

Exercice 1

POO

Soit la classe `Personne` définie de la manière suivante :

```
1 class Personne:
2     def __init__(self, nom, annee, lieu):
3         self.nom = nom
4         self.annee_naissance = annee
5         self.lieu_naissance = lieu
6         self.amis = []
7
8     def age(self, date):
9         # retourne l'age calcule a partir de l'annee de naissance
```

1.1 Pour créer l'objet `marc`, pouvez vous écrire : (choisissez)

- `Personne = marc('Marc',1983,'Marseille')`
- `marc = new Personne('Marc',1983,'Marseille')`
- `marc = Personne('Marc',1983,'Marseille')`
- `Marc = Personne(1983,'Marseille')`

1.2 Méthodes de classe

1.2.1 Quelle instruction doit-on écrire ?

On cherche à connaître l'âge de Marc en 2020, en utilisant la notation pointée :

- `marc.age(2020)`
- `Personne.age(2020)`
- `self.age(2020)`
- autre (préciser)

a. Choisissez la bonne instruction.

b. Quelle sera la valeur retournée ?

1.2.2 Les amis de Marc

a. Ecrire une méthode de classe qui ajoute un ami à l'attribut `self.amis`

```
1     def ajoute_ami(self, ami):
2         ...
```

b. S'agit-il d'une méthode de type *Setter* ou *Getter* ?

- c. Ecrire l'instruction que ajoute gilles à la liste d'amis (depuis le *main*). On suppose que gilles est une instance de la classe `Personne`.
- d. Ecrire une méthode de la classe `Personne` qui retourne la liste d'amis. Cette liste est retournée sous forme d'une chaîne de caractères, avec le nom et l'année de naissance.

```
1     def liste_des_amis(self):
2         ...
3
4 >>> marc.liste_des_amis()
5 gilles : 1983, thierry : 1972
```

Exercice 2

Piles / Files

2.1 Comparer les 2 structures de données : la Pile et la File :

- a. les éléments sont-ils ajoutés les uns à la suite des autres, de la même manière ?
- b. y-a-t-il une méthode prévue par l'interface pour accéder facilement à l'élément au sommet ?
- c. Quelle est la différence majeure entre ces 2 structures de données ?

2.2 Classe Pile

La classe `Pile` est définie de la manière suivante :

```
1 class Pile:
2     def __init__(self):
3         self.pile = [] # on crée une liste vide lors de la création de l'
4             objet.
5
6     def push(self,data):
7         """ajoute un element au sommet de la pile
8         """
9         self.pile.append(data)
10
11    def pop(self):
12        '''Renvoie l'element sur le haut de la pile et l'enleve de la pile
13        .
14        :returns: dernier element empilé
15        :raises: renvoi une erreur si la pile est vide !
16        '''
17        # a completer
18        ...
19
20    def head(self):
21        '''retourne l'element du sommet de la pile
22        '''
23        # a completer
```

2.3 Instancier

2.3.1 Ecrire les-l'instruction-s qui créent une nouvelle pile appelée `pile_A` contenant les valeurs 0, 1, 2. (2 est le sommet).

```

1 >>>
2 >>>
3 >>>
4 >>>
5 >>> pile_A.head()
6 2

```

2.4 Méthodes de classe

2.4.1 Compléter la méthode de classe `pop` directement sur le document.

2.4.2 Ajouter un test d'assertion pour que l'on ne puisse ajouter que des valeurs entières.

Exercice 3

Listes

3.1 Comparer les 2 structures de données : la liste chaînée et la Pile :

- les éléments sont-ils ajoutés les uns à la suite des autres, de la même manière ?
- y-a-t-il une méthode prévue par l'interface pour accéder facilement au dernier élément (sommet) ?
- Y-a-t-il une opération plus facile à réaliser avec la liste chaînée qu'avec une Pile ?

3.2 Classe Domino

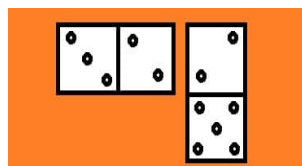


FIGURE 1 – début de partie

Les objets d'une partie de dominos sont définis par la classe Domino ci-dessous :

Voici le script de la classe Domino :

```

1 class Domino:
2     def __init__(self, data, suiv = None):
3         self.value = data
4         self.suiv = suiv
5
6     def set_value(self, data):

```

```
7     self.value = data
8
9     def set_suiv(self,D):
10        self.suiv = D
```

3.2.1 Instancier

Ecrire les instructions qui permettront de créer les objets Domino D1, D2, D3, puis d'établir une liste chaînée de dominos comme illustré ici : 5 :5 => 5 :4 => 4 :2.

```
1 >>> # creation de D1, D2, et D3
2 >>>
3 >>>
4 >>>
5 >>> # liste chainee entre ces objets
6 >>>
7 >>>
```

3.2.2 Insérer

Supposons que l'on ait déjà créé un objet domino D4, dont la valeur est '4:3', et un domino D5 de valeur '3:4'. On souhaite modifier la partie pour insérer ceux-ci dans la partie de la manière suivante : 5 :5 => 5 :4 => 4 :3 => 3 :4 => 4 :2

Ecrire les instructions à ajouter au programme :

```
1 >>>
2 >>>
3 >>>
```

Exercice 4

Bac 2022 Metropole1/ Exercice 5

Cet exercice porte sur la Programmation Orientée Objet.



