

Commencé le	mardi 14 février 2023, 14:19
État	Terminé
Terminé le	mardi 28 février 2023, 13:49
Temps mis	13 jours 23 heures
Points	15,00/15,00
Note	10,00 sur 10,00 (100%)

Question 1

Correct  
Note de 1,00 sur 1,00

Créez une classe Coord et son constructeur contenant les coordonnées d'un élément sur la carte du jeu). Cette classe aura deux **attributs d'instance: x et y**.

Par exemple:

Test	Résultat
c = Coord(1,2)	1
print(c.x)	2
print(c.y)	3
## directement	-4
print(Coord(3,-4).x)	
print(Coord(3,-4).y)	

Réponse : (régime de pénalités : 0 %)

```
1 class Coord:
2     def __init__(self, x = 0, y = 0):
3         self.x = x
4         self.y = y
```

	Test	Résultat attendu	Résultat obtenu	
✓	c = Coord(1,2) print(c.x) print(c.y) ## directement print(Coord(3,-4).x) print(Coord(3,-4).y)	1 2 3 -4	1 2 3 -4	✓

Tous les tests ont été réussis ! ✓

► Montrer / masquer la solution de l'auteur de la question (Python3)

Correct  
Note pour cet envoi : 1,00/1,00.

Question 2

Correct

Note de 1,00 sur 1,00

Pour une classe, de la même manière que l'on a redéfini la représentation des instances, on peut redéfinir l'opérateur `==`, en redéfinissant la méthode `__eq__(self, other)`. Pour un objet `a`, par défaut, `a == b` est vrai si seulement si `a is b`.

Redéfinissez l'opérateur `==` pour `Coord`, qui doit valoir `True` ssi les deux objets ont les mêmes coordonnées.

Par exemple:

Test	Résultat
<code>print(Coord(3,4) == Coord(3,0))</code>	False
<code>print(Coord(0,3) == Coord(0,3))</code>	True
<code>print(Coord(3,4) == Coord(-3,4))</code>	False

Réponse : (régime de pénalités : 0 %)

```
1 class Coord(object):
2     def __init__(self, x = 0, y = 0):
3         self.x = x
4         self.y = y
5     def __eq__(self, other):
6         return self.x == other.x and self.y == other.y
```

	Test	Résultat attendu	Résultat obtenu	
✓	<code>print(Coord(3,4) == Coord(3,0))</code> <code>print(Coord(0,3) == Coord(0,3))</code> <code>print(Coord(3,4) == Coord(-3,4))</code>	False True False	False True False	✓

Tous les tests ont été réussis ! ✓

► Montrer / masquer la solution de l'auteur de la question (Python3)

Correct

Note pour cet envoi : 1,00/1,00.

Question 3

Correct

Note de 1,00 sur 1,00

Dans la classe Coord ajoutez une représentation des instances de la forme "<x,y>".

Par exemple:

Test	Résultat
print(Coord(3,4))	<3,4>
print(Coord(0,-3))	<0,-3>

Réponse : (régime de pénalités : 0 %)

```
1 class Coord(object):
2     def __init__(self, x = 0, y = 0):
3         self.x = x
4         self.y = y
5
6     def __repr__(self):
7         return f"<{self.x},{self.y}>"
8
9     def __eq__(self, other):
10        return self.x == other.x and self.y == other.y
```

	Test	Résultat attendu	Résultat obtenu	
✓	print(Coord(3,4)) print(Coord(0,-3))	<3,4> <0,-3>	<3,4> <0,-3>	✓

Tous les tests ont été réussis ! ✓

► Montrer / masquer la solution de l'auteur de la question (Python3)

Correct

Note pour cet envoi : 1,00/1,00.

Question 4

Correct

Note de 1,00 sur 1,00

Vous pouvez surcharger (redéfinir) la méthode `__add__(self, other)` dans une classe pour définir la valeur renvoyée par l'opérateur `+` avec `other` une instance de cette classe.

Dans la classe **Coord**, définissez `__add__` afin de réaliser (sans effet de bord) l'addition de deux coordonnées.

