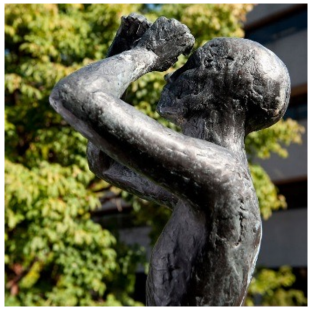


Lecture 2 – Writing a Class

CS2513

Cathal Hoare

A TRADITION OF INDEPENDENT THINKING

UCC

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Lecture Contents

Writing our First Class

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Announcements

- Labs to start this week:
  - G20 and G26 are reserved for labs on Tuesday 2 to 3.
  - First lab will concern some simple Python exercises for revision (conditionals and iteration) and writing a simple class.
  - Task will be made available on Monday. Work will be Submitted at 5pm on Friday.

Announcements

- Good Python Books
  - NOTE: There is no textbook for this module. All examinable material will be based on slides, labs and assignments.
  - Learning Python – Lutz – O’Reilly
  - Effective Python – Slatkin – Addison-Wesley
  - Other sources - python.org
    - Manual, Peps, etc
- A note on Discourse Communities

State

A Light has State: on or off

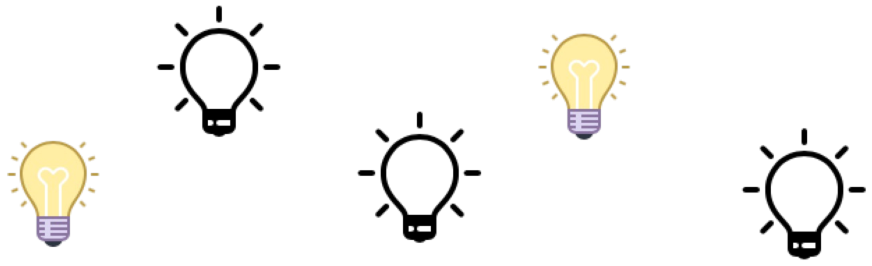
State and Actions/Behaviours

A Light has State: on or off

And we can change its state - by switching it on or off with a switch

We group state and actions/behaviours in a class. This is called encapsulation.

Classes and Objects



- There are many lights - we represent the light with a class. The attributes that describe the light and the actions that can be carried out on the light are encoded in the class.
- One class can describe each light. Each light object is an instance of the class.
- We instantiate an object using a class

Using Encapsulation: Safer Code - Fewer Errors

Ideally, we change the light’s state with a method

The developer who develops the class will understand what is needed to change the state of the light and can encode this in the method

Other developers will not know the requirements of the light as well as they are developing their own code. The light class is an opaque box to them.

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

OO Building Block - The Class

- The basic building block in OO design is the class.
- A class is a set of related state and behaviours that describe some entity.
- We capture state with variables and behaviours through methods (a.k.a. functions).

OO Building Block - The Class

- Classes act as a factory for instantiating one or more objects.
- Classes are a blueprint or recipe for objects.
- An object exists in memory and has its own copy of variables to represent its state. These are known as instance variables.
- When we call a method, its acts on a particular object.

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Our First Class

- Let us write a class that manages information about persons - specifically, their name, job and pay.

Our First Class

```
# In a file called person.py

class Person:
```

This defines a new class and gives it the name Person

The Constructor

- The __init__ method is also known as a constructor. It is always called __init__
- The constructor is called when we create a new instance of a class. It is used to initialise the class, adding its instance attributes (the object’s state) and gives these some initial values.
- It is ‘just’ a function - it is called when we ‘instantiate’ an object
- We can use it just like a function, providing default arguments, etc.

Our First Class

```
# In a file called person.py

class Person:
    def __init__(self, name, job, pay):
        self._name = name
        self._job = job
        self._pay = pay
```

We add a method called __init__ - this is a constructor

The constructor is used to initialise our class.

