université
PARIS-SACLAY

Programmation en Python
Master 2 Réseaux Télécoms
TP 2

Les tests unitaires pour ce TP se trouvent dans les fichiers suivants :

```
testHierarchy.py
testSpecialMethods.py
testGenerator.py
testIterable.py
testComprehensions.py
```

1 Hiérarchie des classes

Ajoutez les définitions des classes manquantes dans le fichier `hierarchy.py`, en respectant l'hiérarchie suivante :

```
Mammalia
├── Prototheria
│   └── Monotremata
└── Theria
    ├── Eutheria
    │   └── Placentalia
    │       ├── Xenarthra
    │       ├── Afrotheria
    │       └── Boreoeutheria
    └── Metatheria
        └── Marsupialia
```

2 Méthodes spéciales

Dans le fichier `point.py` complétez le code des méthodes spéciales suivantes de la classe `Point` :

1. `__init__(self,x,y)`. Ce constructeur doit initialiser un objet de la classe Point, en y ajoutant les attributs `x` et `y` avec les valeurs passées par les arguments correspondants.
2. `__eq__(self,p)`. Cette méthode retournera la valeur vraie si et seulement si les coordonnées des points `self` et `p` sont identiques.
3. `__hash__(self)`. On se rappelle que pour les objets égaux (par rapport à l'opérateur `==`), la fonction de hachage doit retourner la même valeur. *Indice :* les tuples sont hachables.
4. `__str__(self)`. Cette fonction donne une représentation de l'objet lisible par un utilisateur humain. Notamment, pour le point avec les coordonnées 1 et 2, cette méthode retourne la chaîne de caractères `'(1, 2)'`.
5. `__repr__(self)`. Cette méthode donne une représentation fidèle d'un objet, permettent de le reconstituer à l'identique. Notamment, pour le point avec les coordonnées 1 et 2, cette méthode retourne la chaîne de caractères `'point.Point(1, 2)'`.

3 Suite de Fibonacci avec un générateur

Complétez le code de la fonction génératrice `fibgen(n=0)`. Cette fonction retourne un objet générateur d'une suite de Fibonacci. La génération continue tant que les valeurs de la suite sont strictement inférieures à `n`, sauf pour `n==0`, auquel cas le générateur produit une suite infinie. Par exemple, le code ci-dessous

```
for i in fibgen(500):
        print(i)
```

donne le résultat suivant :

```
1
1
2
3
5
8
13
21
34
55
89
144
233
```

377

4 Suite de Fibonacci avec une classe itérable

Dans cet exercice on créera une classe itérable représentant la suite de Fibonacci. Complétez le code de la classe itérable `Fib` et de l'itérateur correspondant `FibIterator_` dans le fichier `iterable.py`.

Dans la classe `Fib` :

1. `__init__(self,n=0)` crée un objet itérable représentant les `n` premiers éléments de la suite de Fibonacci (sans les avoir généré au préalable !). Si `n==0`, l'objet représentera une suite infinie.
2. `__iter__(self)` retourne un itérateur correspondant.
3. `__contains__(self,k)` retourne une valeur vraie si et seulement si `k` est dans la suite de Fibonacci. Pour éviter une boucle infinie, on utilisera le fait que cette suite est croissante.

Dans la classe `FibIterator_` :

1. `__init__(self,n)` crée un itérateur positionné sur le premier élément de la suite. La valeur de l'argument `n` est juste mémorisée dans un attribut.
2. `__next__(self)` retourne l'élément courant de la suite de Fibonacci et avance l'itérateur vers l'élément suivant. S'il n'y a plus d'éléments dans la suite, cette fonction lève une exception `StopIteration`.

5 Compréhensions

Dans cet exercice on utilisera le générateur de la suite de Collatz :

```
def collatz(n):
    while True:
        yield n
        if n==1:
            return
        n = 3*n+1 if n%2 else n//2
```

La *conjecture de Collatz* est l'hypothèse selon laquelle pour tout `n` entier strictement positif, cette suite est finie. A ce jour, cette conjecture a été vérifiée numériquement pour les valeurs de `n` allant jusqu'à $2^{68} \approx 2.95 \cdot 10^{20}$, mais il n'en existe toujours pas de démonstration pour tout `n` (voir Figure 1). Le code à compléter se trouve dans le fichier `comprehensions.py`.

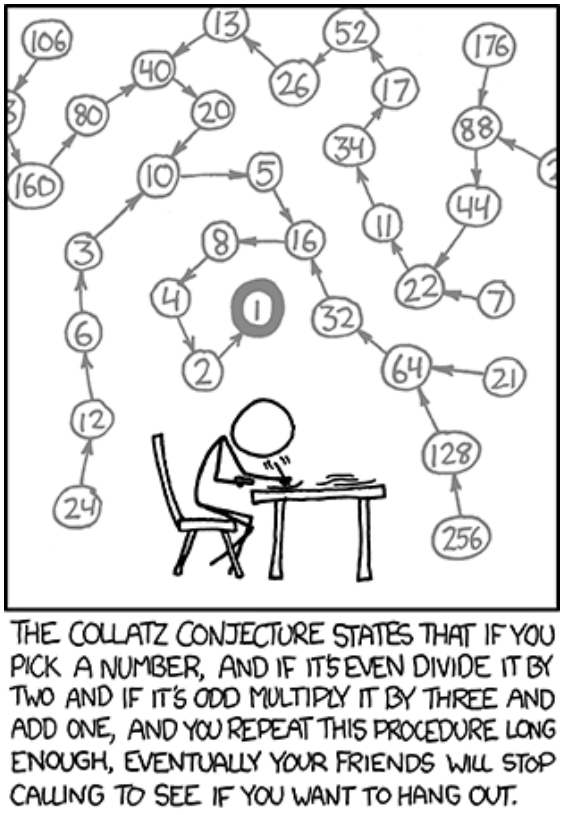

FIGURE 1 – Le dessin du site XKCD (`https://xkcd.com/710/`).

5.1 Une liste en compréhension

Complétez le code de la fonction `collatz_list(n)`, qui retournera la liste des éléments consequtifs de la suite de Collatz qui commence par `n`. Par exemple, `collatz_list(6)` donne la liste suivante :

```
[6, 3, 10, 5, 16, 8, 4, 2, 1]
```

5.2 Un ensemble en compréhension

Complétez le code de la fonction `collatz_set(n)`, qui retournera un ensemble composée des éléments de la suite de Collatz qui commence par `n`. Par exemple, `collatz_set(6)` donne l'ensemble suivant :

```
{1, 2, 3, 4, 5, 6, 8, 10, 16}
```

5.3 Un dictionnaire en compréhension

Complétez le code de la fonction `collatz_dict(n)`. Cette fonction doit retourner un dictionnaire associant à chaque entier `i` compris entre 1 et `n` inclus une liste des éléments de la suite de Collatz qui commence par `i`. Par exemple, `collatz_dict(6)` donne le dictionnaire suivant :

```
{1: [1],
 2: [2, 1],
 3: [3, 10, 5, 16, 8, 4, 2, 1],
 4: [4, 2, 1],
 5: [5, 16, 8, 4, 2, 1],
 6: [6, 3, 10, 5, 16, 8, 4, 2, 1]}
```

5.4 La longueur de la suite de Collatz

Complétez le code de la fonction `collatz_count(n)`, qui retournera la longueur de la suite de Collatz. *Indice :* on peut utiliser la fonction `sum`.