Par exemple:

Test	Résultat
<code>print(Coord(3,5) + Coord(5,5))</code>	<code>&lt;8,10&gt;</code>
<code>print(Coord(0,3) + Coord(0,-3))</code>	<code>&lt;0,0&gt;</code>

Réponse : (régime de pénalités : 0 %)

```
1 class Coord(object):
2     def __init__(self, x = 0, y = 0):
3         self.x = x
4         self.y = y
5
6     def __repr__(self):
7         return f"<{self.x},{self.y}>"
8
9     def __eq__(self, other):
10        return self.x == other.x and self.y == other.y
11
12    def __add__(self, other):
13        return Coord(self.x+other.x, self.y+other.y)
```

	Test	Résultat attendu	Résultat obtenu	
✓	<code>print(Coord(3,5) + Coord(5,5))</code> <code>print(Coord(0,3) + Coord(0,-3))</code>	<code>&lt;8,10&gt;</code> <code>&lt;0,0&gt;</code>	<code>&lt;8,10&gt;</code> <code>&lt;0,0&gt;</code>	✓
✓	<code># pas d'effet de bord</code> <code>c = Coord(0,1)</code> <code>d = Coord(5,5)</code> <code>print(c + d)</code> <code>print(c)</code> <code>print(d)</code>	<code>&lt;5,6&gt;</code> <code>&lt;0,1&gt;</code> <code>&lt;5,5&gt;</code>	<code>&lt;5,6&gt;</code> <code>&lt;0,1&gt;</code> <code>&lt;5,5&gt;</code>	✓
✓	<code>print((Coord(2, 3) + Coord(1, 1)).x)</code> <code>print(isinstance(Coord(2, 3) + Coord(1, 1), Coord))</code> <code># __add__ doit renvoyer un objet Coord !</code>	3 True	3 True	✓

Tous les tests ont été réussis ! ✓

► Montrer / masquer la solution de l'auteur de la question (Python3)

Correct

Note pour cet envoi : 1,00/1,00.

Question 5

Correct

Note de 1,00 sur 1,00

Créez une classe **Map**. Cette classe permettra de placer les éléments du jeu dans une matrice.

```
--- x -->
  0 1 2 3 4
|  0 . . . . .
|  1 . . . . .
y  2 . . . . .
|  3 . . . . .
v  4 . . . . .
```

La classe Map a *les attributs de classe* suivants:

- **ground** : une case de terrain vide (le caractère point '.')
- **dir** : un dictionnaire des directions associées aux touches 'z' (haut), 's' (bas), 'q' (gauche) et 'd' (droite)

Les directions seront définies par des *coordonnées* relatives (par exemple 'z' correspond à <0,-1>)

Par exemple:

Test	Résultat
print(Map.ground)	.
print(Map.dir)	{'z': <0,-1>, 's': <0,1>, 'd': <1,0>, 'q': <-1,0>}
print(Map.dir['z'].y)	-1

Réponse : (régime de pénalités : 0 %)

```
1 class Coord(object):
2     def __init__(self, x = 0, y = 0):
3         self.x = x
4         self.y = y
5
6     def __repr__(self):
7         return f"<{self.x},{self.y}>"
8
9     def __eq__(self, other):
10        return self.x == other.x and self.y == other.y
11
12    def __add__(self, other):
13        return Coord(self.x+other.x, self.y+other.y)
14
15 class Map(object):
16     ground = "."
17     dir = {
18         'z' : Coord(0, -1),
19         's' : Coord(0, 1),
20         'd' : Coord(1, 0),
21         'q' : Coord(-1, 0)
22     }
```

	Test	Entrée	Résultat attendu	Résultat obtenu	
✓	print(Map.ground) print(Map.dir) print(Map.dir['z'].y)		. {'z': <0,-1>, 's': <0,1>, 'd': <1,0>, 'q': <-1,0>} -1	. {'z': <0,-1>, 's': <0,1>, 'd': <1,0>, 'q': <-1,0>} -1	✓
✓	print(type(Map) is type) # si ce test échoue c'est que Map n'est pas une classe (mais une variable) !	True	True	True	✓

Tous les tests ont été réussis ! ✓

► **Montrer / masquer la solution de l'auteur de la question (Python3)**

Correct

Note pour cet envoi : 1,00/1,00.

