



# Bases de la programmation impérative (BPI)

CM3 - Python est un (faux ?) langage objet : classes, instances, références et mémoire

Manuel Selva



## Sommaire du jour

pylint

Références

Problème

Explications

Nos propres classes

Création

Utilisation

À retenir



## Sommaire du jour

pylint

Références

Problème

Explications

Nos propres classes

Création

Utilisation

À retenir



## (encore) Un sondage

Qui a déjà entendu parler de `pylint` ?



## (encore) Un sondage

Qui a déjà entendu parler de `pylint` ?

Qui a `pylint` de configuré dans son éditeur utilisé pour écrire le code des TP BPI ?



## (encore) Un sondage

Qui a déjà entendu parler de `pylint` ?

Qui a `pylint` de configuré dans son éditeur utilisé pour écrire le code des TP BPI ?

Qui lit et corrige les messages de `pylint` ?

```
1 >"""Illustration de pylint"""$
2 $
3 >def affiche_moyenne(note_projet, noteExam):$
4 >    ··· moyenne=(1*note_projet + 3*noteExam) / 4
5 >    ··· print((1*note_projet + 3*noteExam) / 4)$
6 $
7 if __name__ == "__main__":$
8 >    ··· affiche_moyenne(12)$
```



## pylint : pour résumé

- Détecter des erreurs au plus vite, c'est à dire avant d'exécuter le programme
  
- Améliorer la qualité du code





## pylint : pour résumé

- Détecter des erreurs au plus vite, c'est à dire avant d'exécuter le programme
- Améliorer la qualité du code
- (avoir une meilleure note BPI)



## Sommaire du jour

pylint

Références

Problème

Explications

Nos propres classes

Création

Utilisation

À retenir



## Une question facile pour commencer

Que fait ce programme ?

```
l = [6] * 7  
print("the sum of elements in",  
l, "is", sum(l))
```

## Que fait le programme ci-dessous ?

```
1  from termcolor import colored
2
3  def print_stuff(s):
4      for i in range(8):
5          for j in range(8):
6              w = s[i][j]
7              c = 'white' if w else 'blue'
8              # \u2588 est un caractère
9              # UNICODE représentant un
10             # rectangle vertical plein.
11             # on obtient donc un carré
12             # en en affichant deux.
13             t = colored('\u2588\u2588', c)
14             print(t, end='')
15         print()
16
17
18  def init_stuff():
19      res = [[]] * 8
20      start_w = True
21      for i in range(8):
22          white = start_w
23          for j in range(8):
24              res[i].append(white)
25              white = not white
26          start_w = not start_w
27      return res
28
29  def main():
30      stuff = init_stuff()
31      print_stuff(stuff)
32
33  if __name__ == "__main__":
34      main()
```

## Que fait le programme ci-dessous ?

### Les propositions

- A. affiche un échiquier blanc et bleu sur la sortie standard
- B. affiche des trucs sur la sortie standard
- C. affiche un échiquier multicolore sur la sortie standard
- D. affiche des lignes blanches et bleues sur la sortie standard
- E. affiche des colonnes blanches et bleues sur la sortie standard
- F. génère une erreur
- G. il manque des données pour répondre à la question
- H. je ne sais pas



## La bonne réponse

### Les propositions

- A. affiche un échiquier blanc et bleu sur la sortie standard
- B. affiche des trucs sur la sortie standard
- C. affiche un échiquier multicolore sur la sortie standard
- D. affiche des lignes blanches et bleues sur la sortie standard
- E. affiche des colonnes blanches et bleues sur la sortie standard
- F. génère une erreur
- G. il manque des données pour répondre à la question
- H. je ne sais pas



## Explications : une seule liste vide est créée

```
In [3]: liste_de_listes = [[]] * 4
```

```
In [4]: print(liste_de_listes)
[[], [], [], []]
```

```
In [5]: liste_de_listes[1].append(42)
```

