Prepared by Julian García, based on material by
David Albrecht and María García de la Banda

Lecture 15
Classes and Objects

FIT 1008
Introduction to Computer Science

MONASH University
Information Technology

Objectives

- Learn some **Object Oriented Programming** (in Python)
- In particular, to learn:
  - How to **define basic <u>classes</u>**
  - How to **instantiate them into <u>objects</u>**
  - How to define **<u>methods</u>** and how to use them
  - An example of classes for implement stacks
  - The importance of **namespaces and scoping rules**

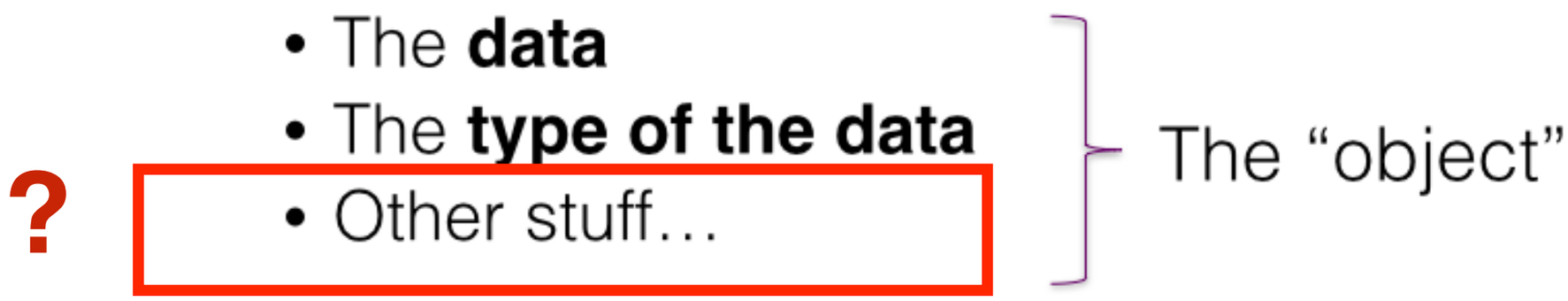
Objectives

- Learn some **Object Oriented Programming** (in Python)
- In particular, to learn:
  - How to **define basic <u>classes</u>**
  - How to **instantiate them into <u>objects</u>**
  - How to define **<u>methods</u>** and how to use them
  - An example of classes for implement stacks
  - The importance of **namespaces and scoping rules**

- We will **NOT** learn about a major OO component: **inheritance**
  - Not needed for FIT1008
  - Central to the idea of OO

Objects

- **Objects**: blocks of memory containing some data.



- Objects are more than **data** fields (or data attributes).
  They **also** have **methods** that can be performed by the object.

- Like real life: **Objects** interact with each other (through methods.)

- **Example**: a human "object" would have:
  - Data describing the human (by its attributes):
    - Name, age, height, weight, eye colour, etc
  - Methods that can be performed by/on the human:
    - Eat, sleep, run, study, etc
  - **Attributes** of the object = **Data + Methods**

- **Example**: a <u>human</u> “object” would have:
  - <u>Data</u> describing the human (by its attributes):
    - Name, age, height, weight, eye colour, etc
  - <u>Methods</u> that can be performed by/on the human:
    - Eat, sleep, run, study, etc
  - **Attributes** of the object = **Data + Methods**

- **Example**: a human "object" would have:
  - Data describing the human (by its attributes):
    - Name, age, height, weight, eye colour, etc
  - Methods that can be performed by/on the human:
    - Eat, sleep, run, study, etc
  - **Attributes** of the object = **Data + Methods**

Remember:
In Python every value is an object.

- **Example**: a human "object" would have:
  - Data describing the human (by its attributes):
    - Name, age, height, weight, eye colour, etc
  - Methods that can be performed by/on the human:
    - Eat, sleep, run, study, etc
  - **Attributes** of the object = **Data + Methods**

Remember:
In Python every value is an object.