Note de 1,00 sur 1,00

- **size** (par défaut 5) : largeur de la matrice
- **pos** (par défaut <1,1>) : coordonnées de départ
- **hero** (par défaut '@') : représentation du joueur

- **mat** : la matrice de taille `size x size` avec toutes les cases initialisées à **Map.ground**
- **elem** : un dictionnaire associant les éléments présents sur le terrain avec leurs coordonnées

Test	Résultat
<pre>print(Map(size=3)._mat) print(Map()._elem)</pre>	<pre>[[ '.', '.', '.' ], [ '.', '@', '.' ], [ '.', '.', '.' ]] { '@': &lt;1,1&gt; }</pre>

```

1 class Coord(object):
2     def __init__(self, x = 0, y = 0):
3         self.x = x
4         self.y = y
5
6     def __repr__(self):
7         return f"<{self.x},{self.y}>"
8
9     def __eq__(self, other):
10        return self.x == other.x and self.y == other.y
11
12    def __add__(self, other):
13        return Coord(self.x+other.x, self.y+other.y)
14
15 class Map(object):
16     ground = "."
17     dir = {
18         'z' : Coord(0, -1),
19         's' : Coord(0, 1),
20         'd' : Coord(1, 0),
21         'q' : Coord(-1, 0)
22     }
23
24     def __init__(self, size=5, pos=Coord(1, 1), hero="@"):
25         self.size = size
26         self.pos = pos
27         self.hero = hero
28         self._mat = [[Map.ground for i in range(self.size)] for j in range(self.size)]
29         self._mat[pos.x][pos.y] = self.hero
30         self._elem = {self.hero: self.pos}

```

	Test	Résultat attendu	Résultat obtenu	
✓	<pre>print(Map(size=3)._mat) print(Map()._elem)</pre>	<pre>[[['.', '.', '.'], ['.', '@', '.'], ['.', '.', '.']] {'@': &lt;1,1&gt;}</pre>	<pre>[[['.', '.', '.'], ['.', '@', '.'], ['.', '.', '.']] {'@': &lt;1,1&gt;}</pre>	✓
✓	<pre>print(Map(pos=Coord(4,4),hero='X')._mat) print(Map(pos=Coord(4,4),hero='X')._elem)</pre>	<pre>[[['.', '.', '.', '.', '.'], ['.', '.', '.', '.', '.'], ['.', '.', '.', '.', '.'], ['.', '.', '.', '.', '.'], ['.', '.', '.', '.', '.', 'X']] {'X': &lt;4,4&gt;}</pre>	<pre>[[['.', '.', '.', '.', '.'], ['.', '.', '.', '.', '.'], ['.', '.', '.', '.', '.'], ['.', '.', '.', '.', '.'], ['.', '.', '.', '.', '.', 'X']] {'X': &lt;4,4&gt;}</pre>	✓
✓	<pre>Map.ground='- ' print(Map(size=3)._mat) # on a dit Map.ground!</pre>	<pre>[[['-', '-', '-'], ['- ', '@', '-'], ['- ', '- ', '-']]</pre>	<pre>[[['-', '-', '-'], ['- ', '@', '-'], ['- ', '- ', '-']]</pre>	✓
✓	<pre>print(Map(size=3)._mat is Map(size=5)._mat) print(Map(size=3)._elem is Map(size=5)._elem) # _mat et _elem sont des attributs d'instance et non de classe!</pre>	<pre>False False</pre>	<pre>False False</pre>	✓

Tous les tests ont été réussis ! ✓

► **Montrer / masquer la solution de l'auteur de la question (Python3)**



Question 7

Correct

Note de 1,00 sur 1,00

Implémentez la représentation de Map. Il s'agira d'une chaîne de *len* lignes de *len* caractères. Pour séparer chaque ligne utilisez le caractère spécial '\n'.

Par exemple:

Test	Résultat
print(Map(size=3))	... .@. ...
print(Map(pos=Coord(4,1),hero='X'))	..... ....X ..... ..... ..... .....

