Partie 1

POO revisions : Questions à reponses courtes

1.1 Quelle est la différence entre une classe et une instance de classe ?

1.2 Définir ce qu'est un attribut de classe et une méthode de classe.

1.3 Quelle fonction est appelée lorsque l'on fait une *instanciation* ?

1.4 Pour créer la classe fruit, vous écrivez :

- class Fruit() :
- class Fruit :
- def Fruit() :

1.5 Pour créer un objet *banane*, vous écrivez :

- banane = Fruit
- banane = Fruit()
- Fruit(banane)

1.6 Quel constructeur est correct ?

```
# a
def init(self):
  x = 0
# b
def __init__(self):
  x = 0
# c
def __init__(self):
  self.x = 0
```

Partie 2

Définir une classe

2.0.1 Classe Point

En géométrie, un point est constitué de 2 coordonnées, `x` et `y`. Vous allez définir complètement la classe `Point` à partir des renseignements suivants :

2.0.2 instanciation

L'instanciation se fera avec

```
def __init__(self, coord_x, coord_y):
  self.x = ...
  self.y = ...
```

2.0.3 Méthode pour translater

La point doit posséder une méthode qui calcule ses nouvelles corrdonnées à partir d'une translation dans la direction `vx`, `vy`.

```
def translater(self, ...
  self.x = ...
  ...
```

2.1 Classe Segment

La classe `Segment` va traiter les données issues de 2 `Points`

2.1.1 instanciation

L'instanciation se fera avec

```
def __init__(self, point1, point2):
  self.p1 = ...
  self.p2 = ...
```

2.1.2 Méthode de tracé : fonction `trace`

On utilisera les méthode du module `Turtle` :

```
import Turtle
tt = Turtle() # instanciation de l'objet tt à partir de la classe Turtle
tt.up()  # lever le crayon
tt.goto(x,y) # se deplace à la position x,y
tt.down() # baisser le crayon
```

2.1.3 Méthode de calcul de distance : fonction `distance`

Ajouter une méthode de calcul de distance à partir de l'expression mathématique :

$$d = math.sqrt((p2.x - p1.x) + (p2.y - p1.y) * *2)$$

2.2 Classe `Polygone`

2.2.1 Créer la classe `Polygone`

2.2.2 Implémenter une méthode qui trace le polygone

2.2.3 Implémenter une méthode qui fait une translation du polygone puis trace celui-ci

Partie 3

Corriger des erreurs dans un script

3.1 nombre d'arguments

Voici la definition de la classe `Domino`. Il s'agit de l'implémentation d'une liste chaînée :

```
class Domino:
    def __init__(self,data, suiv):
        self.value = data
        self.suiv = suiv
```

Lorsque l'on instancie un objet de la manière suivante, la console lève une exception :

```
>>> D1 = Domino('5:5')
```

TypeError Traceback (most recent call last)

```
...
--->D1 = Domino('5:5')
TypeError: __init__() missing 1 required positional argument: 'suiv'
```

1. Modifier la fonction `__init__` pour ne plus avoir ce message d'erreur.
2. Modifier la fonction `__init__` pour que le domino possède 2 attributs numériques à la place d'un seul de type chaine de caractère : On veut instancier le domino D1 de la mnière suivante :

```
D1 = Domino(5,5)
```

3.2 self or not self?

Soit la classe Personne définie de la manière suivante :

```
class Personne:
    def __init__(self, nom, année_naissance, lieu_naissance):
        self.nom = nom
        self.année_naissance = année_naissance
        self.lieu_naissance = lieu_naissance

    def age(date)
        return date - année_naissance
```

On instancie un nouvel objet nommé *marc* de la manière suivante :

`marc = Personne(self,'Marc Assin', 1989, 'Bergerac')`

L'exception suivante est levée :

```
NameError                                 Traceback (most recent call last)
---> 10 marc = Personne(self,'Marc Assin', 1989, 'Bergerac')

NameError: name 'self' is not defined
```

Question a : Corrigez ce problème en écrivant correctement l'instruction d'instanciation.

Question b : On cherche à connaitre l'age de Marc, avec l'instruction suivante :

```
>>> marc.age(2020)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
---> 11 marc.age(2020)

TypeError: age() takes 1 positional argument but 2 were given
```

Que s'est-il passé ? Corriger ce problème en ré-écrivant la méthode de classe correspondante.