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

self

- ‘self’ appears four times in our constructor - as an argument in the constructor’s header and also in the body of the constructor.
- self is a reference to the object instance
  - When we call the constructor for the cathal object, the self argument references the cathal object. When we call the constructor for laura, the self argument references the laura object
- We don't need to pass this when we call the constructor - it is automatically pointed at the new object we create

```
cathal = Person(‘Cathal’, ‘dev’, 55000)
```

Our First Class

```
# In a file called person.py

class Person:
    def __init__(self, name, job, pay):
        self._name = name
        self._job = job
        self._pay = pay

cathal = Person(‘Cathal’, ‘dev’, 55000)
laura = Person(‘Laura’, ‘manager’, 70000)
print(cathal._name, cathal._pay)
print (laura._name, laura._pay)
```

We can now ‘instantiate’ our class (twice) and label the new objects ‘cathal’ and ‘laura’

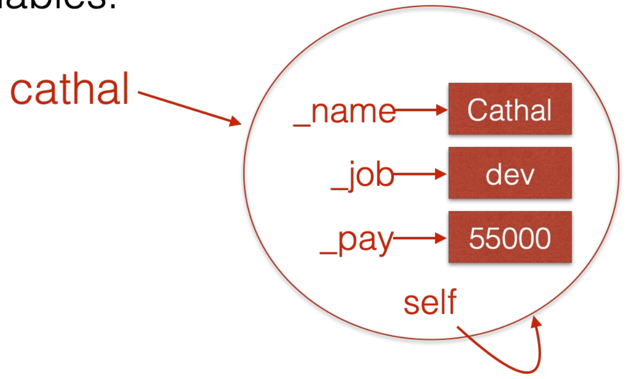
We can also access the data associated with each instance.

Under the hood…

- What happens when we call

  cathal = Person(‘Cathal’, ‘dev’, 55000)

- A new instance is created in memory. The class is used as a blueprint for this instance. The constructor is called to initialise the instance and adds instance variables.

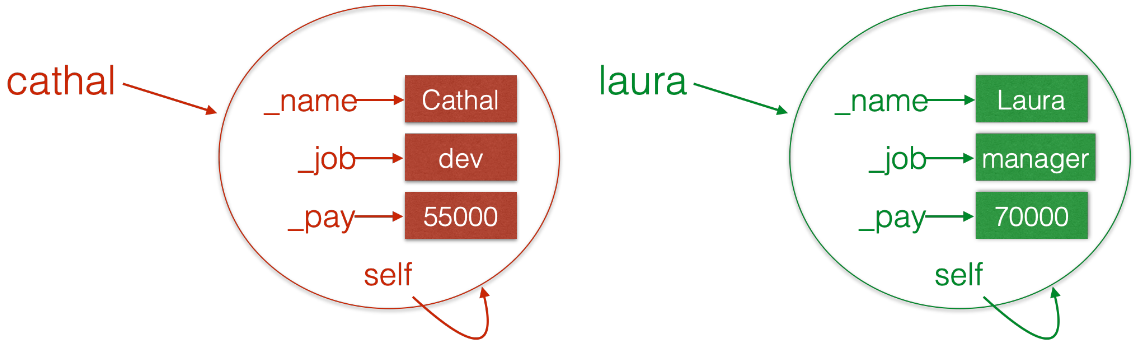


Under the hood…

- And when we call

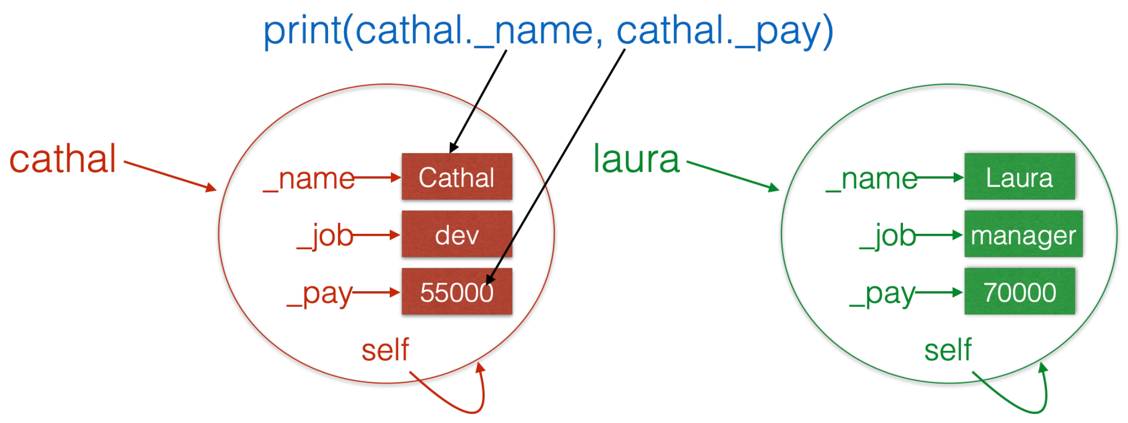
  laura = Person('Laura', 'manager', 70000)

- A second instance is created in memory. The constructor is called to initialise this new instance and adds its instance variables.