Réponse : (régime de pénalités : 0 %)

```
1 class Coord(object):
2     def __init__(self, x = 0, y = 0):
3         self.x = x
4         self.y = y
5
6     def __repr__(self):
7         return f"<{self.x},{self.y}>"
8
9     def __eq__(self, other):
10        return self.x == other.x and self.y == other.y
11
12    def __add__(self, other):
13        return Coord(self.x+other.x, self.y+other.y)
14
15 class Map(object):
16     ground = "."
17     dir = {
18         'z' : Coord(0, -1),
19         's' : Coord(0, 1),
20         'd' : Coord(1, 0),
21         'q' : Coord(-1, 0)
22     }
23     def __init__(self, size=5, pos=Coord(1, 1), hero="@"):
24         self.size = size
25         self.pos = pos
26         self.hero = hero
27         self._mat = [[Map.ground for i in range(self.size)] for j in range(self.size)]
28         self._mat[pos.y][pos.x] = self.hero
29         self._elem = {self.hero: self.pos}
30
31     def __repr__(self):
32         c = ""
33         for ligne in self._mat:
34             for elem in ligne:
35                 c += elem
36             c += "\n"
37         return c
```

	Test	Résultat attendu	Résultat obtenu	
✓	print(Map(size=3))	... .@. ...	... .@. ...	✓
	print(Map(pos=Coord(4,1),hero='X'))	..... ....X ..... ..... ..... .....	..... ....X ..... ..... ..... .....	

Tous les tests ont été réussis ! ✓

► Montrer / masquer la solution de l'auteur de la question (Python3)

Correct

Note pour cet envoi : 1,00/1,00.



Question 8

Correct

Note de 1,00 sur 1,00

Vous pouvez surcharger (redéfinir) la méthode `__len__(self)` dans une classe pour définir la valeur renvoyée par l'appel de `len(o)` avec o une instance de cette classe.

Faites ceci dans **Map**, pour que **len** renvoie la largeur de la matrice de la carte.

Par exemple:

Test	Résultat
<code>print(len(Map()))</code>	5
<code>print(len(Map(size=27)))</code>	27

Réponse : (régime de pénalités : 0 %)

```
1 class Coord(object):
2     def __init__(self, x = 0, y = 0):
3         self.x = x
4         self.y = y
5
6     def __repr__(self):
7         return f"<{self.x},{self.y}>"
8
9     def __eq__(self, other):
10        return self.x == other.x and self.y == other.y
11
12    def __add__(self, other):
13        return Coord(self.x+other.x, self.y+other.y)
14
15 class Map(object):
16     ground = "."
17     dir = {
18         'z' : Coord(0, -1),
19         's' : Coord(0, 1),
20         'd' : Coord(1, 0),
21         'q' : Coord(-1, 0)
22     }
23     def __init__(self, size=5, pos=Coord(1, 1), hero="@"):
24         self.size = size
25         self.pos = pos
26         self.hero = hero
27         self._mat = [[Map.ground for i in range(self.size)] for j in range(self.size)]
28         self._mat[pos.y][pos.x] = self.hero
29         self._elem = {self.hero: self.pos}
30
31     def __repr__(self):
32         c = ""
33         for ligne in self._mat:
34             for elem in ligne:
35                 c += elem
36             c += "\n"
37         return c
38
39     def __len__(self):
40         return self.size
```

	Test	Résultat attendu	Résultat obtenu	
✓	<code>print(len(Map()))</code> <code>print(len(Map(size=27)))</code>	5 27	5 27	✓

Tous les tests ont été réussis ! ✓

► **Montrer / masquer la solution de l'auteur de la question (Python3)**

Correct

Note pour cet envoi : 1,00/1,00.

Question 9

Correct

Note de 1,00 sur 1,00

Vous pouvez surcharger (redéfinir) la méthode `__contains__(self, item)` dans une classe pour définir la valeur renvoyé par l'opérateur `in`. Attention, quand `x in y` est interprété, l'appel correspondant est `__contains__(y, x)`.

Faites ceci dans `Map`, pour que `o in m`, renvoie, pour une carte `m`, vrai si et seulement si:

- `o` est une coordonnée `<x,y>` valable dans la matrice (i.e. `x` et `y` entre 0 et `len(m)-1`)
- ou, `o` est un élément présent sur la carte

Vous pouvez utiliser la fonction `isinstance(o, c)` pour vérifier si `o` est une instance de la classe `c`.