- The **data**
- The **type of the data**
- Other stuff…

} The "object"

Using Objects

- Every value is an object with **data and methods.**

Using Objects

- Every value is an object with **data and methods.**

- So, how do we access the data and methods of an object?
  Through the **“dot”** notation:

Using Objects

- Every value is an object with **data and methods.**

- So, how do we access the data and methods of an object? Through the **"dot"** notation:

```
>>> "abcd".upper()
'ABCD'
```

Using Objects

- Every value is an object with **data and methods.**

- So, how do we access the data and methods of an object? Through the **"dot"** notation:

```
>>> "abcd".upper()
'ABCD'
```

```
>>> x = [1,2,3,4]
>>> x.append(5)
>>> x
[1, 2, 3, 4, 5]
```

Using Objects

- Every value is an object with **data and methods.**

- So, how do we access the data and methods of an object? Through the **"dot"** notation:

```
>>> "abcd".upper()
'ABCD'
```

```
>>> x = [1,2,3,4]
>>> x.append(5)
>>> x
[1, 2, 3, 4, 5]
```

- Also referred to as "**qualifying**" a variable or method.

- The **dot notation is common** to many languages (e.g. Java.)

blueprint

Objects:

blueprint

Class:

Objects:

Class:

blueprint

Objects:

Class:

blueprint

Objects:

instances…

blueprint

noun

1 *the blueprints of the aircraft and its components*: PLAN, design, draft, diagram, drawing, scale drawing, outline, sketch, pattern, map, layout, representation; technical drawing.

2 *the Thai programme provides a blueprint for similar measures in other developing countries*: MODEL, plan, template, framework, pattern, design, example, exemplar, guide, prototype, paradigm, sample, pilot, recipe.

Classes

- A class is a **blueprint** for objects, defines **attributes**: data + methods

Classes

- A class is a **blueprint** for objects, defines **attributes**: data + methods

- Classes are defined using the following syntax:

Classes

- A class is a **blueprint** for objects, defines **attributes**: data + methods

- Classes are defined using the following syntax:

```
class ClassName:
    <statement-1>
      ......
    <statement-N>
```

Classes

- A class is a **blueprint** for objects, defines **attributes**: data + methods

- Classes are defined using the following syntax:

class is a keyword

```
class ClassName:
    <statement-1>
      ……
    <statement-N>
```

Classes

- A class is a **blueprint** for objects, defines **attributes**: data + methods

- Classes are defined using the following syntax:

Convention: Class name starts with uppercase

class is a keyword

```
class ClassName:
    <statement-1>
      ……
    <statement-N>
```

Classes

- A class is a **blueprint** for objects, defines **attributes**: data + methods

- Classes are defined using the following syntax:

Convention: Class name starts with uppercase

class header

class is a keyword

```
class ClassName:
    <statement-1>
    ……
    <statement-N>
```

Classes

- A class is a **blueprint** for objects, defines **attributes**: data + methods

- Classes are defined using the following syntax:

Convention: Class name starts with uppercase

class header

class is a keyword

```
class ClassName:
    <statement-1>
      ......
    <statement-N>
```

class body

Classes

- A class is a **blueprint** for objects, defines **attributes**: data + methods

- Classes are defined using the following syntax:

Convention: Class name starts with uppercase

class header

class is a keyword

```
class ClassName:
    <statement-1>
    .....
    <statement-N>
```

body is indented

class body

Classes

- A class is a **blueprint** for objects, defines **attributes**: data + methods

- Classes are defined using the following syntax:

Convention: Class name starts with uppercase

class header

class is a keyword

```
class ClassName:
    <statement-1>
    .....
    <statement-N>
```

body is indented

class body

Statement-i are mostly "methods"

Classes

- A class is a **blueprint** for objects, defines **attributes**: data + methods

- Classes are defined using the following syntax:

Convention: Class name starts with uppercase

class header

class is a keyword

```
class ClassName:
    <statement-1>
    .....
    <statement-N>
```

body is indented

class body

Statement-i are mostly "methods"

- Every object is created by **instantiating** a class (see later).

- That is why we say that an **object is an instance of a class.**

Methods

- Define **operations** that can be performed by any object created as an instance of the class
- A method definition looks like a function definition
- Except that:
  - It must appear inside the class (in the body)
  - Its first argument must be a reference to the instance whose method was called
    - By **(a very strong) convention**, this argument is named **self**
    - You must **explicitly** add **self** to the method **definition**
    - But it is **automatically** added in a **method call**

The __init__ method

- By **convention** it is the first method in a class.
- First code **executed when creating an instance** (automatically!)
- You can choose not to define it.
- Its first argument (**self**) is a reference to the **instance** whose method was called.

The __init__ method

double underscore

double underscore

- By **convention** it is the first method in a class.

- First code **executed when creating an instance** (automatically!)

- You can choose not to define it.

- Its first argument (**self**) is a reference to the **instance** whose method was called.

