Introduction Python

Python 3

25 avril 2018

Introduction

Python est langage objet multi-paradigme. Il est doté d'un typage dynamique fort. C'est un langage libre.
Ce langage peut s'adapter à tout type de contexte grâce à de nombreux modules (extensions).
Il est très répendu dans le monde scientifique et notamment dans le calcul numérique.
Ce langage a été conçu par Guido van Rossum (Pays-Bas). Le début de ce langage commence pendant des vacances de Noël où le concepteur décide de travailler à sa création, fan des Monty Python il décide de baptiser son langage Python.
Le langage utilise l'indentation comme syntaxe.

Installation

https ://www.python.org/downloads/Python installation Installer également Ipython à l'aide de pip, équivalent de npm

Utilisez Visual Studio Code

Dans VS installez les modules suivants : Python Extension Pack Don Jayamanne Ainsi que Python ms-python-python

LEGB

Nous allons préciser tout de suite une notion fondamentale sur le mécanisme de portée de variable en Python.

L pour localement, Python regarde si vous avez défini la variable localement.

E pour englobante, Python regarde si la variable n'est pas définie dans la première fonction englobante puis remonte ...

G pour globalement, Python va alors chercher la variable de manière globale.

B pour Builtins, Python regarde alors dans le module Builtins.

Exemple LEGB

```
a = 11
b = 22
def f():
    b = 1
    c = 33
    def g():
        print(a) # affiche 11
        print(b) # affiche 1
    g()
f()

try:
    print(c) # n'est pas défini
except NameError:
    print("c n'est pas définie")

import builtins

print = 1 # ne pas faire ça même si c'est possible !

print("Hello ne marche plus ")
```

Passage d'arguments à une fonction

Python offre 4 méthodes pour passer des arguments à une fonction, voyez les exemples qui suivent :

```python
# classique
def g(a,b, c= 1):
    return a + b + c

print(g(2,2))

def h(a,b,c):
    return {'a' : a, 'b' : b, 'c' :c}

# dans l'ordre que l'on souhaite si nommé
print(h(a = 1,c = 2, b = 3))

# passage d'un tuple
def i(*t):
    return t

print(i(1,2,4)) # retourne un tuple (1, 2, 4)

def j(**d):
    return d # retourne un dictionnaire

print(j(a = 1,b = 2,c = 4)) # retourne un dictionnaire {'a': 1, 'b': 2, 'c': 4}
```

Structure de données

En Python une grande force du langage c'est les structures de données. Elles permettent de manipuler les données avec beaucoup d'intelligence et d'optimisation. Python est un langage adapté au Big Data par exemple, notamment pour sa grande richesse au niveau des structures de données.

Définition d'une liste

Une liste est une suite de valeurs séparées par une virgule et placées entre crochets :

```
l = [1,2,3,4,5,6,7]
# accès valeur 1
l[0]
# dernière valeur 7
l[-1]

# slicing retourne une nouvelle liste [6,7]
l[-2:]
```

l'opérateur : sur les listes

```
l = [1,2,3,4,5,6,7]

# affichera [2,3,4,5,6]
l[1:-1]

# affichera deux valeurs en partant du début [1,2]
l[:2]

# à partir de l'indice 2 jusqu'à la fin de la liste
l[2:]

#  début jusqu'à la fin en récupérant les éléments séparer de 2 indices
# affiche [1,3,5,7]
l[::2]
```

Les listes et quelques méthodes

```
l = []
l.append(10)
# équivalent à l[len(l):] = [10]

# ajoute tous les éléments d'un itérable
l.extend( x*x for x in range(10))

# insert 100 à la position 2 donne [0, 1, 100, 4, 9, 16, 25, 36, 49, 64, 81]
l.insert(2, 100)

# pop supprime de la liste un élément à la position 2
l.pop(2)

# supprime tous les éléments d'une liste équivalent à del l[:]
l.clear()

# renvoie le premier élément trouvé ou une exception de type ValueError, ici 2
l.index(100)

# compte le nombre d'élément len compte le nombre d'éléments d'une liste
l.count(2)
len(l)

# tri une liste par ordre croissant on peut passer un argument de type lambda
# lambda
l.sort()
l.sort(lambda x, y: cmp(x,y))
```

Une liste n'est pas copiée elle a une même référence

Si vous copiez une liste dans une autre variable celle-ci sera référencée vers la même liste, elle n'est pas copiée.
On notera également que les listes et les chaînes de caractères en Python ont des nombreuses propriétés en commun.