Question c : On souhaite maintenant créer un nouvel attribut `années` qui calcule et stocke l'age de l'objet dans un nouvel attribut. On ajoutera à la fonction `__init__` la ligne suivante :

```
self.années = ...
```

Compléter

Question d : Comment faudrait-il modifier la liste des attributs de `Personne` pour ajouter la liste des amis de la personne ?

3.3 Getter et Setter

Question a : Pour la classe `Personne` vue plus haut : Ajouter une méthode de classe qui permet de consulter l'année de naissance de la personne.

Question b : Ajouter une méthode de classe qui modifie son attribut `nom`

Question c : Ajouter une méthode de classe qui ajoute un ami. Et une autre qui retourne la liste d'amis.

Partie 4

jeu de Dominos - version classique

Le jeu de *Dominos* est un jeu très simple, ou, pour gagner, il faut être le premier joueur à avoir posé tous ses dominos. Une fois le premier domino placé sur la table, le joueur suivant doit à son tour poser un domino ayant le même nombre de points sur au moins un côté du domino précédemment posé.

Un domino est constitué de côtés, droite/gauche ou haut/bas selon comment la pièce sera disposée.

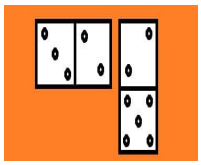

FIGURE 1 – début de partie

La disposition importe peu : il faut que la chaine reste ouverte.

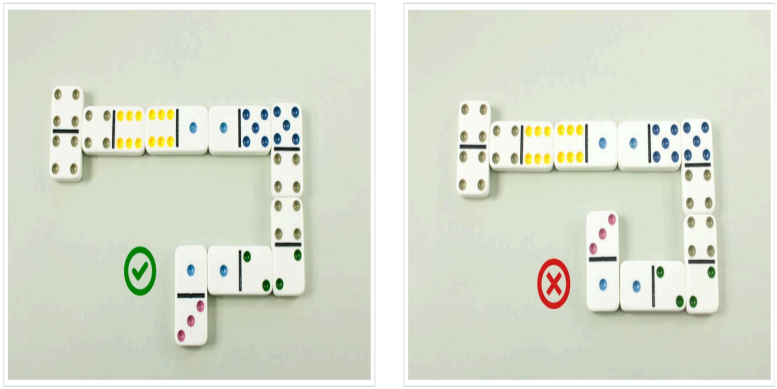

FIGURE 2 – Exemple de disposition juste (à gauche) et fausse (à droite)

On utilise la définition de classes suivantes :

```
class Domino:
    def __init__(self,val1,val2):
        self.val1 = val1
        self.val2 = val2
        self.suiv = None


class Partie:
    def __init__(self,first):
        self.first = first

    def last(self):
        M = self.first
        while not M.suiv is None:
            M = M.suiv
        return '{}:{}'.format(M.val1,M.val2)

    def atteindre_domi(self,val1,val2):
        D = self.first
        while not D.suiv is None and (D.val1,D.val2) != (val1,val2):
            D = D.suiv
        if (D.val1,D.val2) == (val1,val2):
```

```
            return D
        else:
            return self.first

    def inserer(self,D_place,D_a_inserer):
            #à completer

    def __repr__(self):
        M = self.first
        s = '{}:{} '.format(M.val1,M.val2)
            # à completer
        return s
```

Qu a. On cherche à représenter la partie de l'image de gauche (voir plus haut). Les dominos seront instanciés à l'aide des noms D1, D2, D3, ... Ecrire les instructions qui instancient tous les dominos de la partie, avec, pour chacun, leurs valeurs et le domino suivant.

Qu b. Ecrire l'instruction qui doit créer l'objet `partie1` à partir de ce plateau de jeu. (classe `Partie`)

Qu c. Compléter la méthode de classe `__repr__` qui surcharge la fonction `print`

Qu d. Commenter la méthode de classe `atteindre_domi`. A quoi sert-elle ? Quelle est sa complexité asymtotique ?

Qu e. Imaginons que l'état de la partie soit celui-ci :

```
print(partie1)
4:4 => 4:6 => 6:1 => 1:5 => 5:4 => 4:2 => 2:1 => 1:3
```

Compléter la méthode de classe **`inserer`** qui permet d'insérer un domino (double) dans la chaine de dominos à partir des instructions suivantes :

```
D = partie1.atteindre_domi(4,2)
D10 = Domino(2,2)
partie1.inserer(D,D10)
```

Le nouvel état de la partie devrait alors être :

```
print(partie1)
4:4 => 4:6 => 6:1 => 1:5 => 5:4 => 4:2 => 2:2 => 2:1 => 1:3
```