Types, opérateurs, mots-clefs et modules

INFORMATIQUE COMMUNE - TP n° 1.1 - Olivier Reynet

À la fin de ce chapitre, je sais :

- ☞ Utiliser Pyzo pour exécuter un code Python
- ☞ Utiliser le module os pour trouver ou changer le répertoire de travail
- ☞ Utiliser et identifier les types simples (int, float, boolean, complex)
- ☞ Utiliser les opérateurs en lien avec les types numériques et les chaînes de caractères
- ☞ Utiliser les modules os, math et random dans un code simple
- ☞ Utiliser la commande print pour faire afficher des variables formatées dans une chaîne

A Balbutiements

A1. Ouvrir l'IDE Pyzo. À droite se trouve normalement un shell interactif et à gauche un éditeur de texte.
A2. Dans le shell interactif, effectuer l'opération $21 + 7 + 14$.
A3. Définir une variable `i` de type `int`.
A4. Définir une variable `f` de type `float`.
A5. De quel type est le résultat de l'opération `i+f` ?
A6. Définir une variable `b` de type `bool`. Afficher la valeur de cette variable à l'écran à l'aide de la fonction `print`.
A7. Définir une variable `s` de type `string` contenant les caractères "Lycée La Pérouse, Promotion 2022".

Solution :

```
21+7+14  # 42
i = 3
f = 7.2
print(type(i+f)) # float
b = True
print(b)
s = "Lycée La Pérouse, Promotion 2022"
print(s)
```

B Module `os` et premier script Python

Le module `os` est une bibliothèque Python qui permet d'interagir avec le système de fichier de l'ordinateur. Dans cette section, on l'utilise pour savoir où l'interpréteur de IDE Pyzo se place dans le système

de fichiers lorsqu'il exécute les scripts.

B1. En utilisant le shell interactif et la fonction `getcwd` du module `os`, trouver le répertoire de travail courant.

Solution :

```
import os; print(os.getcwd())
```

B2. En utilisant la fonction `chdir` du module `os` qui signifie *change directory*, sélectionner un autre répertoire de travail courant.

Solution : `os.chdir("tp/")`

La commande accepte des chemins absolus ou relatifs.

Un chemin qui commence par un `slash`, c'est à dire qui décrit tout le chemin depuis la racine de l'arborescence, est dit **chemin absolu** : il désigne d'une manière univoque une ressource du système de fichier (un fichier, un répertoire ou un lien). Exemple : `/home/olivier/python/tp1`

Un chemin ne commençant pas par `/` est dit **chemin relatif**. Il désigne une ressource du système de fichier relativement à l'endroit où la commande est lancée. Exemple : `python/tp1`

B3. Dans la partie éditeur de l'IDE, à gauche, inscrire le programme suivant :

```
1 import os
2 current_dir = os.getcwd()
3 print("Current working direxctory is ", current_dir)
```

Ce programme peut aussi être désigné par le terme *script* : c'est une succession d'instructions.

B4. Sauvegarder ce script sous le nom `tp1.py` dans le répertoire courant.

B5. Lancer l'exécution de ce premier script en choisissant Run > Run file as script [1]

B6. En utilisant le module `os`, modifier le script pour se placer dans le répertoire de l'utilisateur `/home/user`.

B7. En utilisant le module `os` et sa documentation en ligne, lister tous les répertoires de ce répertoire.

Solution :

```
1  import os
2  current_dir = os.getcwd()
3  print("Current working directory is ", current_dir)
4  os.chdir("/home/user/reppython/")
5  print("Current working directory content is : ", os.listdir())
```

C Types et expressions

C1. Types et fonctions du module `math`

1. Cette commande aura le bon goût de sauvegarder le fichier dans le répertoire courant en même temps!

(a) Calculer le sinus de 1. Quel est le type de cette donnée?

(b) Calculer la partie entière de la constante `pi` à l'aide de la fonction `floor` du module `math`. Quel est le type de cette donnée?

Solution :

```
from math import sin, floor, pi
print(type(sin(1)))
print(type(floor(pi)))
```

C2. **Types issus d'expressions** Trouver et vérifier les types des expressions suivantes :

(a) 3

(b) -3

(c) 3.

