

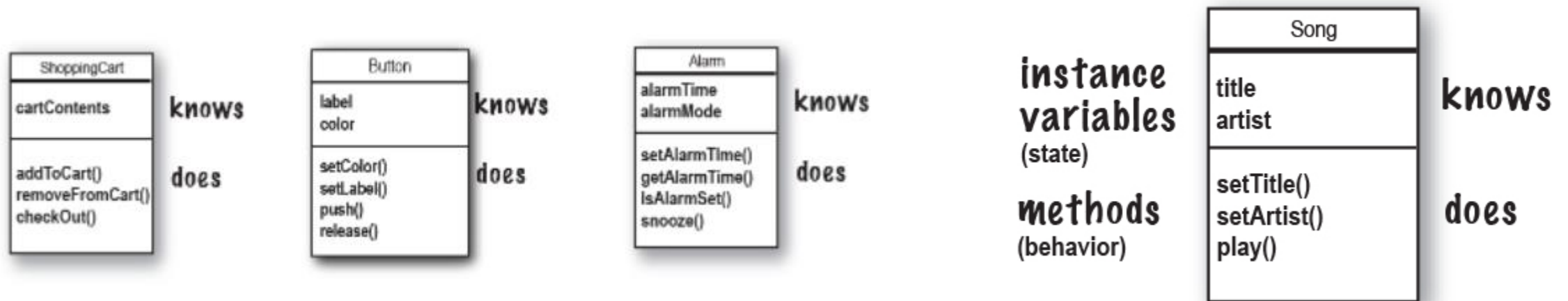
Classe en Python

Ali ASSI

PhD Informatique

Comment modéliser une classe

- Lorsque vous construisez une classe, pensez aux objets qui seront créés à partir de ce type de classe. Pensez-y :
 - ce que l'objet sait (data): **variables d'instance**
 - ce que l'objet peut faire: **méthodes**



instance est une autre façon de dire **objet**


Un diagramme de classe pour la classe Product

- **UML (Unified Modeling Language)** est la norme industrielle utilisée pour modéliser les *classes* et les *objets* d'une application orientée objet.
- Le signe moins (-) dans un diagramme de classe UML indique les champs et les méthodes auxquels les autres classes ne peuvent pas accéder (*private*);
- Le signe plus (+) indique les champs et les méthodes auxquels les autres classes peuvent accéder (*public*).

Product	
-code: String -description: String -price: double	Fields
+setCode(String) +getCode(): String +setDescription(String) +getDescription(): String +setPrice(double) +getPrice(): double +getPriceFormatted(): String	Methods