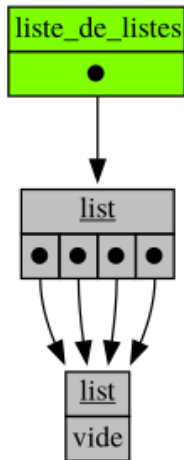
```
In [6]: print(liste_de_listes)
[[42], [42], [42], [42]]
```

# Explications : la preuve en images !

```
In [4]: traceur_v = traceur.Variable(
...: "liste_de_listes", liste_de_listes)

In [5]: traceur.display_graph_from_vars(
...: True, False, traceur_v)
/tmp/tmpdoxg806z.dot.svg

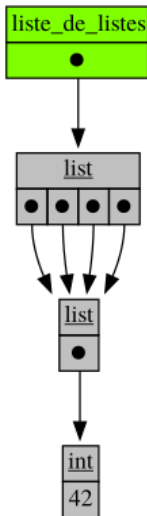
In [6]: !eog /tmp/tmpdoxg806z.dot.svg
```





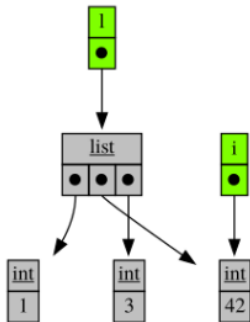
## Explications : et donc quand on append ...

```
In [7]: liste_de_listes[1].append(42)
In [8]: traceur.display_graph_from_vars(
...: True, False, traceur_v)
/tmp/tmpd_4n8khc.dot.svg
In [9]: !eog /tmp/tmpd_4n8khc.dot.svg
```



## Des références partout en Python !

```
In [2]: i = 42
In [3]: l = [1, i, 3]
In [4]: traceur_v_i = traceur.Variable("i", i)
In [5]: traceur_v_l = traceur.Variable("l", l)
In [6]: traceur.display_graph_from_vars(
...: True, False, traceur_v_i, traceur_v_l)
/tmp/tmpw5ynx8ji.dot.svg
In [7]: !eog /tmp/tmpw5ynx8ji.dot.svg
```



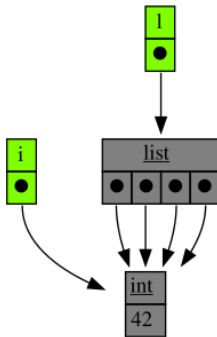
Quelle est la différence avec le code ci-dessous ?

```
In [2]: i = 42  
  
In [3]: l = [i] * 4  
  
In [4]: print(l)  
[42, 42, 42, 42]  
  
In [5]: i = 17  
  
In [6]: print(l)  
[42, 42, 42, 42]
```

## La réponse en images

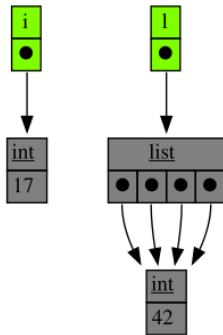
```

In [17]: i = 42
In [18]: l = [i] * 4
In [19]: traceur_v_i = traceur.Variable("i", i)
In [20]: traceur_v_l = traceur.Variable("l", l)
In [21]: traceur.display_graph_from_vars(
...: True, False, traceur_v_i, traceur_v_l)
/tmp/tmp44gc5pja.dot.svg
In [22]: !eog /tmp/tmp44gc5pja.dot.svg
  
```



# La réponse en images

```
In [22]: i = 17
In [23]: traceur.display_graph_from_vars(
...: True, False, traceur_v_i, traceur_v_l)
/tmp/tmpjtdg24mg.dot.svg
In [24]: !eog /tmp/tmpjtdg24mg.dot.svg
```





## Quatre définitions fondamentales

- classe : ensemble d'attributs (et de méthodes)



## Quatre définitions fondamentales

- **classe** : ensemble d'attributs (et de méthodes)
- **instance** : zone mémoire contenant un ensemble d'attributs tels que définis par la classe à laquelle l'instance est attachée



## Quatre définitions fondamentales

- **classe** : ensemble d'attributs (et de méthodes)
- **instance** : zone mémoire contenant un ensemble d'attributs tels que définis par la classe à laquelle l'instance est attachée
- **référence** : zone mémoire contenant un lien vers une instance





