IUT2 Université Grenoble Alpes
Département STID

Examen écrit

Programmation objet – R3.VCOD.10

Lundi 13 novembre 2023

Documents interdits - Calculatrice interdite

Répondez sur cette copie

NOM PRÉNOM :

---

1. Choisissez parmi les propositions suivantes les 3 composantes d'un objet.
   ☐ État ☐ Classe ☐ Héritage ☐ Identité ☐ Sous-classe ☐ Comportement

2. Reliez entre eux les 4 avantages de la Programmation Orientée Objet et leurs définitions

Modularité	•	• Les objets peuvent gérer le niveau de visibilité de leurs composantes.
Abstraction	•	• Les objets sont faciles à manipuler.
Encapsulation	•	• Les objets représentent des concepts du monde réel.
Ré-utilisabilité	•	• Les objets sont facilement réutilisables et extensibles.

3. L'appel à un constructeur permet de définir :
   ☐ Une classe ☐ Un objet ☐ Une sous-classe

4. Cochez les cases suivantes si l'expression est vraie :
   ☐ Une classe est une instance d'un objet
   ☐ Un objet hérite d'une classe
   ☐ Un objet instancie une classe
   ☐ Une classe est un modèle pour créer des objets

5. Cochez les cases suivantes si l'expression est vraie :
   ☐ Deux objets différents peuvent avoir la même identité (la même référence)
   ☐ Deux instances d'objet d'une même classe ont nécessairement les mêmes valeurs d'attributs
   ☐ Deux instances d'objet d'une même classe ont nécessairement le même comportement
   ☐ Deux instances d'objets de 2 sous-classes de la même classe mère ont un comportement identique

6. Les expressions suivantes sont-elles vraies ou fausses ?
   a. Une classe peut ne pas avoir d'instances :..... ☐ Vrai ☐ Faux
   b. En Python, une classe peut avoir plusieurs sous-classes : . ☐ Vrai ☐ Faux
   c. En Python, une classe peut hériter de plusieurs classes : . ☐ Vrai ☐ Faux
   d. Toute sous-classe a au moins un attribut et/ou une méthode qui diffèrent de la classe dont elle hérite :..... ☐ Vrai ☐ Faux

7. Soit le code suivant permettant de définir la classe Personne

```
class Personne:
    def __init__(self, nom : str, prenom : str, adresse : str):
        self.firstname = prenom
        self.lastname = nom
        self.address = adresse


    def get_firstname(self):
        return self.firstname


    def get_lastname(self):
        return self.lastname


    def get_address(self):
        return self.address


    def change_firstname(self, nouv_prenom : str):
        self.firstname = nouv_prenom


    def change_lastname(self, nouv_nom : str):
        self.lastname = nouv_nom


    def change_address(self, nouv_adresse : str):
        self.address = nouv_adresse
```

Soit le code suivant :

```
p1 = Personne("Durant", "Paul", "Grenoble")
p2 = Personne("Dupont", "Jeannine", "SMH")

p2.change_address(p1.get_address())
p2.change_lastname(p1.get_lastname())
```

Indiquez le résultat des instructions suivantes :

- `print(type(p1))` : .....
- `print(isinstance(p2,Personne))` : .....
- `print(p1.lastname)` : .....
- `print(p2.get_lastname())` : .....

8. Soit le code suivant permettant de définir les classes Etudiant et Enseignant, sous-classes de la classe Personne :

```
class Etudiant(Personne):
    def __init__(self, nom : str, prenom : str, adresse : str, age : int):
        self.firstname = prenom
        self.lastname = nom
        self.address = adresse
        self.age = age


    def get_age(self):
        return self.age


    def change_age(self, nouv_age : int):
        self.age = nouv_age


    def print_infos(self):
        print("Je suis un étudiant", self.get_firstname(), self.get_lastname(),
                self.get_address(), self.age)


class Enseignant(Personne):
            def __init__(self, nom : str, prenom : str, adresse : str, stat : str):
                self.firstname = prenom
                self.lastname = nom
                self.address = adresse
                self.statut = stat
```

```
            def get_statut(self):
                return self.statut


            def change_statut(self, nouv_statut : str):
                self.statut = nouv_statut


            def print_infos(self):
                print("Je suis un enseignant", self.get_firstname(),
                      self.get_lastname(), self.get_address(), self.statut)
```

Soit le code suivant :

```
e1 = Etudiant("Durand", "Théo", "GNB", 18)
e2 = Enseignant("Dupont", "Jeannine", "SMH", "Professeur")
```

Écrivez ci-dessous les instructions permettant d'afficher les informations de Théo Durand et Jeannine Dupont, et donnez le résultat attendu.