Exemple

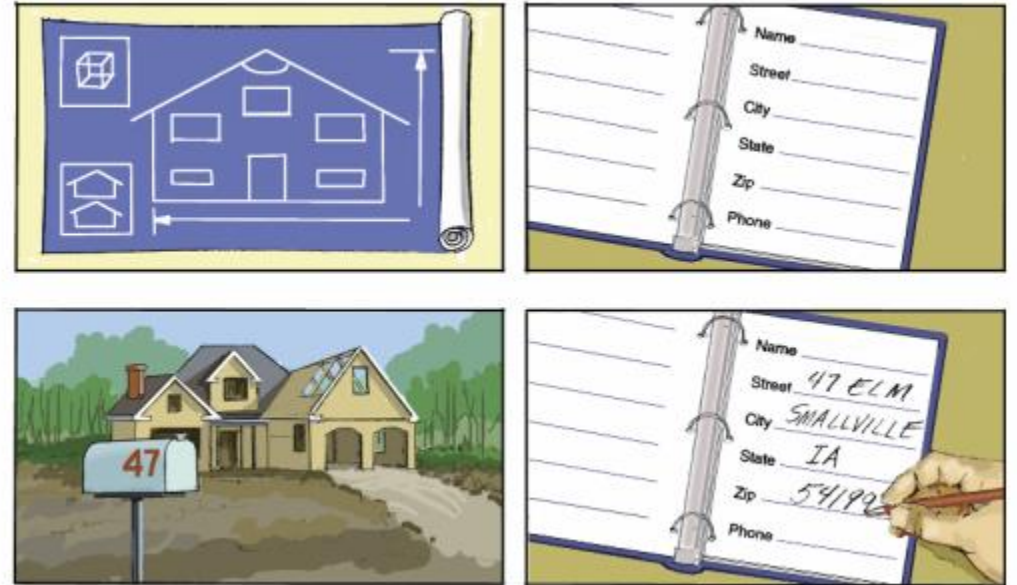
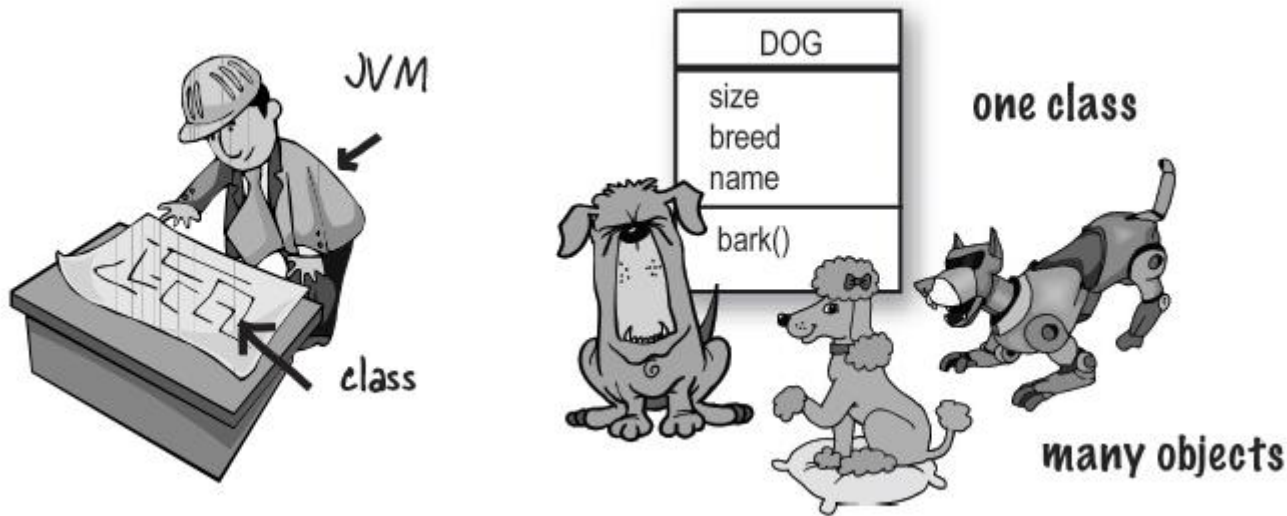


Voiture	
ID	
Prix	
Couleur	
Description	
Nombre jours location	
Augmenter nombre jours location	
Diminuer nombre jours location	
Calculer Prix Total Location	

Client
- numero : long
- nom : String
Client(numero : long, nom : String)
+ getNumero() : long
+ getNom() : String
+ setNom(nom : String) : void
+ toString()
+ main(args : String [])

Quelle est la différence entre une classe et un objet ?

- Une classe n'est pas un objet (mais elle est utilisée pour le construire).



- Chaque objet créé à partir d'une classe peut avoir ses **propres** valeurs pour les variables d'instance de cette classe.

Déclaration d'une classe, des variables d'instance et des méthodes

```
employee.py ×  main.py
employee.py > Employee > __init__
1  class Employee:
2
3      ## class attribute
4      alias = 'Data Management'
5
6      def __init__(self, name, position, age, salary, experience):
7          ## Instance attributes
8              self.name = name
9              self.position = position
10             self.age = age
11             self.salary = salary
12             self.experience = experience
13
14     def introduce(self):
15         print("My name is " + self.name + " and I am " + str(self.age) + " years old.")
```

Déclaration d'une classe et des variables d'instance - Suite

```
employee.py  main.py  ×  
main.py > ...  
1  from employee import Employee  
2  
3  # instance or object  
4  emp = Employee("Developer", 'Pierre', 29,50000,5)  
5  ## accessing class members  
6  print(emp.position)    ## accessing self.position  
7  print(emp.age)        ## accessing self.age  
8  print(Employee.age)   ## Error (Instance attributes only be access by a instance of the class)  
9  print(emp.alias)      ## Data Management  
10 print(Employee.alias) ## Data Management (a class attribute can be access by instacen and classs :  
11 emp.introduce()       ## My name is Developer and I am 29 years old.
```