Exercice maximun d'une liste avec indice

Écrire une fonction qui permet de retourner le maximun d'une liste avec son indice. Vous pouvez utiliser les fonction builtins suivantes de Python : len pour connaître le nombre d'élément(s) d'une liste, range qui permet de générer une liste itérable dans une structure for.

Exercice lettre en majuscule

Écrire une fonction qui permet de vérifier qu'une liste de caractères ne contient que des majuscules. Documentez vous sur la fonction all de Python, elle permet de retourner un boolean.

Exercice multiplication

Écrire une fonction qui prend deux listes de nombres de même longueur et multiplie les éléments de la liste terme à terme en faisant leur somme.

Compréhension de liste

Elles permettent la construction de liste de manière concise.
Ci-dessous construction d'une liste des cubes :

```
cubes = [x**3 for x in range(10)]
```

Un exemple avec un tuple

Ci-dessous on utilise un tuple que l'on verra plus loin dans une compréhension de liste :

```
print([(x, y) for x in [1,2,3] for y in [3,2,1] if x != y])
# [(1, 3), (1, 2), (2, 3), (2, 1), (3, 2), (3, 1)]
```

de manière assez équivalente on écrirait :

```
for exp1 in seq1:
    for exp2 in seq2:
    ...
        for expN in seqN:
            if (cond):
```

Exercice multiplication avec tuple

Écrire une fonction qui prend deux tuples de nombres de même longueur et multiplie les éléments du tuple terme à terme en faisant leur somme.

Exercice Transposer une matrice

Soit la matrice suivante : transposez celle-ci, c'est-à-dire transformez les lignes en colonnes à l'aide d'une compréhension de liste :

```
matrix = [
    [1, 2, 3, 4, 5],
    [6, 7, 8, 9, 10],
    [11, 12, 13, 14],
 ]
```

```
transposed = [
    [1, 6, 11],
    [2, 7, 12],
    [3, 8, 13],
    [4, 9, 14],
    [5, 10, 15]
]
```

Les dictionnaires

Un autre type natif existe en Python : les dictionnaires. Ils sont indexés par des clés, qui peuvent être de n'importe quel type immuable, comme par exemple les chaînes de caractères ou les nombres. Par contre une liste ne peut être une clé car muable. Un dictionnaire est un ensemble non ordonné de pair clé/valeur devant être unique dans le dictionnaire. On peut définir un dictionnaire en utilisant une paire d'accolade vide :

```
a = {}
```

Stocker des données

Les principales opérations sur les dictionnaires sont de stocker une valeur pour une clé et extraire la valeur correspondante pour une clé. On peut également supprimer une paire clé/valeur avec l'opérateur del de Python.
Si vous stockez une valeur pour une clé déjà présente dans le dictionnaire l'ancienne valeur sera perdue.
Et si vous essayez d'accéder à une valeur pour une clé qui n'existe pas une exception sera levée.

Tableau de hash

Un dictionnaire est un tableau de hash, ceci permet d'accéder à une valeur du dictionnaire en un temps constant et extrèmement rapide. De même vérifier qu'une clé existe est directe :

```
students = {'alan' : 1, 'albert' : 2, 'brice' : 3}

print('albert' in students)
```

Visualiser le hash d'un dictionnaire

La fonction map et le mot clé hash permet de voir les hash dans la structure de dictionnaire :

```
students = {'alan' : 1, 'albert' : 2, 'brice' : 3}

map(hash, students)
[4477144871804925236, 8410158949543912268, -28357329630056009 46]
```

Dans un tableau de hash on implémente une fonction f(key,dict) permettant d'accéder directement à une valeur du dictionnaire. Notons que la fonction de hash de Python est très rapide.

