Décorateurs en Python

La version électronique contient des liens vers les documents en ligne, les outils mentionnés, etc.

Quelques lectures possibles : les bases et les sous-programmes (à partir de la planche 39) et les classes et les objets.

1 Création d'un environnement virtuel Python

Exercice 1 Le module venv permet de créer un environnement Python propre à un projet, avec ses propres modules installés, indépendamment de ceux installés sur le système.

Voici les étapes pour créer une environnement virtuel dans le dossier `~/nosave/py3` :

```
python3 -m venv $HOME/nosave/py3
$HOME/nosave/py3/bin/python --version            # python3, normalement !
```

Pour l'activer, il suffit alors de faire :

```
source $HOME/nosave/py3/bin/activate
which python                # Celui de py3 !
which pip                   # Aussi celui de py3
```

L'outil `pip` permet d'installer des paquets Python. Par exemple, pour installer l'outil de test pytest, on peut faire :

```
pip install pytest
```

Pour désactiver l'environnement virtuel Python, il suffit de faire :

```
deactivate
```

Activer l'environnement oblige à taper une commande un peu longue. On peut définir un alias pour le faire plus rapidement :

```
alias p3="source $HOME/nosave/py3/bin/activate"
```

Pour que cet alias soit permanent, on peut l'ajouter dans le fichier `$HOME/.bashrc`.

2 Quelques éléments algorithmiques

Exercice 2 Que donne l'évaluation des expressions suivantes :

```
a = [8, 1, 4, 6]
b = [8, 1, 4, 6]
c = b
a is b          # ?
a == b          # ?
c is b          # ?
p, *m, d = a    # p ? m ? a ?
len(a)          # ?
a[-1]           # ?
a[0]            # ?
x = 'a'
s = str(x)    # ?
r = repr(x)    # ?
```

Exercice 3 Comparer :

```
l = []
for i in range(2, 10):
    l.append(2 * i + 5)
```

et :

```
l = [2 * i + 5 for i in range(2, 10)]
```

Exercice 4 Dans le code qui suit, que sont a et b ? Quelle est la valeur des expressions ou l'effet des instructions.

```
a = { 1, 2, 3, 4, 3, 2, 1}
b = { 'fr': 'France', 'de': 'Allemagne' }
len(a)                  # ?
len(b)                  # ?
b[5] = 'fr'             # ?
x = b['fr']             # ?
y = b.get('it', -1)     # ?
del b[5]                # ?
b.items()               # ?
b.values()              # ?
b.keys()                # ?
p = b.pop('fr')         # ?
```

3 Les sous-programmes

Exercice 5 : Premier sous-programme

5.1. Expliquer la fonction suivante (fichier `index.py`) :

```python
def index(sequence, element):
    """Chercher l'indice de la première occurrence de 'element' dans 'sequence'

    :param sequence: la séquence
    :param element: l'élément cherché
    :returns: l'indice de la première occurrence de 'element'
    :raises ValueError: l'élément n'est pas dans la séquence
    """
    for indice, elt in enumerate(sequence):
        if elt == element:   # On l'a trouvé !
            return indice
    else:    # jamais trouvé
        raise ValueError('élement non trouvé')
```

5.2. Lire et comprendre le fichier `index_test.py` qui est un test pytest :

```python
import pytest
from index import index

@pytest.fixture
def liste1():
    return [1, 4, 2, 6]

def tests_nominaux(liste1):
    assert index(liste1, 1) == 0
    assert index(liste1, 4) == 1
    assert index(liste1, 2) == 2
    assert index(liste1, 6) == 3

def tests_erreurs(liste1):
    with pytest.raises(ValueError):
        index(liste1, 7)
    with pytest.raises(ValueError):
        index(liste1, 'x')

def test_avec_str():
    assert index('Bonjour', 'j') == 3
```

5.3. Exécuter ce fichier en faisant :

```
pytest index_test.py
```

Exercice 6 Exécuter le programme suivant (fichier `fn_param_multiples.py`) et en déduire à quoi correspondent `p` et `kw` dans la signature de f. Que signifie `*p` en paramètre de `print` ? Est-ce que l'appel `f(1, 2, 3)` est possible ?

```
def f(a=4, *p, x, **kw):
    print('a =', a)
    print('p =', p)
    print('kw =', kw)
    print(*p, sep='... ', end=' !\n')

f(1, 2, 3, x=5, y=6, z=7)
```

Exercice 7 Expliquer la fonction suivante :

```
def zero(f, a, b, *, precision=10e-5):
    '''Retourner une abscisse où la fonction f s'annule entre a et b'''
    assert f(a) * f(b) <= 0

    if a > b:
        a, b = b, a
    while b - a > precision:
        milieu = (a + b) / 2
        if f(a) * f(milieu) > 0:
            a = milieu
        else:
            b = milieu
    return (a + b) / 2
```

1. Que représente le paramètre `f` ?
2. Comment donner une valeur au paramètre appelé `precision` ?
3. Appeler cette fonction avec la fonction $f(x) = x^2 - 2x - 15$ sur l'intervalle $[0, 15]$ avec une précision de 0.01.

4 Décorateurs en Python

Exercice 8 : Comprendre le principe
Dans le programme suivant, le terme `@fn_bavard` devant la définition de la fonction est un décorateur. Il ajoute une propriété à l'objet.

```
def fn_bavard(f):
    def f_interne(*p, **k):
        print('debut de f_interne()')
        f(*p, **k)
        print('fin de f_interne()')
    print('dans fn_bavard')
    return f_interne

@fn_bavard
def exemple(x, y='ok'):
    print('exemple:', y, x)

print('Appel à exemple')
exemple('?')
print(exemple.__qualname__)
```

En Python, un tel décorateur est une fonction qui prend en paramètre une fonction (la fonction décorée) et qui retourne une fonction. C'est cette fonction qui sera la vraie définition de la fonction `exemple`.