Déclaration d'une classe et des variables d'instance - Suite

- `__init__` est une méthode réservée aux classes en Python.
- Elle s'agit d'un constructeur qui est automatiquement appelé lors de la création de l'objet de la classe.
- Elle est principalement utilisée pour définir les variables de la classe. Les variables à l'intérieur du constructeur sont des attributs d'instance.
- Les attributs d'instance ne sont accessibles que par l'objet de la classe.
- Si vous souhaitez créer un objet et lui transmettre ultérieurement des valeurs, vous devez utiliser ***self.variable_name*** pour chaque variable définie dans le constructeur de la classe.

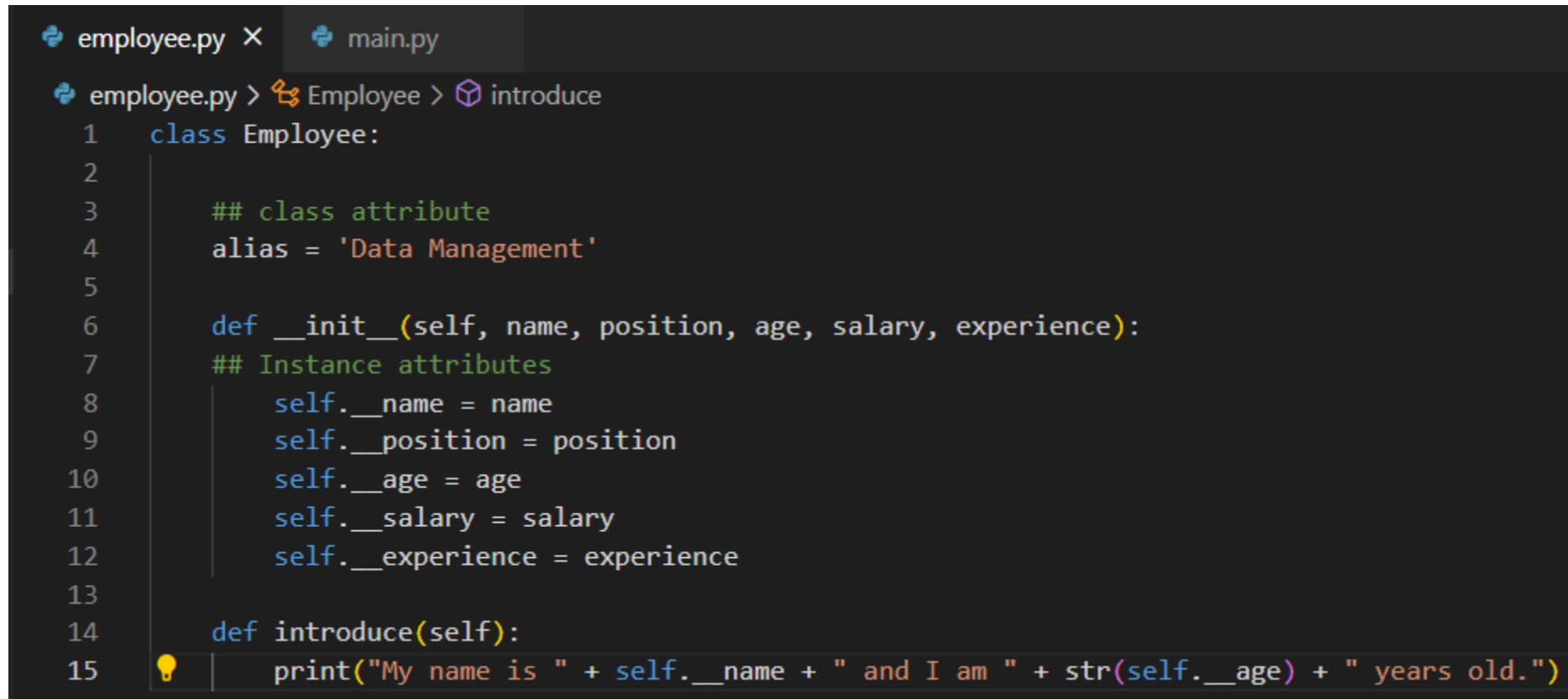
Les quatre piliers de la POO

- Encapsulation
- Abstraction
- Héritage
- Polymorphisme

Encapsulation

- En utilisant **l'encapsulation**, nous pouvons cacher les variables d'instance de la classe aux autres classes.