Constructeur : liste vers dictionnaire

Le constructeur dict permet de transformer une liste en dictionnaire :

```
# prendre une iste et la transformer en dictionnaire
d = dict([('alan', 1001), ('albert', 10101), ('jack', 1111)])

print(d) # {'alan': 1001, 'albert': 10101, 'jack': 1111}
```

Opérations sur les dictionnaires

```
students = {'alan': 1001, 'albert': 10101, 'jack': 1111}
students['brice'] = 1000 # ajoute une paire clé/valeur

print(students['brice']) # accède à un élément

del students['brice'] # supprime un élément du dictionnaire

list(students.keys()) # transforme les clés en liste

sorted(students.keys()) # ordonne le dictionnaire par rapport à ses clés
```

Exercice comptage

Codez une fonction comptStr qui à partir d'une chaîne de caractères split celle-ci en liste et transforme la liste en dictionnaire associant le mot et le nombre de caractères composant le mot.

Exercice arbre

Soit le dictionnaire graph ci-dessous, comptez pour chaque sommet le nombre de noeud(s) adjacent(s) à ce dernier. $graph = \{'A' : ['B'],' B' : ['A',' C',' D'],' C' : ['B',' D'],' D' : ['B',' C']\}$

Définition d'un tuple

Les tuples sont un autre type natif dit de séquence comme les listes et les dictionnaires, les tuples sont non modifiables (non mutable), ils n'ont pas de méthode.
Un tuple est donc protégé en écriture. C'est un tableau de hash donc rapide pour l'accès et le parcours de ses éléments. Comme il est non mutable on peut l'utiliser comme clé d'une liste par exemple. On rappelle qu'une clé d'une liste doit être non mutable.

```
# définition d'un tuple
t = 'a', 'b', 'c'
```

Ils peuvent être imbriqués :

```
# définition d'un tuple
t = 'a', 'b', 'c'

u  = t, (1,2,4)

print(u) # (('a', 'b', 'c'), (1, 2, 4))
```

Notez que souvent on utilisera des parenthèses pour définir un tuple par exemple : (1,2,3). Dans certains cas ils seront nécessaires, comme les tuples imbriqués.

Un tuple est immuable

On peut accéder à une valeur d'un tuple, mais on ne peut pas modifier une de ses valeurs :

```
In [14]: t = 1,2

In [15]: t[0]
Out[15]: 1

In [16]: t[0] = 4
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-16-66917334790a> in <module>()
----> 1 t[0] = 4

TypeError: 'tuple' object does not support item assignment
```

Déballage, opérateur sur un tuple

Déballage de séquence. L'opération suivante permet de déballer des valeurs d'un tuple dans des variables :

```
t = 'a', 'b', 'c'
x, y, z = t

print(x,y,z)
```

Exercice fonction zip et opérateur *

Reprendre la matrice précédente et transposer celle-ci à l'aide de la fonction native zip et de l'opérateur de déballage * :

```
m = [[1,2], [3,4]]

print(list(zip(*m))) # zip renvoie un générateur
```

Les ensembles

Python fournit également un type de donnée pour les ensembles. Un ensemble est une collection non ordonnée sans élément dupliqué. Les ensembles supportent les opérations mathématiques comme les unions, intersections, différences et différences symétriques.
Pour définir un ensemble on peut utiliser les accolades mais attention on ne peut pas définir un ensemble vide comme suit {} (dans ce cas cela définit un dictionnaire). Pour définir un ensemble vide on utilisera la déclaration Python suivante : set().