Under the hood…

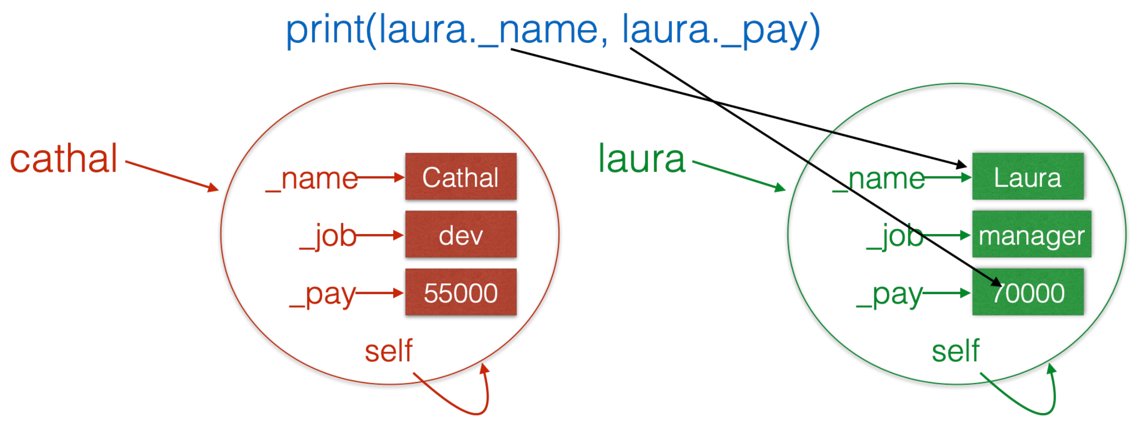
- When we call



- The name and pay associated with the ‘cathal’ object are printed

Under the hood…

- Similarly, when we call



- The name and pay associated with the ‘laura’ object are printed

Outputting Objects

- If we were to print our object we get a weird string

  print(cathal) <__main__.Person object at 0x101bd8d68>

- This isn't very useful for humans. How do we modify our code so that we can describe our object as a concatenation of its attributes.
- Python objects have several 'special' methods - we have already seen __init__ which is a constructor.
- We can use another, __str__ to return a representation of a object instance.

Outputting Objects

- __str__ generates an “informal” or nicely formatted string representation of an object. We can decide what information to show. This is especially useful when we want to test or inspect our code.

- __str__ can generate any string, but it must return a string.

Outputting Objects

#In our class we can add a method

def __str__(self):

description = ("%s %s %d" % (self._name, self._job, self._pay)) return description

#In our main() function, we can now print an instance: ...

print(cathal)

#Which for the cathal instance would output “Cathal dev 55000”

Methods

- A method is a function that belongs to a class It is like a regular function except that:
  - It is contained in a class
  - Its first parameter is self
    - This causes it to act on the member variables of a particular instance of the class

Methods

- Lets add a method to our Person class to allow modification of the pay attribute when the person is awarded a pay rise
- We will pass the rise as a percentage of the person's salary (i.e. award an n% pay rise)

Methods

```
class Person(object):
    def __init__(self, name, job, pay):
        self._name = name
        self._job = job
        self._pay = pay

    def givePayRaise(self, percentage):
        if percentage < 0 or percentage > 100:
            print("%i is an illegal percentage" % (percentage))
        else:
            self._pay += self._pay // 100 * percentage

     def __str__(self):
      ...

cathal = Person('Cathal', 'dev', 70000)
print(cathal)
cathal.givePayRaise(10)
print(cathal)
```

Next Time:

Developing Our Classes Futher