- L'un des objectifs de l'encapsulation d'une classe est de protéger son contenu afin qu'il ne soit pas endommagé par les actions du code externe.
- En Python, pour mettre en œuvre l'encapsulation il faut rendre la variable privée.
- Si vous essayez d'y accéder en dehors de la classe normalement, en créant un objet, vous obtiendrez une **erreur**.
- Pour accéder au membre privé, vous devez utiliser ***object.__privateMember***.

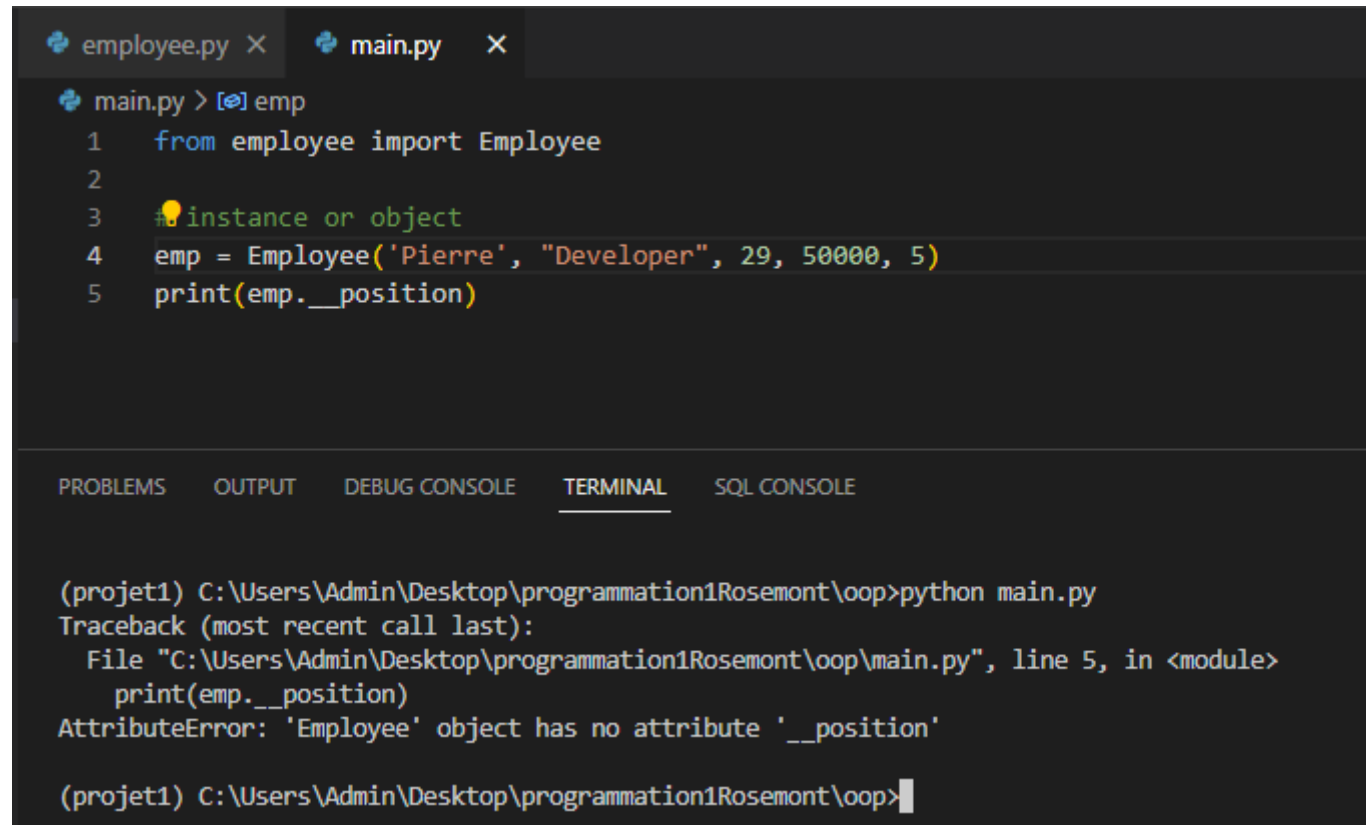
Exemple encapsulation



```
employee.py ×  main.py
employee.py > Employee > introduce
1  class Employee:
2
3      ## class attribute
4      alias = 'Data Management'
5
6      def __init__(self, name, position, age, salary, experience):
7          ## Instance attributes
8          self.__name = name
9          self.__position = position
10         self.__age = age
11         self.__salary = salary
12         self.__experience = experience
13
14     def introduce(self):
15         print("My name is " + self.__name + " and I am " + str(self.__age) + " years old.")
```

