Devoir TERMINALE NSI n°1

Récursivité, POO

Terminale – spécialité NSI	**Octobre 2020**
Durée 2H	
Calculatrice interdite, mémento Python autorisé	

Les 4 exercices sont indépendants les uns des autres et peuvent être traités dans l'ordre que vous souhaiterez.

PARTIE I : POO

Exercice 1 (8 points)

Ecrire le code d'une classe `Rectangle` disposant :
- de deux attributs `longueur` et `largeur`,
- d'une méthode `perimetre(self)` qui renvoie le périmètre du rectangle,
- d'une méthode `aire(self)` qui renvoie l'aire du rectangle,
- d'une méthode `grossir(self, delta)` qui permet d'augmenter simultanément les attributs `largeur` et `longueur` du rectangle de `delta` unités,
- d'une méthode `__lt__(self, rect)` qui renvoie True si l'aire du rectangle qui appelle la méthode est plus petite que l'aire du rectangle `rect` et qui renvoie `False` sinon.

Note : lt pour less than

Figure 1 : Console d'exécution illustrant les tests de la classe

```
>>> rect_b = Rectangle(10, 20)
>>> rect_a.aire()
35
>>> rect_b.perimetre()
60
>>> rect_b.grossir(3)
```

```
class Rectangle():
    def __init__(self, long, larg):
        self.largeur = larg
        self.longueur = long

    def perimetre(self):
        return 2 * self.largeur + 2 *
self.longueur

    def aire(self):
        return self.largeur * self.longueur

    def grossir(self, delta):
        self.largeur = self.largeur + delta
        self.longueur = self.longueur +
delta

    def __lt__(self, rect):
        return self.aire() < rect.aire()
```

détail du barème :
1 attributs appelés par nom défini dans `__init__`
1 attributs appelés par `self.X`
1 sait modifier la valeur d'un attribut dans la définition de la classe
1 paramètres explicites dans `__init__`
1 affectation des paramètres explicites de `__init__` aux attributs
1 présence de `self` comme paramètre implicite des méthodes
1 méthodes appelées avec `()` dans la classe
1 sait créer une instance dans la définition de la classe
1 emploi de `return`
6 les 6 méthodes respectent la spécification (+ + 1 2 1 1)
2 respect de la syntaxe (`":"`, indentation), nom attributs et méthodes corrects

PARTIE II : Questions de cours

Exercice 2 (6 points)

1. Comment appelle-t-on également un objet d'une classe ? **une instance de classe**
2. Comment appelle-t-on le principe selon lequel certains *attributs* peuvent être partiellement ou totalement inaccessible à d'autres objets ? **c'est le principe d'encapsulation**
3. A quoi sert ou Comment est utilisé la méthode __repr__() de une classe? **C'est la redefinition de la méthode print pour un objet de la classe.**
4. Quelles sont les opérations de bases nécessaires au bon fonctionnement d'une classe ?

   Un constructeur et des getters/setters
5. On considère la file suivante : | 5 | 9 | 3 | 1 |

Après avoir enfilé un nouvel élément, elle devient

2	5	9	3	1

Quel élément sera défilé en premier ? justifier en quelques mots.

Fonctionnement en FIFO, c'est l'élément de valeur 1 qui sera defiler en premier.

La file deviendra

2	5	9	3

6. Est-il plutôt préférable de modéliser la descente d'un toboggan par des enfants par une file ou par une pile ? Justifier.

Par une file

PARTIE III : Récursivité

Exercice 3 (4 points)

1. Que faut-il obligatoirement avoir dans une fonction récursive ?
   Il faut obligatoirement une condition d'arrêt de la récursivité ou autrement dit le cas de base.

On rappelle la définition de la factorielle de n, notée n!
comme le nombre définit par 0! = 1 et sinon n! = n(n - 1)!
On nomme une fonction `factorielle(n)` qui calcule la factorielle de façon récursive.

2. Indiquer ce que valent les appels suivants :
   factorielle(0) → retourne 1
   factorielle (3) → retourne 3

3. Ecrire la fonction `factorielle(n)` qui calcule la factorielle de façon récursive.
   Vous veillerez à documenter votre fonction avec une docstring minimal, et vous accompagnerez votre fonction des tests nécessaires à la validation du bon fonctionnement de la fonction (en utilisant les `assert`)

```
def fact(n) :
    if n == 0:
        return 1
    else :
        return n * fact(n - 1)

assert( fact(6) == 720)
assert(fact(10) == 3628800)
assert(fact(0) == 1)
assert(fact(1) == 1)
print(fact(6))
```

Exercice 4 (2 points)

Ecrire la fonction récursive `puissance(x,n)` qui calcule le nombre x^n, pour tout n >= 1

Vous utiliserez le fait que

- si n est pair, $a^n = (a \times a)^{n/2}$
- sinon $a^n = a \times (a \times a)^{(n-1)/2}$

```
def puissance(a,n):
    if n == 0 :
        return 1
    #n est pair
    if (n%2==0):
        return puissance(a*a,n/2)
    #n est impair
    else :
        return a*puissance(a*a, (n-1)/2)
```