(d) 3.5

(e) 0.334

(f) .334

(g) `3 + 3`

(h) `3 + 5.5`

(i) `3 * 3`

(j) `42 / 21`

(k) `5.5 * pi`

(l) `None`

(m) `True`

(n) `False`

(o) `[]`

(p) `[1,2,42]`

(q) `range(42)`

Solution :

```
for data in [3, -3, 4., 5.5, 0.334, .42, 1e-7, -77e3, 3 + 3, 3 + 5.5, 3 * 3,
      42 / 21, 5.5 * pi, None, True, False, [], [1, 2, 42], range(42)]:
  print(data, type(data))

Console output :
    3 <class 'int'>
    -3 <class 'int'>
    4.0 <class 'float'>
    5.5 <class 'float'>
    0.334 <class 'float'>
    0.42 <class 'float'>
    1e-07 <class 'float'>
    -77000.0 <class 'float'>
```

```
    6 <class 'int'>
    8.5 <class 'float'>
    9 <class 'int'>
    2.0 <class 'float'>
    17.27875959474386 <class 'float'>
    None <class 'NoneType'>
    True <class 'bool'>
    False <class 'bool'>
    [] <class 'list'>
    [1, 2, 42] <class 'list'>
    range(0, 42) <class 'range'>
```

D Opérateurs, types et priorités

Évaluer le type puis la valeur de ces expressions à l'écrit. Les vérifier sur l'ordinateur par la suite.

D1. `10 + 20 * 30`

D2. `100 + 200 / 10 - 3 * 10`

D3. `-1**2`

D4. `(-1)**2`

D5. `3 * 39 % 5`

D6. `39 % 5 * 3`

D7. `name = "Guillaume"; age = 13; expression = name == "Guillaume"or name == "Alix"and age < 8`

D8. `name = "Guillaume"; age = 13; expression = (name == "Guillaume"or name == "Alix")and age < 8`

Solution :

```
print(10 + 20 *30)  # 610  int
print(10 + 200 /10 +3 *10)   # 60.0  # float
print(-1**2)   # -1 exp has precedence
print((-1)**2) # 1 () has precedence
print(3 * 39 % 5) # 2 same precedence, left to right associativity
print(39 % 5 * 3) # 12 same precedence, left to right associativity
name = "Guillaume"; age =  13; expression = name == "Guillaume" or name == "
    Alix" and age < 8; print(expression)   # True  and has precedence
name = "Guillaume"; age =  13; expression = (name == "Guillaume" or name ==
    "Alix") and age < 8; print(expression) # False () have precedence
```

E Opérateurs sur les chaînes de caractères et boucle for

Pour les questions suivantes, vous aurez besoin d'utiliser les opérateurs + et * sur les chaînes et parfois des boucles `for`. On rappelle ici la syntaxe de la boucle `for` :

```
for i in range(10):
  print(i)
```

E1. Créer une chaîne de caractères `"00000111000000111000000111"` qui peut représenter un signal numérique périodique.

Solution :

```
((("0"*5)+"1"*3)*3
```

E2. Créer la chaîne de caractères `"AACAACGAACAACGTAACAACGAACAACGT"` qui peut représenter une chaîne de nucléotides.

Solution :

```
((((((("A"*2)+"C")*2)+"G")*2)+"T")*2
```

E3. Importer la fonction `choice` du module `random` (et uniquement elle!). À l'aide de cette fonction et d'une boucle for, créer une chaîne de 50000 nucléotides au hasard qui peut représenter le génome d'un virus.

Solution :

```
from random import choice

s = ""
for i in range(50000):
  s += choice("ACGT")
print(s)
```

F Module random et affichage de chaînes de caractères

Les questions qui suivent combinent l'utilisation du module `random`, la création de chaînes de caractères et leur affichage avec `print`. On cherche à formater correctement la sortie du programme sur l'écran.

F1. À l'aide de la fonction `randint` du module `random`, simuler le lancer de trois dés six fois de suite. Les résultats s'afficheront sous la forme `<D1 : 3, D2 : 6, D3 : 1>`.

Solution :

```
from random import randint

for k in range(6):
  # print("<D1 :", randint(1, 6),", D2 : ", randint(1,6), ", D3 : ", randint
      (1,6), ">")
```

```python
print(f"<D1 : {randint(1, 6)}, D2 : {randint(1,6)}, D3 : {randint(1,6)}>")
```

F2. À l'aide de la fonction `randrange` du module `random`, générer un tirage de loto. Le résultat s'affichera ainsi : `25-14-17-21-5+7`.

