

Devoir surveillé : Programmation

L1 MPC1

29 avril 2026 - Durée: 2h

On rappelle qu'aucun document, ni équipement électrique ou électronique n'est autorisé.

Cependant l'usage d'une boule en cristal ou d'un tarot divinatoire sera cependant toléré.

Les réponses sont à donner directement sur le sujet. Un espace est réservé pour chaque réponse.

Nom :	
Prénom :	

Barème. Pour chaque question :

- ne pas répondre donne 0 point,
- répondre de façon exacte donne : \mathbf{p} points
- répondre de façon inexacte¹ :
 - pour une réponse de type VRAI/FAUX : $-\mathbf{p}$ points,
 - pour une réponse libre : $-\mathbf{p}$ points,
 - pour une réponse de type 1 parmi 3 : $-\frac{\mathbf{p}}{2}$ points.

¹le nombre de points négatifs varie pour que si l'on répond de façon aléatoire l'espérance soit nulle

1

Difficulté : ★ ☆ ☆

Question :

Je permets à un objet d'être différent d'un autre objet de la même classe. Je suis :

Cochez l'unique bonne réponse :

- ① un attribut
- ② une méthode
- ③ une constante de classe

2

Difficulté : ★ ☆ ☆

Question :

Une méthode privée peut être utilisée par une autre méthode définie dans la même classe.

Cochez l'unique bonne réponse :

- ① faux
- ② vrai

3

Difficulté : ★ ☆ ☆

Question :

Une méthode privée peut être utilisée par un attribut défini dans la même classe.

Cochez l'unique bonne réponse :

- ① vrai
- ② faux

4

Les deux classes de la figure 1 sont liées.

4.1

Difficulté : ★ ★ ☆ Question :

Ce lien est :

Cochez l'unique bonne réponse :

- ① Une dépendance de `StateObject` pour `Store`
- ② Une dépendance mutuelle des deux classes
- ③ Une dépendance de `Store` pour `StateObject`

4.2

Difficulté : ★ ★ ☆ Question :

Ce lien est appelé :

Cochez l'unique bonne réponse :

- ① Agrégation
- ② Composition
- ③ Héritage

4.3

Difficulté : ★ ★ ☆ Question :

Il est représenté par :

Cochez l'unique bonne réponse :

- ① ◇
- ② ◆
- ③ △

5

On utilise les deux classes de la figure 1. Trois Propositions de code sont données :

```
1.      state = StateObject()
        state.set_state(1)
        store = Store(state)
        store.saved_state = state.get_state()
```

```
2.      state = StateObject()
        state.set_state(1)
        store = Store(state)
```

```
3.      state = StateObject(1)
        store = Store(state)
```

5.1

Difficulté : ★ ★ ☆ Question :

Une seule des 3 propositions est correcte par rapport au schéma UML. Laquelle ?

Cochez l'unique bonne réponse :

- ① proposition 2
- ② proposition 1
- ③ proposition 3

5.2

Difficulté : ★ ★ ★ Question :

Une seule des 3 propositions est incorrecte en python. Laquelle ?

Cochez l'unique bonne réponse :

- ① proposition 1
- ② proposition 2
- ③ proposition 3

6

Difficulté : ★ ☆ ☆

Question :

Que représente le premier paramètre (souvent appelé `self`) dans une méthode codée en python ?

Cochez l'unique bonne réponse :

- ① la classe à laquelle la méthode appartient
- ② c'est un paramètre comme un autre sans signification particulière
- ③ l'objet qui a appelé la méthode

7

Difficulté : ★ ☆ ☆

Question :

En UML, la description d'une méthode sans paramètre doit posséder des parenthèses (eg: `ma_methode()`)

Cochez l'unique bonne réponse :

- ① vrai
- ② faux

8

Difficulté : ★ ★ ☆

Question :

En python, pour la méthode définie telle que :

```
def ma_methode(self , param=0):  
    # corps de la methode
```

Cochez l'unique bonne réponse :

- ① la valeur par défaut de `param` est 0
- ② `param` vaut 0 dans tous les cas
- ③ la méthode n'a pas de paramètre

9

Difficulté : ★ ★ ★

Question :

En python, pour la classe définie telle que :

```
class GreenCarpet :  
    def __init__(self):  
        self.dices = tuple(Dice() for i in range(5))
```

La relation en `GreenCarpet` et `Dice` est :