Par exemple:

Test	Résultat
<code>print(Coord(0,4) in Map())</code>	True
<code>print(Coord(0,4) in Map(3))</code>	False
<code>print(Coord(0,5) in Map())</code>	False
<code>print(Coord(-1,3) in Map())</code>	False
<code>print(Coord(1,-1) in Map())</code>	False
<code>print(Coord(5,2) in Map())</code>	False
<code>print(Coord(4,4) in Map())</code>	True
<code>print('@' in Map())</code>	True
<code>print('X' in Map())</code>	False
<code>print('X' in Map(pos=Coord(4,4), hero='X'))</code>	True

Réponse : (régime de pénalités : 0 %)

```
1 class Coord(object):
2     def __init__(self, x = 0, y = 0):
3         self.x = x
4         self.y = y
5
6     def __repr__(self):
7         return f"<{self.x},{self.y}>"
8
9     def __eq__(self, other):
10        return self.x == other.x and self.y == other.y
11
12    def __add__(self, other):
13        return Coord(self.x+other.x, self.y+other.y)
14
15 class Map(object):
16     ground = "."
17     dir = {
18         'z' : Coord(0, -1),
19         's' : Coord(0, 1),
20         'd' : Coord(1, 0),
21         'q' : Coord(-1, 0)
22     }
23    def __init__(self, size=5, pos=Coord(1, 1), hero="@"):
24        self.size = size
25        self.pos = pos
26        self.hero = hero
27        self._mat = [[Map.ground for i in range(self.size)] for j in range(self.size)]
28        self._mat[pos.y][pos.x] = self.hero
29        self._elem = {self.hero: self.pos}
30
31    def __repr__(self):
32        c = ""
33        for ligne in self._mat:
34            for elem in ligne:
35                c += elem
36            c += "\n"
37        return c
38
39    def __len__(self):
40        return self.size
41
42    def __contains__(self, item):
43        if isinstance(item, Coord) and 0 <= item.x < len(self) and 0 <= item.y < len(self):
44            return True
45        if isinstance(item, str):
46            for ligne in self._mat:
47                if item in ligne:
48                    return True
49        return False
```

	Test	Résultat attendu	Résultat obtenu	
✓	<pre>print(Coord(0,4) in Map()) print(Coord(0,4) in Map(3)) print(Coord(0,5) in Map()) print(Coord(-1,3) in Map()) print(Coord(1,-1) in Map()) print(Coord(5,2) in Map()) print(Coord(4,4) in Map())</pre>	<pre>True False False False False False False True</pre>	<pre>True False False False False False False True</pre>	✓
✓	<pre>print('@' in Map()) print('X' in Map()) print('X' in Map(pos=Coord(4,4), hero='X'))</pre>	<pre>True False True</pre>	<pre>True False True</pre>	✓
✓	<pre>print(type("Coord", (), {}]() in Map()) # Please use isinstance!</pre>	<pre>False</pre>	<pre>False</pre>	✓

Tous les tests ont été réussis ! ✓

► **Montrer / masquer la solution de l'auteur de la question (Python3)**

Correct

Note pour cet envoi : 1,00/1,00.

**Attention:** l'état de la matrice et le dictionnaire d'objets doivent toujours être cohérents

Ajoutez à **Map** les méthodes suivantes :

- **get(c)** : retourne l'élément aux coordonnées **c**
- **pos(e)** : retourne les coordonnées de l'élément **e**
- **put(c, e)** : place l'élément **e** aux coordonnées **c**
- **rm(c)** : supprime l'élément aux coordonnées **c**

*Note:* Pour supprimer un élément dans un dictionnaire vous pouvez utiliser l'instruction del :

```
del dict[x] #supprime l'entrée x de dict
```

Plus de détails sur l'utilisation des dictionnaires :

- **Apprendre à programmer avec Python 3**, pages 160-167 (pages 182-189 du pdf), partie **"Dictionnaires"**.
- [https://www.w3schools.com/python/python\\_ref\\_dictionary.asp](https://www.w3schools.com/python/python_ref_dictionary.asp)

Par exemple:

