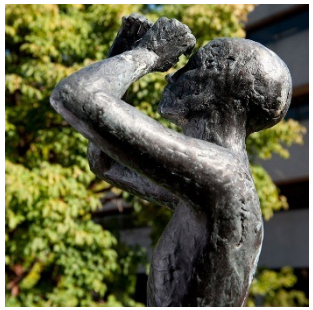


Lecture 3 – Writing a Class

CS2513

Cathal Hoare

A TRADITION OF INDEPENDENT THINKING

UCC

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Lecture Contents

Writing our First Class

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 are available on Monday. Work will be by end of Monday 25th September.

Our First Class

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

Our First Class

Keyword indicates we are declaring a class

We can name our class using any alphanumerics beginning with letter
Choose descriptive name for the human Reader.

```
class Person:

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

cathal = Person("Cathal", "Developer, 55000")
laura = Person("Laura", "Architect", 70000)
```

Our First Class

Constructor sets up or initializes our class instance. This can include several steps including adding instance variables. In this case we are adding three Instance variables. Constructor is always called
Python reserved keyword __init__

```
class Person:

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

cathal = Person("Cathal", "Developer, 55000")
laura = Person("Laura", "Architect", 70000)
```

Instance variables are variables that belong to the instance. Each instance can different values for these variables. self refers to the instance. We assign a variable by using self.[variable name] and assigning it a value. The underscore is for the human reader – it indicates that these values should be changed via a method and not directly accessed in order to maintain Encapsulation

Our First Class

The constructor can take arguments like any function or method. We can use these to pass values into the constructor. The first will always be self (we'll explain why when we examine Python's memory model)

```
class Person:

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

cathal = Person("Cathal", "Developer, 55000")
laura = Person("Laura", "Architect", 70000)
```

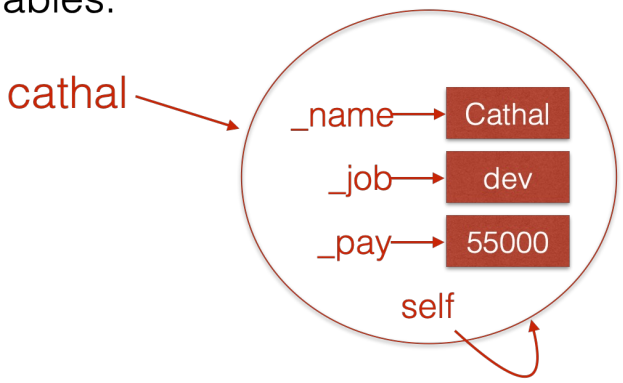
Within the constructor, we can treat the arguments as a local variable. They can be part of an expression or used in any way that a variable can otherwise be used.

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.




```
class Person:

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

cathal = Person("Cathal", "Developer, 55000")
laura = Person("Laura", "Architect", 70000)
```

self acts as a reference to the object maintained inside the class. It always refers to 'this instance' – this there because the cathal variable is outside the scope of the constructor.

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

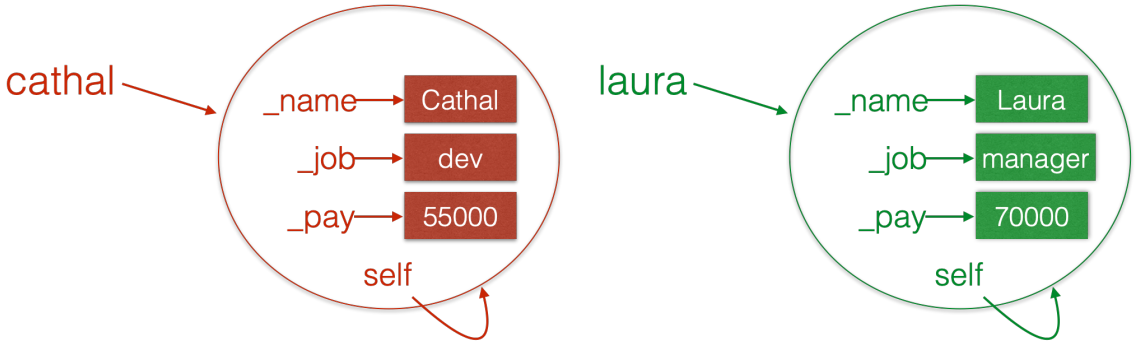
cathal = Person(‘Cathal’, ‘dev’, 55000)

Under the hood…

- And when we call

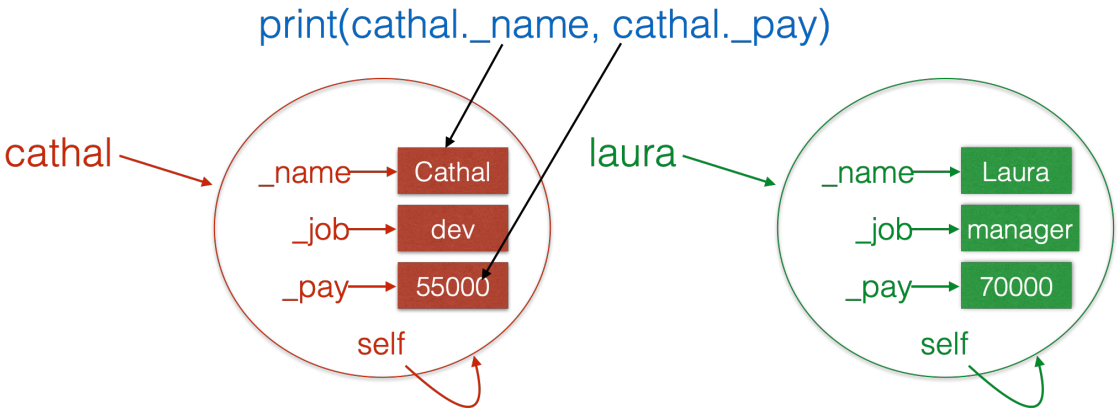
  laura = Person('Laura', 'manager', 70000)

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



Under the hood…

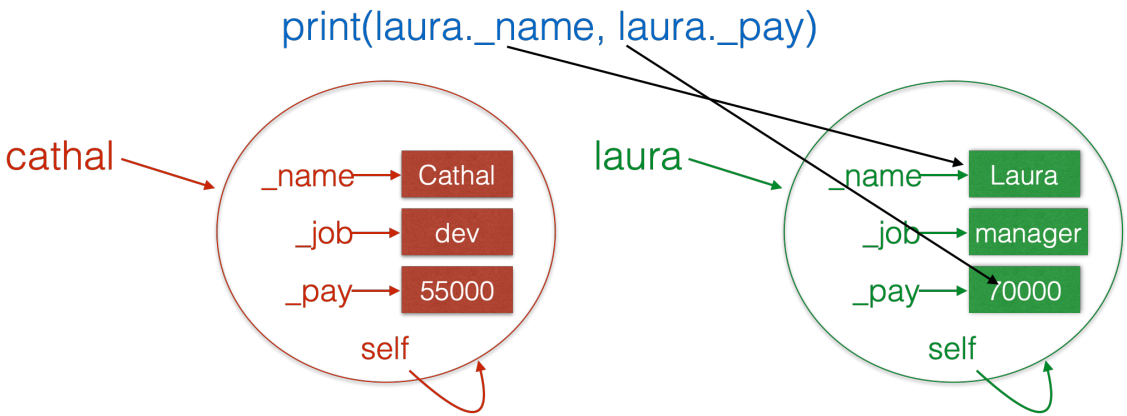
- When we call



- 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