`__position` est un membre privé de la classe

Exemple encapsulation -Suite



```
employee.py × main.py ×  
main.py > [?] emp  
1 from employee import Employee  
2  
3 # instance or object  
4 emp = Employee('Pierre', "Developer", 29, 50000, 5)  
5 print(emp.__position)  
  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE  
  
(projet1) C:\Users\Admin\Desktop\programmation1Rosemont\oop>python main.py  
Traceback (most recent call last):  
  File "C:\Users\Admin\Desktop\programmation1Rosemont\oop\main.py", line 5, in <module>  
    print(emp.__position)  
AttributeError: 'Employee' object has no attribute '__position'  
  
(projet1) C:\Users\Admin\Desktop\programmation1Rosemont\oop>
```

décorateurs @property et @method_name.setter

- **@property** est utilisé pour obtenir la valeur d'un attribut privé. Nous devons placer @property avant la méthode où nous renvoyons la variable privée.
- **@method_name.setter** est utilisé pour modifier la valeur de la variable privée. Nous utilisons @method_name.setter devant la méthode.

```
employee.py X main.py
employee.py > Employee
1 class Employee:
2     ## class attribute
3     alias = 'Data Management'
4
5     def __init__(self, name, position, age, salary, experience):
6     ## Instance attributes
7         self.__name = name
8         self.__position = position
9         self.__age = age
10        self.__salary = salary
11        self.__experience = experience
12
13        @property
14        def name(self):
15            return self.__name
16
17        ## the attribute name and the method name must
18        # be same which is used to set the value for the attribute
19        @name.setter
20        def name(self, var):
21            self.__name = var
22
23        def introduce(self):
24            print("My name is " + self.__name + " and I am " + str(self.__age) + " years old.")
25
```