Solution :

```python
from random import randrange  # borne sup exclue

s = ""
for k in range(5):
  s += str(randrange(1, 50, 1)) + "-"  # str concatenation
s = s[:-1]  # remove last - symbol !
s += "+" + str(randrange(1, 10, 1))
print(s)
```

F3. À l'aide de la fonction `shuffle` du module `random`, générer un mélange de cartes à partir de la main "78910VDRA", représentation d'une main ordonnée sous la forme d'une chaîne de caractères. Attention, bien lire la documentation, il se peut qu'il y ait un piège !

Solution : En fait, on ne peut pas mélanger les cartes directement dans la chaîne de caractères car la séquence est immuable. Cela aurait possible avec une liste. Par contre, on peut échantillonner la chaîne avec la fonction `sample`.

```python
from random import shuffle, sample
deck = "78910VDRA"
print(sample(deck, k=len(deck)))
```

F4. À l'aide du module `random` et d'une distribution uniforme, générer des variations du marché aléatoirement. La sortie aura la forme suivante :

```
BRENT 0.90%
ONCE D'OR -2.45%
EUR/USD -5.06%
VIX INDEX 3.17%
BITCOIN/USD 6.09%
```

Vous avez le choix des valeurs numériques : celles-ci sont des valeurs réelles ce qui donne une idée de la plage de valeur à utiliser pour le module `random`.

Solution :

```python
from random import uniform

keys = ["BRENT", "ONCE D'OR", "EUR/USD", "VIX INDEX", "BITCOIN/USD"]
```

```
s = ""
vinf, vsup = -8.1, 7.9
for k in range(len(keys)):
  s += str(keys[k]) + " " + str(uniform(vinf, vsup)) +"%\n"
print(s)
```

G Des séquences indiçables, tronçonables et itérables

Les chaînes de caractères `str` en python3 sont des séquences immuables. Au titre de séquence, elles sont :

indiçables on accède directement à un élément donné d'après son indice et les crochets `[]`.

tronçonnables on peut en extraire des tronçons grâce aux crochets `[start:stop:step]`,

itérables on peut parcourir tous ses éléments via une boucle for.

Voici un exemple pour chaque fonctionnalité :

```
s = "CPGE"
print(s[1])          # indexing --> P
print(s[0] + s[2])   # concatenation --> CG
print(s[1:3])        # slicing start stop --> PG
print(s[-1])         # negative indexing --> E
print(s[-3:-1])      # negative slicing --> PG
print(s[::-1])       # reverse string --> EGPC
print(s[0:-1:2])     # slicing start stop step string --> CG
```

G1. Extraire une lettre sur trois de la chaîne de caractères `"L3kedf pppzcyqbthhhguodinol huc5n'ryeztsyrtuj xdbbiongnfc oo!ze"` en commençant par la première lettre et afficher le résultat.

Solution :

```
s="L3kedf pppzcyqbthhhguodinol huc5n'ryeztsyrtuj xdbbiongnfc oo!ze"
print(s[0:-1:3])  # Le python c'est bon !
```

G2. **Palindromes :** écrire un code Python qui permet de savoir si un mot est un palindrome[2].

On peut commencer avec une boucle `while` dont on rappelle ici la syntaxe :

```
ch = "my string"
i = 0
while i < len(ch) :
    print(i)
    i += 1
```

La fonction `len` renvoie la longueur de la chaîne de caractères. Dans un second temps, on s'affranchira de la boucle en utilisant les propriétés des chaînes de caractères.

2. se lit dans les deux sens, comme *rotor* ou *radar* par exemple.

Solution :

```
    def is_palindrome(s):
        return s == s[::-1]


    def simple_is_palindrome(s):
        deb = 0
        fin = len(s) - 1
        while deb < fin and s[deb] == s[fin]:
            deb += 1
            fin -= 1
        return fin <= deb


    test_words = ["ressasser", "hannah", "citrouille", "elle", "radar", "
        rotor", "bob", "PHP", "SOS", "FIN"]
    for w in test_words:
        print(is_palindrome(w))
        print(simple_is_palindrome(w))
```