Python
Programmation Orientée Objet (POO)

Objectifs

- Améliorer son code python
- Le rendre modulaire
- Le rendre résilient

Programme

- Histoire
- Concept
- Classes
- Encapsulation de données
- Méthodes spéciales
- Héritage
- Packages
- Exceptions

Histoire

- Concept 60’s / 70’s
  - Smalltalk
- Premiers langage 80’s
  - C++, Objective-C, ...
- Âge d’or 90’s
  - PHP, Java, Python

Concept

- ≠ programmation procédurale
- Briques logicielles appelées *objets*
  - Représentation d’un concept, idée ou entité physique
- Possède une structure interne, un comportement et sait interagir avec ses pairs
- Objectif : représenter ces objets et leurs relations permet de retranscrire les éléments du réel sous forme virtuelle

Exemple

- Une voiture
  - Qu’est-ce qu’elle est ?
  - Qu’est-ce qu’elle fait ?
- Une personne
  - Qu’est-ce qu’elle est ?
  - Qu’est-ce qu’elle fait ?
- Une relation entre les 2 ?

Classe

- L’objet est nommé classe
- Ce qu’il est sera ses propriétés
- Ce qu’il fait sera des méthodes

Objets en Python

voiture.py

```
class Voiture:
    color = None
    running = False

    def __init__(self, color):
        self.color = color

    def demarrer(self):
        self.running = True

    @staticmethod
    def nombreRoues(cls):
        return 4

    def etat(self):
        if self.running == True:
            return 'la voiture est démarrée'
        else:
            return 'la voiture est à l'arrêt'
```

Object en python

```
>>> from voiture import Voiture


>>> print(Voiture.nombreRoues())

>>> voiture_rouge = Voiture('rouge')
>>> print(voiture_rouge.etat())
'la voiture est à l’arrêt'
>>> voiture.demarrer()
>>> print(voiture_rouge.etat())
'la voiture est démarrée'

>>> voiture_grise = Voiture(‘gris')
```

Bonnes pratiques

- PEP 8 -- Style Guide for Python Code
  - Nom des classes en CamelCase
  - Nom des fonctions et variables en snake_case
  - ‘self’ premier argument méthode d’instance
  - ‘cls’ premier argument méthode de classe
  - ...

Encapsulation de données

- Par défaut, possible d’accéder directement aux propriétés

```
>>> voiture = Voiture('rouge')
>>> voiture.color
'rouge'
```

- Pratique à éviter
- Solution : accès par méthodes (getters, setters)

Encapsulation

- Utilisation des modificateurs
  - Public
    - Propriété ou méthode accessible tout le temps
  - Protected
    - Accessible que par la classe ou ses sous-classes
  - Private
    - Accessible que par la classe

Encapsulation

- Utilisation des modificateurs
  - Public
    - Nom de variable normal
  - Protected
    - Nom de variable précédé par un seul _ (underscore)
  - Private
    - Nom de variable précédé par __ (2 underscores)

Encapsulation

```
class Voiture:
    brand = "Renault"
    _color = "rouge"
    __is_running = False


    def isRunning(self):
        return self.__is_running
```

```
>>> from voiture2 import Voiture
>>> voiture = Voiture()
>>> voiture.brand
'Renault'
>>> voiture._color
'rouge'
>>> voiture.__is_running
Traceback (most recent call last):
  File "<stdin>", line 1, in
<module>
AttributeError: 'Voiture' object has
no attribute '__is_running'
>>> voiture.isRunning()
False
>>> voiture._Voiture__is_running
False
```

Méthodes spéciales

- Python fournit des méthodes spéciales par défaut
  - __repr__
  - __init__
  - ...

  - http://www.ironpythoninaction.com/magic-methods.html

Héritage

- Un objet peut hériter d'un autre objet
  - On peut avoir un objet « Person » et un objet « Employee » qui hérite de Person. Employee hérite alors de toutes les propriétés et méthodes

```
class Person:
    def __init__(self, name):
        self.name = name
    …

class Employee(People):
    def getCompany():
        return self.company
```

Héritage

- Une méthode peut être « surchargée » :
  - Avec le même nom dans la classe fille, elle annule et remplace la méthode du parent.
- Un méthode du parent peut être appelée dans une méthode de l’enfant

```
class Person:
    def parler(self):
        print('Salut')

class Employee:
   def parler(self):
        Person.parler()
        print('Bonjour')
```

Packages Python

- Chaque fichier Python d’un projet est appelé « module »
- Un semble de modules peut former un « package »
  - Sert à organiser son code par blocs de fonctionnalités

Packages

- Sans package, le « import » ne marche qu’avec les modules dans le même répertoire
- Pour créer un package, créer un fichier __init__.py à la racine du dossier faisant office de package

app.py

```
from package.module import Module
Import package.module
```

Packages

- Pour import tout un package avec « import package », il faut modifier le __init__.py

__init__.py

```
from .module import Module
from .module2 import Module2
```

Exceptions

- Si un bout de code « plante », il génère une erreur (ex : 5/0) appelée Exception
- On peut intercepter ces exceptions pour éviter de planter tout le script

```
>>> v = 0
>>> w = 5
>>> try:
...     w/v
...     print('OK')
... except:
...     print('Erreur')
...
Erreur
```

Exceptions

- On peut cibler une exception en particulier

```
>>> v = 0
>>> w = 5
>>> try:
...     w/v
...     print("Ok pas erreur")
... except TypeError:
...     print("Merci d'utiliser des chiffres")
... except ZeroDivisionError:
...     print("Merci de ne pas utiliser le 0")
...
Merci de ne pas utiliser le 0
```