## Quatre définitions fondamentales

- **classe** : ensemble d'attributs (et de méthodes)
- **instance** : zone mémoire contenant un ensemble d'attributs tels que définis par la classe à laquelle l'instance est attachée
- **référence** : zone mémoire contenant un lien vers une instance
- **variable** : nom symbolique désignant une zone mémoire (en Python, toutes les variables SONT DES RÉFÉRENCES)



## Sommaire du jour

pylint

Références

Problème

Explications

Nos propres classes

Création

Utilisation

À retenir



## Qu'est-ce qu'une classe ?

"Classes provide a means of bundling data and functionality together.

Creating a new class creates a new type of object, allowing new instances of that type to be made.

Each class instance can have attributes attached to it for maintaining its state.

(Class instances can also have methods defined by its class for modifying its state.)"

<https://docs.python.org/3.6/tutorial/classes.html>



## Création d'une classe TortueLogo

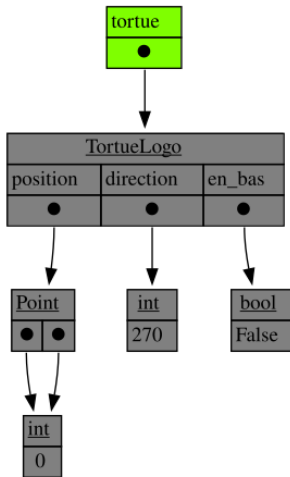
```
1 class TortueLogo:
2     """Une tortue logo telle que vue en TP."""
3
4     def __init__(self, position, direction, en_bas):
5         self.position = position
6         self.direction = direction
7         self.en_bas = en_bas
```



## Utilisation de la classe TortueLogo

```
1
2 def test_tortue():
3     """Instancie une tortue et joue un peu avec."""
4
5     # Instanciation et affichage de la tortue
6     tortue = TortueLogo(svg.Point(0, 0), 270, False)
7     print(tortue)
8
9     # Modification des attributs de la tortue
10    # qui est donc MUTABLE puis nouvel affichage
11    tortue.direction -= 90
12    tortue.en_bas = True
13
14    # Affichage à nouveau
15    print(tortue)
16
17 if __name__ == "__main__":
18    test_tortue()
```

## Une TortueLogo en mémoire ça donne quoi ?





## print d'une TortueLogo

```
In [14]: print(tortue)
<tortue_logo.TortueLogo object at 0x7fb4f9698e50>
```



## print d'une TortueLogo

```
In [14]: print(tortue)
<tortue_logo.TortueLogo object at 0x7fb4f9698e50>
```

- documentation de `print` dit :  
"All arguments are converted to strings like `str()` does"





## print d'une TortueLogo

```
In [14]: print(tortue)
<tortue_logo.TortueLogo object at 0x7fb4f9698e50>
```

- documentation de `print` dit :  
"All arguments are converted to strings like `str()` does"
- documentation de `str` dit :  
"`str(object)` returns `object.__str__()`"

## print d'une TortueLogo

```
def __str__(self):
    str_repr = (f'TortueLogo :\n'
               f' position = {self.position}\n'
               f' direction = {self.direction}\n'
               f' en_bas = {self.en_bas}')
```

```
return str_repr
```

```
In [3]: import svg

In [4]: from tortue_logo import TortueLogo

In [5]: tortue = TortueLogo(svg.Point(0, 0), 270, False)

In [6]: print(tortue)
TortueLogo :
position = Point(x=0, y=0)
direction = 270
en_bas = False
```



## Sommaire du jour

pylint

Références

Problème

Explications

Nos propres classes

Création

Utilisation

À retenir



## À retenir

- j'installe `pylint` **d'ici ce soir !**
- une instance est une **zone mémoire** avec des attributs
- en Python, variables et attributs sont des **références vers des instances**
- les instances de nos propres classes sont **mutables**
- je définis `__str__` dans mes propres classes pour que `print` sache comment afficher les instances de mes classes