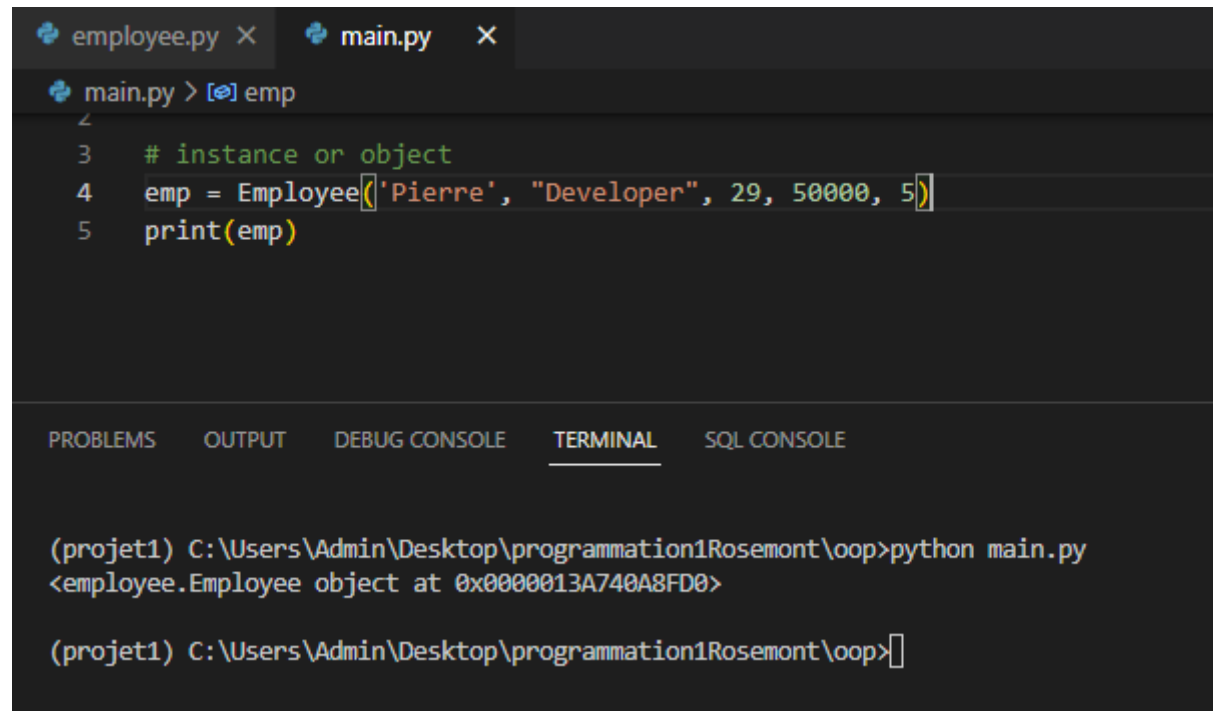
```
employee.py main.py X
main.py > ...
1 from employee import Employee
2
3 # instance or object
4 emp = Employee('Pierre', "Developer", 29, 50000, 5)
5 print(emp.name)
6
7 emp.name = 'Alain'
8 print(emp.name)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE

(projet1) C:\Users\Admin\Desktop\programmation1Rosemont\oop>python main.py
Pierre
Alain
```

__str__()

- Si on imprime la variable **emp**, qu'est qu'on obtient?



The screenshot shows a Python IDE with two tabs: `employee.py` and `main.py`. The `main.py` tab is active, showing the following code:

```
main.py > [?] emp
<
3  # instance or object
4  emp = Employee(['Pierre', "Developer", 29, 50000, 5])
5  print(emp)
```

Below the code editor, the `TERMINAL` tab is selected, displaying the output of running the script:

```
(projet1) C:\Users\Admin\Desktop\programmation1Rosemont\oop>python main.py
<employee.Employee object at 0x0000013A740A8FD0>

(projet1) C:\Users\Admin\Desktop\programmation1Rosemont\oop>
```

Exemple: `__str__()`

`__str__` : Cette méthode est appelée lorsqu'un objet est transmis à la fonction intégrée `print()` ou lorsque la fonction `str()` est appelée sur un objet. Elle est utilisée pour renvoyer une représentation sous forme de chaîne de caractères de l'objet.

```
employee.py × main.py
employee.py > Employee
1 class Employee:
2     ## class attribute
3     alias = 'Data Management'
4
5     def __init__(self, name, position, age, salary, experience):
6         ## Instance attributes
7         self.__name = name
8         self.__position = position
9         self.__age = age
10        self.__salary = salary
11        self.__experience = experience
12
13        @property
14        def name(self):
15            return self.__name
16
17        ## the attribute name and the method name must
18        # be same which is used to set the value for the attribute
19        @name.setter
20        def name(self, var):
21            self.__name = var
22
23        def __str__(self):
24            return f"Im am = {self.name} and so on."
25
26        def introduce(self):
27            print("My name is " + self.name + " and I am " + str(self.__age) + " years old.")
28
```

```
employee.py main.py ×
main.py > ...
1
2
3 # instance or object
4 emp = Employee('Pierre', "Developer", 29, 50000, 5)
5 print(emp)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE

<employee.Employee object at 0x0000013A740A8FD0>

(projet1) C:\Users\Admin\Desktop\programmation1Rosemont\oop>python main.py
Im am = Pierre and so on.
```