Exemple d'utilisation des ensembles

```python
a = set('aaabbbccc')
print(a) # {'b', 'a', 'c'}

A = {'a', 'r', 'b', 'c', 'd'}
B = {'r', 'c'}
print(A-B) # {'a', 'd', 'b'} différence

A = {'a', 'r', 'b', 'c', 'd'}
B = {'r', 'c', 'f'}
print(A|B) # union ensembliste {'d', 'c', 'a', 'b', 'r', 'f'} sans répétition bien sûr

print(A&B) # intersction {'c', 'r'}
print(A^B) # A ou B mais pas dans les deux {'d', 'b', 'f', 'a'}

# On peut créer des compréhensions d'ensemble
a = {x for x in 'blablabla' if x not in 'bobob'}

print(a) # {'l', 'a'}
```

TP

Nous allons maintenant faire un TP pour mettre en application ce que l'on vient de voir dans le cours.

Modélisation et structure de données TP 1/3

Récupérez les sources de l'exercice sur Github dans le dossier B2 :
`https://github.com/Antoine07/Python/tree/master/B2`

Créez un nouvelle clé au tabelau populations, appelez cette clé "relation" et pour l'instant donner comme valeur à cette clé un tableau vide.

Parcourez le tableau relationships pour récupérer les relations entre les personnes de cette population. Par exemple $(0, 1)$ indique que Alan est en relation avec Albert. Ajoutez à la liste populations les relations de chaque personne de ce tableau :
$populations[0]["relation"].append(populations[1])$
Ajouterait à Alan sa relation avec Albert dans la liste populations.

Modélisation et structure de données TP 2/3

Calculer maintenant le nombre moyen des relations de notre population dans ce nouveau tableau.

Modélisation et structure de données TP 3/3 ***

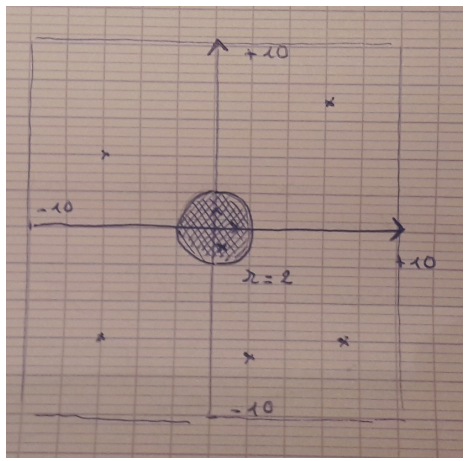
Créez une liste représentant les users (id) et le nombre de relation(s) qu'ils possèdent. Et retournez l'utilisateur qui possède le plus de relation(s).

Trouvez les amis des amis de chaque utilisateur.

En utilisant set sur une liste cela permet de supprimer les doublons, vous obtenez alors un ensemble. Retournez une liste sans doublon.

Modélisation TP cible et probabilité

Soit un cercle de rayon r = 2 inscrit dans un carré de dimension 20×20 parfaitement au milieu. Importer le module random et math dans votre script. Tirez au hasard une flèche dans ce carré, on supposera que toutes les flèches tombent dans le carré. Comptez le nombre de fois que vous atteignez la cible en simulant N = 1000 lancers. Faites un ratio des nombres d'impacts par rapport aux nombres de tires que pouvez-vous en dire ?



Juniper Green règles TP 1/2

Présentation des règles du jeux :

Le joueur 1 choisi un nombre compris entre 1 et 100

À tour de rôle, chaque joueur choisi un nombre parmi les multiples ou diviseurs du nombre choisi précédemment par son adversaire, ce nombre est également inférieur à 100.

Un nombre déjà joué ne peut être rejoué.

Le perdant est le joueur qui ne peut plus proposer un multiple ou diviseur.

Juniper Green fonctions utiles TP 2/2

Écrivez les fonctions utiles suivantes pour le jeu :

Générer une liste des valeurs possibles. À chaque fois qu'un joueur choisira une valeur il faudra retirer cette valeur de la liste des valeurs possibles, utilisez la méthode remove : *possibles*.*remove*(5) retirer 5 de la liste si il existe.

Créez les deux fonctions suivantes :

possible_multiple(n), cette fonction retournera tous les multiples possibles de n, compris entre 2*n et 100.

possible_divisor(n), même chose cette fonction ajoutera tous les diviseurs de n compris entre d = 2 et 100.

Pour fusionner deux listes, l, m vous pouvez, en Python, utiliser l'opérateur + : $l + m$.