Cochez l'unique bonne réponse :

- ① un héritage
- ② une agrégation
- ③ une composition

10

Difficulté : ★ ★ ☆

Question :

En python, pour la classe définie telle que :

```
class DiceStat(Dice):  
    def __init__(self, value=1):  
        super().__init__(value)  
        self.stat = dict()
```

La ligne commençant par `super` signifie :

Cochez l'unique bonne réponse :

- ① un appel à la méthode `__init__` pour un objet de type `super`
- ② un appel récursif à la méthode `__init__` que l'on est entrain de définir
- ③ un appel à la méthode `__init__` de la classe `Dice`

11

On considère le diagramme UML de la figure 2

11.1

Difficulté : ★ ★ ★

Question :

Liens entre les trois classes :

Cochez l'unique bonne réponse :

- ① la classe **Student** hérite de la classe **Person** et il y a une relation d'association (une agrégation ou une composition) entre la classe **Group** et la classe **Student**
- ② la classe **Student** hérite de la classe **Person** et la classe **Group** hérite de la classe **Student**
- ③ il y a une relation d'association entre la classe **Person** et la classe **Student** (une agrégation ou une composition) et la classe **Group** hérite de la classe **Student**

11.2

Difficulté : ★ ★ ☆

Question :

pour une instance de la classe **Student**

Cochez l'unique bonne réponse :

- ① l'id est uniquement décrite par son **name** et son **firstname**
- ② l'id est uniquement décrite par son **ine_number**
- ③ l'id est uniquement décrite par son **ine_number**, son **name** et son **firstname**

12

Difficulté : ★ ☆ ☆

Question :

Le nom **saved_state** de la classe **Store** du diagramme de la figure 1 est :

Cochez l'unique bonne réponse :

- ① une méthode
- ② une fonction
- ③ un attribut

13

La figure 3 montre le diagramme UML du design pattern *Observer*. Les objets de la classe `Observer` s'inscrivent à un sujet (via la méthode `add`) pour être prévenus de chaque changement d'état (via la méthode `notify`). Tous les codes demandés sont à écrire en python.

13.1

Difficulté : ★ ☆ ☆

Question :

Quel est la fonction de ce couple de classe ?

Cochez l'unique bonne réponse :

- ① permet aux objets observant un sujet de le modifier à tout moment
- ② permet au sujet de modifier les objets qui l'observent
- ③ permet aux objets observant un sujet d'être informé des changements d'un sujet par celui-ci

13.2

Difficulté : ★ ★ ☆

Question :

Un design pattern est un assemblage de classes permettant de résoudre un problème courant en programmation. Proposez un cas d'utilisation du design pattern *Observer* (dans le cadre d'une interface graphique par exemple).

Écrivez votre réponse :

13.3

Difficulté : ★ ★ ☆

Question :

Écrivez la méthode `add` de la classe `Subject`, qui ajoute l'objet passé en paramètre à la liste des éléments observant le sujet.

Écrivez votre réponse :

13.4

Difficulté : ★ ★ ★

Question :

Écrivez la méthode `update` de la classe `Observer` qui place dans `subject_state` l'état courant d'objet `subject` de la classe `Subject` passé en paramètre.

Écrivez votre réponse :

13.5

Difficulté : ★ ★ ★ S

Question :

Écrivez la méthode `notify` de `Subject` qui permet la mise à jour des éléments observant le `Subject`

Écrivez votre réponse :

14

Difficulté : ★ ☆ ☆

Question :

Quand fait-on un appel à une méthode `__init__` ?

Cochez l'unique bonne réponse :

- ① lors de la création d'une classe
- ② lors de la création d'un attribut
- ③ lors de la création d'une instance de la classe

15

Difficulté : ★ ★ ☆

Question :

Donnez le diagramme UML d'une classe A ayant les propriétés suivantes :

- un attribut att_1 , de type réel, dont l'accès est protégé et qui a comme valeur initiale 3.14.
- un attribut att_2 , librement accessible qui est une série de chaînes de caractères.
- une méthode $delta$ de visibilité protégée qui utilise un entier n (non modifié), un objet O de la classe B qu'elle modifie, et qui retourne une valeur booléenne. On ne représentera pas la classe B sur le diagramme.