Test	Résultat
print(Map().get(Coord(0,4))) print(Map().get(Coord(1,1))) print(Map().get(Coord(2,3)))	. @ .
print(Map().pos('@')) print(Map(pos=Coord(2, 3), hero='X').pos('X'))	<1,1> <2,3>
m = Map() m.put(Coord(3,2), 'X') m.put(Coord(0,0), 'A') print(m)	A.... .@... ...X. ..... .....
print(m._elem)	{ '@': <1,1>, 'X': <3,2>, 'A': <0,0> }
m = Map() m.rm(Coord(1,1)) print(m)	..... ..... ..... ..... .....
print(m._elem)	{ }

Réponse : (régime de pénalités : 0 %)


```
1 class Coord(object):
2     def __init__(self, x = 0, y = 0):
3         self.x = x
4         self.y = y
5
6     def __repr__(self):
7         return f"<{self.x},{self.y}>"
8
9     def __eq__(self, other):
10        if type(self) == type(other):
11            return self.x == other.x and self.y == other.y
12        raise NotImplementedError
13
14    def __add__(self, other):
15        return Coord(self.x+other.x, self.y+other.y)
16
17 class Map(object):
18     ground = "."
19     dir = {
20         'z' : Coord(0, -1),
21         's' : Coord(0, 1),
22         'd' : Coord(1, 0),
23         'q' : Coord(-1, 0)
24     }
25
26    def __init__(self, size=5, pos=Coord(1, 1), hero="@"):
27        self.size = size
28        self.hero = hero
29        self._mat = [[Map.ground for i in range(self.size)] for j in range(self.size)]
30        self._mat[pos.y][pos.x] = hero
```

```

31         self._elem = {hero: pos}
32
33     def __repr__(self):
34         c = ""
35         for ligne in self._mat:
36             for elem in ligne:
37                 c += elem
38             c += "\n"
39         return c
40
41     def __len__(self):
42         return self.size
43
44     def __contains__(self, item):
45         if isinstance(item, Coord) and 0 <= item.x < len(self) and 0 <= item.y < len(self):
46             return True
47         if isinstance(item, str):
48             for ligne in self._mat:
49                 if item in ligne:
50                     return True
51         return False
52

```

	Test	Résultat attendu	Résultat obtenu	
✓	print(Map().get(Coord(0,4))) print(Map().get(Coord(1,1))) print(Map().get(Coord(2,3)))	. @ .	. @ .	✓
✓	print(Map().pos('@')) print(Map(pos=Coord(2, 3), hero='X').pos('X'))	<1,1> <2,3>	<1,1> <2,3>	✓
✓	m = Map() m.put(Coord(3,2),'X') m.put(Coord(0,0),'A') print(m)  print(m._elem)	A... .@... ...X. ..... .....  { '@': <1,1>, 'X': <3,2>, 'A': <0,0> }	A... .@... ...X. ..... .....  { '@': <1,1>, 'X': <3,2>, 'A': <0,0> }	✓
✓	m = Map() m.rm(Coord(1,1)) print(m)  print(m._elem)	..... ..... ..... ..... .....  { }	..... ..... ..... ..... .....  { }	✓
✓	m = Map() Map.ground = '-' m.rm(Coord(1,1)) print(m) # On a dit ground! Map.ground = '.'	..... .-... ..... ..... .....	..... .-... ..... ..... .....	✓
✓	print(Map(pos = Coord(2,3)).get(Coord(2,3))) print(Map(pos = Coord(2,3)).get(Coord(3,2)))	@ .	@ .	✓
✓	print(Map(pos=Coord(3,2)).get(Coord(3,2))) print(Map(pos=Coord(3,2)).pos('@'))	@ <3,2>	@ <3,2>	✓
✓	m = Map(pos=Coord(3,2)) m.rm(Coord(3,2)) print(m)	..... ..... ..... ..... .....	..... ..... ..... ..... .....	✓
✓	m = Map() m._elem['G']=Coord(0,0) m._mat[0][0]='G' print(m) print(m._elem) print(m.get(Coord(0,0))) print(m.pos('G')) m.rm(Coord(0,0)) print(m) print(m._elem)	G... .@... ..... ..... .....  { '@': <1,1>, 'G': <0,0> } G <0,0> ..... .@... ..... ..... .....  { '@': <1,1> }	G... .@... ..... ..... .....  { '@': <1,1>, 'G': <0,0> } G <0,0> ..... .@... ..... ..... .....  { '@': <1,1> }	✓

Tous les tests ont été réussis ! 