Point Class

```
class Point:
    def __init__(self, x, y):
        self.x_coordinate = x
        self.y_coordinate = y
```

Point Class

don't forget **self**

```
class Point:
    def __init__(self, x, y):
        self.x_coordinate = x
        self.y_coordinate = y
```

Point Class

don't forget **self**

```
class Point:
    def __init__(self, x, y):
        self.x_coordinate = x
        self.y_coordinate = y
```

Instance variables

Point Class

don't forget **self**

```
class Point:
    def __init__(self, x, y):
        self.x_coordinate = x
        self.y_coordinate = y
```

Local variables

Instance variables

Point Class

don't forget **self**

```
class Point:
    def __init__(self, x, y):
        self.x_coordinate = x
        self.y_coordinate = y
```

Local variables

Instance variables

- **Instance variables** start with **self**:
  - Their **value** is specific to each instance.
  - Their **name** is global to all methods in the class. You can access it from every method in the class.

Point Class

don't forget **self**

Local variables

```
class Point:
    def __init__(self, x, y):
        self.x_coordinate = x
        self.y_coordinate = y
```

Instance variables

- **Instance variables** start with **self**:
  - Their **value** is specific to each instance.
  - Their **name** is global to all methods in the class. You can access it from every method in the class.

- If a variable does not start with self it is **local** to the method that binds it. Local variables <u>can't be seen</u> by <u>other</u> methods.

The argument **self** in methods in a Python class refers to:

A) The address of the **class**.

B) The address of the **object** calling the method containing the argument **self**.

C) The address of the **method** using the argument self.

D) None of the above.

Let's put this into a file called *point.py*

```
class Point:
    def __init__(self, x, y):
        self.x_coordinate = x
        self.y_coordinate = y
```

point is the name of the file where the class is defined.

```
>>> import point
```

Let's put this into a file called *point.py*

```
class Point:
    def __init__(self, x, y):
        self.x_coordinate = x
        self.y_coordinate = y
```

We **instantiate** the class by "calling it" as if it was a function with the arguments of __init__.
p1 is now an instance of Point.

point is the name of the file where the class is defined.

```
>>> import point
>>> p1 = point.Point(1,3)
```

Let's put this into a file called *point.py*

```
class Point:
    def __init__(self, x, y):
        self.x_coordinate = x
        self.y_coordinate = y
```

We **instantiate** the class by "calling it" as if it was a function with the arguments of __init__.
p1 is now an instance of Point.

innit is called automatically!

point is the name of the file where the class is defined.

```
>>> import point
>>> p1 = point.Point(1,3)
```

Let's put this into a file called *point.py*

```
class Point:
    def __init__(self, x, y):
        self.x_coordinate = x
        self.y_coordinate = y
```

We **instantiate** the class by "calling it" as if it was a function with the arguments of __init__.
p1 is now an instance of Point.

point is the name of the file where the class is defined.

innit is called automatically!

p1. "dot" lets us access attributes from the instance p1

```
>>> import point
>>> p1 = point.Point(1,3)
>>> p1.x_coordinate
1
>>> p1.y_coordinate
3
```

Let's put this into a file called *point.py*

```
class Point:
    def __init__(self, x, y):
        self.x_coordinate = x
        self.y_coordinate = y
```

We **instantiate** the class by "calling it" as if it was a function with the arguments of __init__.
p1 is now an instance of Point.

point is the name of the file where the class is defined.

innit is called automatically!

p1. "dot" lets us access attributes from the instance p1

p2 is now another instance

```
>>> import point
>>> p1 = point.Point(1,3)
>>> p1.x_coordinate
1
>>> p1.y_coordinate
3
>>> p2 = point.Point(-4,7)
>>> p2.x_coordinate
-4
>>> p2.y_coordinate
7
```

Let's put this into a file called *point.py*

```
class Point:
    def __init__(self, x, y):
        self.x_coordinate = x
        self.y_coordinate = y
```

We **instantiate** the class by "calling it" as if it was a function with the arguments of __init__.
p1 is now an instance of Point.

point is the name of the file where the class is defined.

innit is called automatically!

p1. "dot" lets us access attributes from the instance p1

p2 is now another instance

Every instance has some built-in attributes. For example **__class__**

Let's put this into a file called *point.py*

```
class Point:
    def __init__(self, x, y):
        self.x_coordinate = x
        self.y_coordinate = y
```

```
>>> import point
>>> p1 = point.Point(1,3)
>>> p1.x_coordinate
1
>>> p1.y_coordinate
3
>>> p2 = point.Point(-4,7)
>>> p2.x_coordinate
-4
>>> p2.y_coordinate
7
>>> p1.__class__
<class 'point.Point'>
```

