", afficher "Bravo !"
    - Sinon, afficher "Mot magique incorrect", et recommencer

Question 10. Ecrire un programme qui effectue les opérations suivantes

- Choisir un nombre entier au hasard entre 1 et 100 : c'est le nombre mystère que l'utilisateur va essayer de découvrir.
- Demander à l'utilisateur de proposer un nombre
    - Si l'utilisateur a trouvé le nombre mystère, afficher "Bravo"
    - Si l'utilisateur a proposé un nombre strictement plus petit que le nombre mystère, afficher "Plus grand" et demander un autre nombre
    - Si l'utilisateur a proposé un nombre strictement plus grand que le nombre mystère, afficher "Plus petit" et demander un autre nombre

Question 11. Modifier le programme précédent pour afficher le nombre d'étapes nécessaires pour trouver le nombre mystère.

5) Fonctions récursives

a) Premier exemple : la factorielle

Une fonction récursive est une fonction qui s'appelle elle-même. En Python, il n'y a pas de déclaration particulière pour indiquer qu'une fonction est récursive : il suffit d'utiliser le nom de la fonction dans la fonction elle-même.

Un exemple simple est la fonction factorielle, définie sur les entiers positifs par

$$Factorielle(n) = \begin{cases} 1 & si\ n = 0 \\ n \times Factorielle(n-1) & si\ n > 0 \end{cases}$$

Question 12. Ecrire une fonction qui calcule la factorielle d'un nombre entier.

Remarque : Si cette fonction est appelée avec pour argument un nombre négatif, une erreur doit avoir lieu, car ce cas de figure n'est pas censé se produire. Pour faire ainsi "planter" le programme tout en informant l'utilisateur des raisons du plantage, on peut utiliser la syntaxe suivante :
`raise ValueError("Texte explicatif de l'erreur")`

b) Suite de Fibonacci :

On définit la suite de Fibonacci de la manière suivante :

$$Fib(n) = \begin{cases} 0 & si\ n = 0 \\ 1 & si\ n = 1 \\ Fib(n-2) + Fib(n-1) & si\ n > 1 \end{cases}$$

Question 13. Ecrire une fonction qui calcule le n[e] terme de la suite de Fibonacci.

6) Questions bonus

a) Damier

★ **Question 14.** Ecrire une fonction qui affiche un damier composé de symbole `X` et `O`, et dont la taille est passé en argument à la fonction.

Par exemple, `afficher_damier(6)` affiche

```
XOXOXO
OXOXOX
XOXOXO
OXOXOX
XOXOXO
OXOXOX
```

b) Cible

★ **Question 15.** Ecrire une fonction qui affiche une cible formée de `n` zones contenues les unes dans les autres, selon les règles suivantes :

- Chaque zone contient un chiffre
- La zone la plus au bord contient le chiffre 1
- Les chiffres augmentent au fur et à mesure qu'on se rapproche du centre
- La zone le plus au centre contient le chiffre `n`

Par exemple, `afficher_cible(4)` affiche

```
1111111
1222221
1233321
1234321
1233321
1222221
1111111
```

c) Suite de Fibonacci (suite)

Il existe plusieurs façons de calculer le n[e] terme de la suite de Fibonacci, et certaines sont beaucoup plus efficaces que d'autres. Une bonne façon d'estimer cette efficacité est de compter le nombre d'opérations élémentaires effectuées.

★ **Question 16.** Ecrire une fonction qui calcule le n[e] terme de la suite de Fibonacci, ainsi que le nombre d'appels récursifs nécessaires pour obtenir cette valeur.

Remarque : Votre fonction devra donc renvoyer un couple de résultats. Pour cela, il suffit d'utiliser des parenthèses : `return (resultat1, resultat2)`

Pour extraire les valeurs contenues dans un couple `couple`, plusieurs solutions sont possibles

- Déclarer deux variables en même temps : `x,y = couple`
- Utiliser des crochets pour cibler les éléments `x = couple[0]` et `y = couple[1]`

★ **Question 17.** Ecrire une fonction qui calcule le n[e] terme de la suite de Fibonacci en faisant au plus n appels récursifs.

d) Binôme de Newton:

Le binôme de Newton (ou coefficient binomial), noté $\binom{n}{p}$, correspond au nombre de façons de choisir *p* éléments parmi *n*, sans considération d'ordre.

★ **Question 18.** Que vaut $\binom{n}{0}$? Que vaut $\binom{n}{1}$?

Pour $0 < p \leq n$, on a la relation suivante : $\binom{n}{p} = \frac{n}{1} \times \frac{n-1}{2} \times \cdots \times \frac{n-p+1}{p}$

★ **Question 19.** Ecrire une fonction qui calcule ce binôme à partir de cette formule

On a également les formules $\binom{n}{p} = \frac{n!}{p! \times (n-p)!}$ (avec $x! = factorielle(x)$) et $\binom{n+1}{p+1} = \binom{n}{p} + \binom{n}{p+1}$

★ **Question 20.** Ecrire deux fonctions qui calculent ce binôme à partir de ces nouvelles formules, et comparer les résultats avec ceux de la question précédente.

Quelles sont les méthodes qui vous paraissent les plus efficaces ?