► **Montrer / masquer la solution de l'auteur de la question (Python3)**

Correct

Note pour cet envoi : 4,00/4,00.

Question 11

Correct

Note de 2,00 sur 2,00

**Attention:** A partir de maintenant, vous ne devriez plus avoir besoin d'accéder directement aux attributs `_mat` et `_elem`, mais uniquement aux méthodes `get/pos/put/rm` ou aux méthodes spéciales (`len/str/in/...`), ceci afin d'assurer leur cohérence.

*Note:* Le " " du nom des attributs `"_mat"` et `"_elem"` est justement *une convention de nommage* qui permet d'indiquer que ces attributs sont "protégés" et ne devraient pas être manipulés, ni même lus directement (sauf pour des tests).

Ajoutez une méthode **move(e, way)** qui déplace l'élément **e** dans la direction **way**. Pour que le déplacement soit possible, il faut que les nouvelles coordonnées soient valables et que l'emplacement d'arrivée soit vide.  
Si le déplacement est impossible, la méthode ne modifie pas la carte.

Par exemple:

Test	Résultat
<pre>m = Map(3) m.move('@', Coord(-1,0)) print(m)  print(m._elem) m.move('@', Coord(0,1)) print(m)  print(m._elem)</pre>	<pre>... @.. ...  {'@': &lt;0,1&gt;} ... ... @..  {'@': &lt;0,2&gt;}</pre>
<pre>m = Map(3, pos=Coord(0,0)) m.move('@', Coord(-1,0)) print(m)  print(m._elem) m.move('@', Coord(0,2)) print(m)  print(m._elem) m.move('@', Coord(0,1)) print(m)  print(m._elem)</pre>	<pre>@.. ... ...  {'@': &lt;0,0&gt;} ... ... @..  {'@': &lt;0,2&gt;} ... ... @..  {'@': &lt;0,2&gt;}</pre>
<pre>m = Map(3) m.put(Coord(2,1), 'X') m.move('X', Coord(-1,0)) print(m) print(m._elem)  m.move('X', Coord(0,1)) print(m)  print(m._elem)</pre>	<pre>... .@X ...  {'@': &lt;1,1&gt;, 'X': &lt;2,1&gt;} ...  .@. ..X  {'@': &lt;1,1&gt;, 'X': &lt;2,2&gt;}</pre>

Réponse : (régime de pénalités : 0 %)

```
1 class Coord(object):
2     def __init__(self, x = 0, y = 0):
3         self.x = x
4         self.y = y
5
6     def __repr__(self):
7         return f"<{self.x},{self.y}>"
8
9     def __eq__(self, other):
10        if type(self) == type(other):
11            return self.x == other.x and self.y == other.y
12        raise NotImplementedError
13
14    def __add__(self, other):
15        return Coord(self.x+other.x, self.y+other.y)
16
17 class Map(object):
18     ground = "."
19     dir = {
20         'z' : Coord(0, -1),
21         's' : Coord(0, 1),
22         'd' : Coord(1, 0),
```

```

23     'q' : Coord(-1, 0)
24 }
25
26 def __init__(self, size=5, pos=Coord(1, 1), hero="@"):
27     self.size = size
28     self.hero = hero
29     self._mat = [[Map.ground for i in range(self.size)] for j in range(self.size)]
30     self._mat[pos.y][pos.x] = hero
31     self._elem = {hero: pos}
32
33 def __repr__(self):
34     c = ""
35     for ligne in self._mat:
36         for elem in ligne:
37             c += elem
38         c += "\n"
39     return c
40
41 def __len__(self):
42     return self.size
43
44 def __contains__(self, item):
45     if isinstance(item, Coord) and 0 <= item.x < len(self) and 0 <= item.y < len(self):
46         return True
47     if isinstance(item, str):
48         for ligne in self._mat:
49             if item in ligne:
50                 return True
51     return False
52

