Python

La suite

Sommaire

- Les fonctions
- Les fonctions récursive
- Les dictionnaires
- Les classes
- Les méthodes
- Les attributs
- Le constructeur

Bibliographie

https://docs.python.org/fr

http://www.xavierdupre.fr

https://python.doctor

Les fonctions

Les fonctions sont des portions de programmes qui reproduisent les mêmes instructions.

La fonction suivante calcule un polynôme de second degré :

$x^2 + x - 5$

```
def polynome ( x ) :
    x2 = x*x
    return x2 + x - 5
```

A chaque fois qu'on appellera la fonction polynôme, elle fera le même calcul sur des x différents. Cela évite principalement d'avoir à recopier les mêmes lignes à chaque fois qu'on en a besoin.

Les fonctions

- On peut appeler une fonction depuis une autre fonction.
- Une fonction peut prendre autant de paramètres que l'on veut à condition qu'ils aient des noms différents.
- On peut aussi leur associer une **valeur par défaut** .
- Une variable créée à l'intérieur d'une fonction n'existe pas à l'extérieur : **c'est une variable locale.**

```
from math import log  # on importe une fonction existante
def log_base ( x, base = 10 ) :
    return log (x) / log(base)

y = log_base (1000)      # identique à y = log_base (1000, 10)
z = log_base (1000, 2)   # logarithme en base deux
y,z
```

```
def calcul(x) :
    y = x**2
    z = x + y
    return z

print(z)    # déclenche une exception
```

Les fonction

Pour appeler une variable externe à une fonction au sein d'une fonction :

```
a = 2

def test():
    global a
    a = 3
    print("intérieur de test", a)

test()
print("après test", a)
```

Les fonctions récursives

Une fonction peut être récursive : elle s'appelle elle-même. Mais il est important de savoir qu'il existe un cas dans lequel elle ne s'appelle pas pour arrêter la récursion.

```
def recursive(x) :
    if x / 2 < 1 :
        print("je ne m'appelle pas pour x=",x)
        return 1
    else :
        print("je m'appelle pour x=",x)
        return recursive (x/2) + 1

recursive( 10 )
```

```
je m'appelle pour x= 10
je m'appelle pour x= 5.0
je m'appelle pour x= 2.5
je ne m'appelle pas pour x= 1.25
```

```
4
```

Les fonctions récursives

Avantage : Plus rapide qu'une boucle classique et peut faciliter l'écriture e certain programme.

Inconvénient : Là où une boucle ne conserve en mémoire que l'opération courante la récursion garde toutes les fonctions ouvertes et non encore terminées, se qui sature vite la mémoire.

Les dictionnaires

Un dictionnaire est une sorte de liste mais au lieu d'utiliser des index , on utilise des clés alphanumériques.

```
>>> a = {}
>>> a["nom"] = "Wayne"
>>> a["prenom"] = "Bruce"
>>> a
{'nom': 'Wayne', 'prenom': 'Bruce'}
```

Comment récupérer les clés et les valeurs d'un dictionnaire python par une boucle?

```
>>> fiche = {"nom":"Wayne","prenom":"Bruce"}
>>> for cle,valeur in fiche.items():
...     print cle, valeur
...
nom Wayne
prenom Bruce
```

Les classes

Définition :

Une classe est un ensemble incluant des variables ou ***attributs*** et des fonctions ou ***méthodes***.

En *python*, les classes sont des types modifiables.

Syntaxe S1 : Déclaration d'une classe

```
class nom_classe :
    # corps de la classe
    # ...
```

Les classes

Il est tout-à-fait possible de se passer des classes pour rédiger un programme informatique. Leur utilisation améliore néanmoins sa présentation et la compréhension qu'on peut en avoir.

*Pour faciliter l'utilisation d'une class on peut faire ce que l'on appelle une **instantiation** :*

Syntaxe S2 : Instanciation d'une classe

```
cl = nom_classe()
```

Faire un instantiation c'est créer une variable de type objet.

Les méthodes

Définition :

Les méthodes sont des fonctions qui sont associées de manière explicite à une classe. Elles ont comme particularité un accès privilégié aux données de la classe elle-même.

Le premier paramètre qui doit de préférence s'appeler self renvoie à l'objet sur lequel on appliquer la méthode.