Let's put this into a file called *point.py*

```
class Point:
    def __init__(self, x, y):
        self.x_coordinate = x
        self.y_coordinate = y
```

```
>>> import point
>>> p1 = point.Point(1,3)
>>> p1.x_coordinate
1
>>> p1.y_coordinate
3
>>> p2 = point.Point(-4,7)
>>> p2.x_coordinate
-4
>>> p2.y_coordinate
7
>>> p1.__class__
<class 'point.Point'>
```

Recap

- We create a **class** by:
  - Simply writing class Name, in some file.
  - Adding **indented statements** (such as methods) to it
  - All methods have **self** as first argument

- We create an **<u>object</u>**:
  - By instantiating the class. Calling Name() with the appropriate arguments, as given by **__init__**
  - The object has access to all the **attributes** defined by the class using the **"dot" notation** (e.g., the_stack.push)

Class variables

Variables whose values are shared by **all instances** of the class

```
>>> class Silly:
...     i = 8
...
```

Class variables

Variables whose values are shared by **all instances** of the class

Defined in the body of a class, but **outside** any methods.

```
>>> class Silly:
...     i = 8
...
```

Class variables

Variables whose values are shared by **all instances** of the class

Defined in the body of a class, but **outside** any methods.

```
>>> class Silly:
...     i = 8
...
```

Class variables

Variables whose values are shared by **all instances** of the class

Defined in the body of a class, but **outside** any methods.

```
>>> class Silly:
...     i = 8
...
>>> Silly.i
8
```

Belongs to the **class**. It exists without the object. Values are accessed through the **class**.

Class variables

Variables whose values are shared by **all instances** of the class

```
>>> class Silly:
...     i = 8
...
>>> Silly.i
8
>>> s1 = Silly()
>>> s1.i
8
```

Defined in the body of a class, but **outside** any methods.

Belongs to the **class**. It exists without the object. Values are accessed through the **class**.

Can also be accessed through instances.

Class variables

Variables whose values are shared by **all instances** of the class

```
>>> class Silly:
...     i = 8
...
>>> Silly.i
8
>>> s1 = Silly()
>>> s1.i
8
>>> s2 = Silly()
>>> s2.i
8
```

Defined in the body of a class, but **outside** any methods.

Belongs to the **class**. It exists without the object. Values are accessed through the **class**.

Can also be accessed through instances.

All instances share the same value.

Class variables

```
>>> Silly.i = 11
```

You can <u>modify</u> the value of a **class variable**

Class variables

```
>>> Silly.i = 11
>>> s1.i
11
>>> s2.i
11
```

You can modify the value of a **class variable**

All instances will share the new value

Class variables

```
>>> Silly.i = 11
>>> s1.i
11
>>> s2.i
11
>>> s1.i = 6
```

You can modify the value of a **class variable**

All instances will share the new value

Class variables

```
>>> Silly.i = 11
>>> s1.i
11
>>> s2.i
11
>>> s1.i = 6
```

You can modify the value of a **class variable**

All instances will share the new value

Modifying the value of a **class variable through the instance**?

Class variables

```
>>> Silly.i = 11
>>> s1.i
11
>>> s2.i
11
>>> s1.i = 6
>>> s1.i
6
```

You can modify the value of a **class variable**

All instances will share the new value

Modifying the value of a **class variable through the instance**?

Class variables

```
>>> Silly.i = 11
>>> s1.i
11
>>> s2.i
11
>>> s1.i = 6
>>> s1.i
6
>>> s2.i
11
```

You can modify the value of a **class variable**

All instances will share the new value

Modifying the value of a **class variable through the instance**?

?

Class variables

```
>>> Silly.i = 11
>>> s1.i
11
>>> s2.i
11
>>> s1.i = 6
>>> s1.i
6
>>> s2.i
11
```

You can modify the value of a **class variable**

All instances will share the new value

Modifying the value of a **class variable through the instance**?

?

Class variables

```
>>> Silly.i = 11
>>> s1.i
11
>>> s2.i
11
>>> s1.i = 6
>>> s1.i
6
>>> s2.i
11
```

You can modify the value of a **class variable**

All instances will share the new value

Modifying the value of a **class variable through the instance**?

?

Names and scoping!

Summary

- Classes and instances (objects)
- Methods (the **__init__** method in particular)
- Class variables