Ainsi, on pourrait remplacer `@fn_bavard` par l'instruction suivante placée après la définition de la fonction `exemple`.

```
exemple = fn_bavard(exemple)
```

8.1. Exécuter le programme et comprendre ce qu'il se passe.
8.2. Exécuter le programme avec Python Tutor (Visualize Execution).
8.3. On remarque que le nom de exemple (dernière ligne) n'est pas le bon ! Il faudrait copier sur la décoration (la fonction interne) le nom, la documentation... de la fonction décorée. Ceci peut être fait en ajoutant la décoration `@functools.wraps(f)` devant la définition de la fonction `f_interne`. Bien sûr, il faut importer functools (`import functools`, par exemple en début de fichier).

Vérifier que le nom de la fonction exemple est maintenant le bon !

Exercice 9 : Décorateur deprecated
Définir un décorateur `deprecated` qui afficher un message du sytle « la fonction XXX ne devrait plus être utilisée... » à chaque appel de la fonction marquée « deprecated ».

Exercice 10 : Décorateur trace
On veut pouvoir tracer tous les appels et les retours de la fonction décorée.

Voici un exemple de programme utilisant le décorateur trace.

```python
from trace import trace

@trace
def fact(n):
    if n <= 1:
        return 1
    else:
        return n * fact(n - 1)

@trace
def est_pair(n):
    return n == 0 or est_impair(n - 1)

@trace
def est_impair(n):
    return n > 0 and est_pair(n - 1)

def main():
    x = 3
    print(f'fact({x}) =', fact(x))
    print(f'{x} est', 'pair' if est_pair(x) else 'impair')

if __name__ == '__main__':
    main()
```

Et le résultat de son exécution :

```
  File "trace_exemple_simple.py", line 20
    print(f'fact({x}) =', fact(x))
                        ^
SyntaxError: invalid syntax
```

Bien sûr, si on enlève les décorateurs, on aura comme résultat de l'exécution :

```
fact(3) = 6
3 est impair
```

10.1. Écrire le décorateur trace.
10.2. Que se passe-t-il si une exception se propage à l'extérieur d'une fonction tracée ? Adapter en conséquence le décorateur trace.

5 Classes en Python

Exercice 11 Expliquer les principaux constituants de la classe suivante :

```python
class Robot:
    """ Robot qui sait avancer d'une case et pivoter à droite de 90°.
        Il est repéré par son abscisse x, son ordonnée y et sa direction.
```

```
    """
    # des attributs de classe
    _directions = ('nord', 'est', 'sud', 'ouest')    # direction en clair
    _dx = (0, 1, 0, -1)        # incrément sur X en fonction de la direction
    _dy = (1, 0, -1, 0)        # incrément sur Y en fonction de la direction

    def __init__(self, x, y, direction):
        """ Initialiser le robot self à partir de sa position (x, y) et sa direction. """
        self.x = x
        self.y = y
        self.direction = Robot._directions.index(direction)

    def avancer(self):
        """ Avancer d'une case dans la direction.  """
        self.x += Robot._dx[self.direction]
        self.y += Robot._dy[self.direction]

    def pivoter(self):
        """ Pivoter ce robot de 90° vers la droite.  """
        self.direction = (self.direction + 1) % 4

    def afficher(self, *, prefix=''):
        print(prefix, self, sep='')

    def __str__(self):
        return '({}, {})>{}'.format(self.x, self.y, Robot._directions[self.direction])

    @classmethod
    def changer_langue(cls, langue):
        if langue.lower() == 'fr':
            cls._directions = ('nord', 'est', 'sud', 'ouest')
        else:
            cls._directions = ('north', 'east', 'south', 'west')
```

Exercice 12 Comprendre, exécuter et commenter le programme suivant.

```
from robot import Robot

r1 = Robot(4, 10, 'est')
r1.afficher(prefix='r1 = ')
r2 = Robot(15, 7, 'sud')
r2.afficher(prefix='r2 = ')
r1.pivoter()
r1.afficher(prefix='r1 = ')
r2.avancer()
r2.afficher(prefix='r2 = ')
Robot.pivoter(r2)            # syntaxe "classique"
Robot.afficher(r2, prefix='r2 = ')
print('Robot.pivoter :', Robot.pivoter)
```

```
print('r2.pivoter :', r2.pivoter)
Robot.changer_langue('en')
r2.afficher(prefix='r2 = ')
print("type de r1 :", type(r1))
```

Exercice 13 Comprendre, exécuter et commenter le programme suivant.

```
from robot import Robot

r1 = Robot(4, 10, 'est')
print('r1 =', r1)
r2 = Robot(15, 7, 'sud')
print('r2 =', r2)

r1.nom = "D2R2"        # on a ajoutÃ© un nom Ã  r1 mais r2 n'a pas de nom

def tourner_gauche(robot):
    robot.direction = (robot.direction + 3) % 4

Robot.pivoter_gauche = tourner_gauche

r2.pivoter_gauche()    # direction devient 'est'
print('r2 =', r2)

del r1.x        # suppression d'un attribut
r1.avancer()        # AttributeError: 'Robot' object has no attribute 'x'
```

Exercice 14

14.1. Comment sont gérer les droits d'accès en Python ?

14.2. Comment garantir que les attributs d'un objet auront toujours des valeurs possibles. En particulier, il faudrait contrôler la valeur de la direction qui doit toujours être un entier compris entre 0 et 3 !

14.3. Comment garantir le principe de l'accès uniforme ?