FIGURE 2 – début de partie

Les participants à un jeu de LaserGame sont répartis en équipes et s'affrontent dans ce jeu de tir, revêtus d'une veste à capteurs et munis d'une arme factice émettant des infrarouges.

Les ordinateurs embarqués dans ces vestes utilisent la programmation orientée objet pour modéliser les joueurs. La classe Joueur est définie comme suit :

```
1 class Joueur:
2     def __init__(self, pseudo, identifiant, equipe):
3         """constructeur """
4         self.pseudo = pseudo
5         self.equipe = equipe
6         self.id = identifiant
7         self.nb_de_tirs_emis = 0
8         self.liste_id_tirs_recus = []
9         self.est_actif = True
10
11    def tire(self):
12        """méthode déclenchée par l'appui sur la gachette"""
13        if self.est_actif == True:
14            self.nb_de_tirs_emis = self.nb_de_tirs_emis + 1
15
16    def est_determine(self):
17        """methode qui renvoie True si le joueur réalise un
18        grand nombre de tirs"""
19        return self.nb_de_tirs_emis > 500
20
21    def subit_un_tir(self, id_recu):
22        """méthode déclenchée par les capteurs de la
23        veste"""
24        if self.est_actif == True:
25            self.est_actif = False
26            self.liste_id_tirs_recus.append(id_recu)
```

4.1 Question 1

Parmi les instructions suivantes, recopier celle qui permet de déclarer un objet `joueur1`, instance de la classe `Joueur`, correspondant à un joueur dont le pseudo est `Sniper`, dont l'identifiant est `319` et qui est intégré à l'équipe `A` :

- **Instruction 1** : `joueur1 = ["Sniper", 319, "A"]`
- **Instruction 2** : `joueur1 = new Joueur["Sniper", 319, "A"]`
- **Instruction 3** : `joueur1 = Joueur("Sniper", 319, "A")`
- **Instruction 4** : `joueur1 = Joueur{"pseudo":"Sniper", "id":319, "equipe":"A"}`

4.2 Question 2

La méthode `subit_un_tir` réalise les actions suivantes :

Lorsqu'un joueur actif subit un tir capté par sa veste, l'identifiant du tireur est ajouté à l'attribut `liste_id_tirs_recus` et l'attribut `est_actif` prend la valeur `False` (le joueur est désactivé). Il doit alors revenir à son camp de base pour être de nouveau actif.

- Écrire la méthode `redevenir_actif` qui rend à nouveau le joueur actif uniquement s'il était précédemment désactivé.
- Écrire la méthode `nb_de_tirs_recus` qui renvoie le nombre de tirs reçus par un joueur en utilisant son attribut `liste_id_tirs_recus`.

4.3 Question 3

Lorsque la partie est terminée, les participants rejoignent leur camp de base respectif où un ordinateur, qui utilise la classe Base, récupère les données. La classe Base est définie par :

- ses attributs :
 - `equipe` : nom de l'équipe (str). Par exemple, "A" ,
 - `liste_des_id_de_l_equipe` qui correspond à la liste (list) des identifiants connus des joueurs de l'équipe,
 - `score` : score (int) de l'équipe, dont la valeur initiale est 1000 ;
- ses méthodes :
 - `est_un_id_allie` qui renvoie True si l'identifiant passé en paramètre est un identifiant d'un joueur de l'équipe, False sinon,
 - `incremente_score` qui fait varier l'attribut `score` du nombre passé en paramètre,
 - `collecte_information` qui récupère les statistiques d'un participant passé en paramètre (instance de la classe Joueur) pour calculer le score de l'équipe.

```

1 def collecte_information(self, participant):
2     if participant.equipe == self.equipe : # test 1
3         for id in participant.liste_id_tirs_recus:
4             if self.est_un_id_allie(id): # test 2
5                 self.incremente_score(-20)
6             else:
7                 self.incremente_score(-10)

```

- A. Indiquer le numéro du test (**test 1** ou **test 2**) qui permet de vérifier qu'en fin de partie un participant égaré n'a pas rejoint par erreur la base adverse.
- B. Décrire comment varie quantitativement le score de la base lorsqu'un joueur de cette équipe a été touché par le tir d'un coéquipier.

On souhaite accorder à la base un bonus de 40 points pour chaque joueur particulièrement déterminé (qui réalise un grand nombre de tirs).

4.4 Question 4 :

Recopier et compléter, en utilisant les méthodes des classes Joueur et Base, les 2 lignes de codes suivantes qu'il faut ajouter à la fin de la méthode `collecte_information` :

```

1 ..... #si le participant réalise un grand nombre de tirs
2 ..... #le score de la Base augmente de 40

```