Écrivez votre réponse :

16

Difficulté : ★ ★ ★

Question :

Sur un ordinateur, les fichiers sont organisés en *répertoires*. Un répertoire est *quelque chose* qui contient des fichiers (élémentaires) ou d'autres répertoires. Représenter la notion de répertoire par un diagramme UML.

Écrivez votre réponse :



17

Difficulté : ★ ★ ★ S

Question :

Si C est une classe, donnez la différence entre $c = C$ et $c = C()$

Écrivez votre réponse :



18

Difficulté : ★ ★ ☆

Question :

Complétez la classe **Student** de la figure 2 pour que l'on puisse connaître et modifier la dernière note d'informatique d'un étudiant.

Écrivez votre réponse :

19

19.1

Difficulté : ★ ★ ★

Question :

Proposez le diagramme UML d'une classe **Conversion** dont l'initialisation prend 3 paramètres :

- la monnaie à convertir,
- la monnaie dans laquelle convertir,
- un réel représentant le taux de conversion (1 unité de la monnaie à convertir correspond à *taux* unité de la devise dans laquelle convertir).

Cette classe doit avoir une méthode permettant de connaître le nombre d'unité de la monnaie dans laquelle convertir à partir d'un nombre d'unité de la monnaie à convertir.

Écrivez votre réponse :

19.2

Difficulté : ★ ★ ★ S

Question :

Implémentez la classe en python

Écrivez votre réponse :

20

Difficulté : ★ ★ ★ S

Question :

A quoi pourrait bien servir le design pattern de la figure 4 ?

Écrivez votre réponse :

21

Difficulté : ★ ★ ☆

Question :

On considère une classe Point permettant de créer des points du plan à partir de leurs deux coordonnées : x et y. cette classe contient une méthode add() réalisant l'addition de ces deux points. Comment doit-on décrire cette méthode en UML?

Cochez l'unique bonne réponse :

- ① add(self, point : Point): Point
- ② add(point: Point): Point
- ③ add(x : double, y: double): Point

Annexes

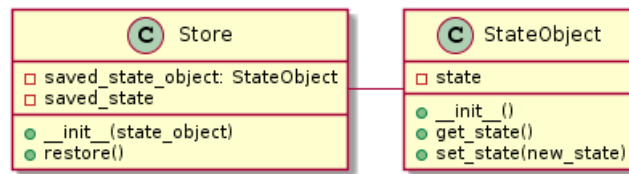


Figure 1: Le diagramme UML des classes Store et StateObject.

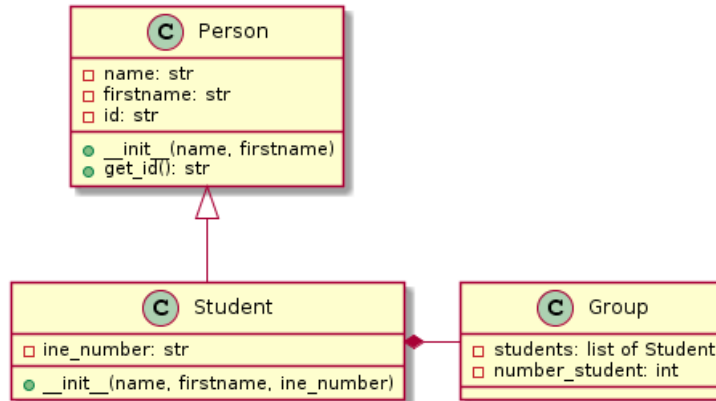


Figure 2: Le diagramme UML des classes Person, Student et Group.

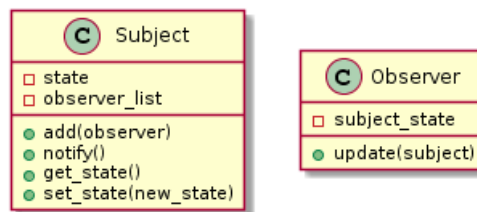


Figure 3: Le diagramme UML des classes Subject et Observer.

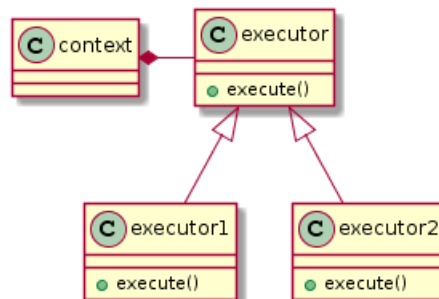


Figure 4: Un design pattern bien utile.