.....

.....

.....

.....

.....

.....

.....

.....

9. On s'intéresse à présent à du code simulant des personnages de RPG. On souhaite que l'exécution des instructions montrées dans le code ci-dessous à gauche produise le résultat montré à droite :

```
arthur = rpg.PersoRPG("Arthur", 100, 100)

arthur.print_info()

sarah = rpg.PersoRPG("Sarah", 200, 50)

sarah.print_info()

mini = rpg.Warrior("Minimoy")

mini.print_info()
```

Résultat attendu :

```
Perso Arthur with 100/100 hp and 100 attack damage
Nb de personnage(s) : 1, en comptant Arthur
Perso Sarah with 200/200 hp and 50 attack damage
Nb de personnage(s) : 2, en comptant Sarah
Warrior Minimoy with 120/120 hp and 20 attack damage
Nb de personnage(s) : 3, en comptant Minimoy
```

Les codes des classes PersoRPG, Warriors et Archers sont donnés ci-après. Pour obtenir le résultat attendu, on doit ajouter **un attribut de classe** nommé **NBPERSOS** initialisé à 0 et comptant le nombre de personnages total du jeu (PersoRPG, Warriors et Archers) et gérer correctement les affichages d'informations.

Indiquez proprement les modifications à apporter dans le code des classes donné ci-dessous pour **déclarer l'attribut NBPERSOS**, le **mettre à jour** et **l'utiliser pour afficher** les informations attendues. Écrivez directement les instructions requises aux bons endroits dans le code ci-dessous.

```
class PersoRPG:

    def __init__(self, name, max_hp, attack_dmg):

        self.name = name

        self.max_hp = max_hp

        self.current_hp = self.max_hp

        self.attack_dmg = attack_dmg
```

```
    def get_current_hp(self):

        return self.current_hp

    def set_current_hp(self, new_current_hp):

        self.current_hp = new_current_hp

    def is_alive(self):

        if self.current_hp > 0:

            return True

        else:

            return False

    def print_info(self):

        print("Perso %s with %s/%s hp and %s attack damage" % (self.name, self.current_hp,

              self.max_hp, self.attack_dmg))



class Warrior(PersoRPG):

    def __init__(self, name):

        self.name = name

        self.max_hp = 120

        self.current_hp = self.max_hp

        self.attack_dmg = 20

    def print_info(self):

        print("Warrior %s with %s/%s hp and %s attack damage" % (self.name,

             self.current_hp,self.max_hp, self.attack_dmg))



class Archer(PersoRPG):

    def __init__(self, name):

        self.name = name

        self.max_hp = 20

        self.current_hp = self.max_hp
```

```
        self.attack_dmg = 50

    def print_info(self):

        print("Archer %s with %s/%s hp and %s attack damage" % (self.name,

              self.current_hp,self.max_hp, self.attack_dmg))
```

10. Soit le code suivant, suite de la question 9 :

```
def combat(perso1, perso2):
    turn_to_attack = 1
    while perso1.is_alive() and perso2.is_alive():
        if turn_to_attack == 1:
            perso2.set_current_hp(perso2.get_current_hp() - perso1.attack_dmg)
            turn_to_attack = 2
        elif turn_to_attack == 2:
            perso1.set_current_hp(perso1.get_current_hp() - perso2.attack_dmg)
            turn_to_attack = 1
    if perso1.is_alive():
        print("%s won the fight and has %s hp remaining!" % (perso1.name, perso1.get_current_hp()))
    elif perso2.is_alive():
        print("%s won the fight and has %s hp remaining!" % (perso2.name, perso2.get_current_hp()))
    else:
        print("Both characters are dead")
```

Quel est le résultat du code suivant (répondez à droite) :

```
arthur = rpg.PersoRPG("Arthur", 100, 100)
mini = rpg.Warrior("Minimoy")
combat(mini, arthur)
```

.....
.....
.....
.....

11. On souhaite connaitre pour tout personnage son dernier adversaire. Que proposez-vous comme modifications pour prendre en compte ce nouveau besoin **et** afficher, comme dans l'exemple ci-dessous, que, suite au combat entre Arthur et Minimoy, Minimoy est le dernier adversaire de Arthur et inversement.

```
Nom du dernier adversaire de Arthur : Minimoy
Nom du dernier adversaire de Minimoy : Arthur
```

Indiquez ci-dessous les modifications nécessaires pour faire cela en précisant dans quelles classes et/ou quelles méthodes et/ou quelle partie du programme principal ces modifications doivent être réalisées. Ecrire le code correspondant sera valorisé.