Syntaxe S3 : Déclaration d'une méthode

```
class nom_classe :
    def nom_methode(self, param_1, ..., param_n):
        # corps de la méthode...
```

Les méthodes

Exemple : Une méthode qui renvoie un nombre aléatoire compris entre 0 et n.

```
rnd = 42

class exemple_classe:
    def methode1(self, n):
        """simule la génération d'un nombre aléatoire
           compris entre 0 et n-1 inclus"""
        global rnd
        rnd = 397204094 * rnd % 2147483647
        return int(rnd % n)
```

```
nb = exemple_classe()
l1 = [nb.methode1(100) for i in range(0, 10)]
print(l1)    # affiche [19, 46, 26, 88, 44, 56, 56, 26, 0, 8]
```

```
[46, 42, 89, 66, 48, 12, 61, 84, 71, 41]
```

Les attributs

Définition :

Les attributs sont des variables qui sont associées de manière explicite à une classe. Les attributs de la classe se comportent comme des variables globales pour toutes les méthodes de cette classe.

```
class exemple_classe:
    def methode1(self, n):
        """simule la génération d'un nombre aléatoire
           compris entre 0 et n-1 inclus"""
        self.rnd = 42  # déclaration à l'intérieur de la méthode,
        # doit être précédé du mot-clé self
        self.rnd = 397204094 * self.rnd % 2147483647
        return int(self.rnd % n)


nb = exemple_classe()
li = [nb.methode1(100) for i in range(0, 10)]
print(li)  # affiche [19, 19, 19, 19, 19, 19, 19, 19, 19, 19]
```

Le constructeur

Le constructeur d'une classe est une méthode qui, si elle est elle est définis s'exécute implicitement l'or de la création de chaque instance.

Elle suit la même syntaxe que les autres méthodes à ceci près que son **nom est imposé : __init__**

Hormis le premier paramètre, invariablement self, il n'existe pas de contrainte concernant la liste des paramètres excepté que le constructeur **ne doit pas retourner de résultat**.

Le constructeur

```
class MaClass:
    description = {'promo' : 'DevData', 'annee' : 2020, 'blaze' : 'Data Jungle'}
    promo = description ['promo']
    annee = description ['annee']
    blaze = description ['blaze']

    def __init__(self, prenom):
        self.prenom = prenom[::-1]
```

```
rafik = MaClass('Rafik')
```

```
rafik.prenom
```

```
'kifaR'
```

```
rafik.description
```

```
{'promo': 'DevData', 'annee': 2020, 'blaze': 'Data Jungle'}
```

```
print(rafik.promo)
print(rafik.annee)
print(rafik.blaze)
```

Ce qu’il faut retenir

Une classe permet en quelque sorte de regrouper ensemble des informations liées. Elles n’ont de sens qu’ensemble et les méthodes manipulent ces données liées.

C’est le cas pour un segment qui est toujours défini par ces deux extrémités qui ne vont pas l’une sans l’autre.

Allez plus loin : **Attributs implicites**

Pratique :

Le programme labyrinthe.py lit un fichier contenant le plan d'un labyrinthe. Il parcourt toutes les cases vides à partir de la position courante en les marquant avec un «.», afin d'éviter de tourner en rond. Il essaie successivement les directions droite, gauche, bas puis haut. C'est ce que fait la procédure récursive *thesee.*

1. Essayez de comprendre le fonctionnement de ce programme en l'exécutant «à la main» sur le petit labyrinthe.
2. Modifiez le programme Python présenté ci-dessus de telle manière que l'on puisse suivre le cheminement dans le labyrinthe: «droite», «gauche», «bas», «haut». Affichez aussi le labyrinthe chaque fois que la procédure *Thesee* est appelée, afin de suivre l'évolution des marques.

Projet : application de carnet d'adresse

Création d'une appli de carnet d'adresse pour y stocker nom, prénom, date de naissance.

1. Exécuter le code et comprenez son fonctionnement.
2. Modifier la class *CarnetAdr* et y ajouter la méthode *ajouter_fiche* permettant d'ajouter un contact a une liste de contact.
3. Sauvegarder les données en utilisant le la méthode de stockage *dump()* du module *pickle.*