```

	Test	Résultat attendu	Résultat obtenu	
✓	<pre> m = Map(3) m.move('@', Coord(-1,0)) print(m)  print(m._elem) m.move('@', Coord(0,1)) print(m)  print(m._elem) </pre>	<pre> ... @.. ...  {'@': &lt;0,1&gt;} ... ... @..  {'@': &lt;0,2&gt;} </pre>	<pre> ... @.. ...  {'@': &lt;0,1&gt;} ... ... @..  {'@': &lt;0,2&gt;} </pre>	✓
✓	<pre> m = Map(3, pos=Coord(0,0)) m.move('@', Coord(-1,0)) print(m)  print(m._elem) m.move('@', Coord(0,2)) print(m)  print(m._elem) m.move('@', Coord(0,1)) print(m)  print(m._elem) </pre>	<pre> @.. ... ...  {'@': &lt;0,0&gt;} ... ... @..  {'@': &lt;0,2&gt;} ... ... @..  {'@': &lt;0,2&gt;} </pre>	<pre> @.. ... ...  {'@': &lt;0,0&gt;} ... ... @..  {'@': &lt;0,2&gt;} ... ... @..  {'@': &lt;0,2&gt;} </pre>	✓
✓	<pre> m = Map(3) m.put(Coord(2,1),'X') m.move('X', Coord(-1,0)) print(m) print(m._elem)  m.move('X', Coord(0,1)) print(m)  print(m._elem) </pre>	<pre> ... .@X ...  {'@': &lt;1,1&gt;, 'X': &lt;2,1&gt;} ... .@. ..X  {'@': &lt;1,1&gt;, 'X': &lt;2,2&gt;} </pre>	<pre> ... .@X ...  {'@': &lt;1,1&gt;, 'X': &lt;2,1&gt;} ... .@. ..X  {'@': &lt;1,1&gt;, 'X': &lt;2,2&gt;} </pre>	✓
✓	<pre> Map.ground = '-' m = Map(3) m.move('@', Coord(-1,0)) print(m) </pre>	<pre> --- @-- ---</pre>	<pre> --- @-- ---</pre>	✓

Tous les tests ont été réussis ! ✓

► [Montrer / masquer la solution de l'auteur de la question \(Python3\)](#)

Correct

Note pour cet envoi : 2,00/2,00.



Ajoutez le code suivant (qui permet de lire un caractère au clavier):

```
def getch():
    """Single char input, only works only on mac/linux/windows OS terminals"""
    try:
        import termios
        # POSIX system. Create and return a getch that manipulates the tty.
        import sys, tty
        fd = sys.stdin.fileno()
        old_settings = termios.tcgetattr(fd)
        try:
            tty.setraw(fd)
            ch = sys.stdin.read(1)
        finally:
            termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
        return ch
    except ImportError:
        # Non-POSIX. Return msvcrt's (Windows') getch.
        import msvcrt
        return msvcrt.getch().decode('utf-8')
```

Ainsi que la méthode suivante dans **Map**:

```
def play(self, hero='@'):
    while True:
        print(self)
        self.move(hero, Map.dir[getch()])
```

Vous pouvez maintenant essayer **dans votre terminal** votre embryon de jeu, en appelant la méthode **play**.

Pour exécuter votre programme depuis votre terminal vous devez :

- Vous placer dans le répertoire qui contient votre fichier (*cd ...*)
- Exécuter *python3.6 rogue.py* (le nom de votre fichier, qui doit contenir l'instruction *Map().play()* à la fin)

Au cours de cette leçon, vous avez appris :

1. Que les attributs sont définis pour une classe ou pour une instance
2. Que le comportement des opérateurs et fonctions prédéfinies du langage Python peuvent être redéfinis pour une classe donnée, grâce aux méthodes spéciales `__`
3. Que des convention de nommage visent à "protéger" des attributs

**Question bonus :** en utilisant *get/pos/put*, et de la même manière que vous avez redéfini l'opérateur *in*, vous pouvez définir les méthodes `__getitem__(self, key)` et `__setitem__(self, key, value)` afin de définir le comportement des `[]`. Ainsi, vous pourrez écrire :

```
print(map['@'])          # les coordonnées de '@' (pos)
print(map[Coord(1,1)])  # ce qui se trouve aux coordonnées <1,1> (get)
map[Coord(1,0)] = '@'   # ajout de '@' en <1,0> (put)
map['@'] = Coord(2,0)    # un put ou un déplacement si '@' déjà présent
```

Vous pouvez aussi définir la méthode `__delitem__(self, item)` pour redéfinir le comportement de l'instruction *del*.