To Get Started

- Paper sheet
- Online: http://www.eecs.qmul.ac.uk/~william/CAS-London-2020.html
- Download sample programs
  - Create directory
  - Unzip
  - Recommend copy sample files before editing

Object Oriented Programming in A Level

William Marsh

[Original version co-authored with Melissa Bustamante]

Queen Mary University of London
SUPPORTED BY MAYOR OF LONDON
COMPUTING AT SCHOOL EDUCATE · ENGAGE · ENCOURAGE
KING'S College LONDON

Session Aim and Outline

Outline

- Using classes: the Face
- Attributes and the constructor
- Reflection : Decomposition and design
- *Practical break*
- Reflection: How versus Why
- Progression
- Misconceptions
- Python versus java

Aims

- Be able to motivate the use of classes and objects
- Be able to explain OOP in relations Abstraction and Decomposition
- ...progression in OOP
- Be aware of issues for teaching OOP

A Face Class: Becoming a User of Objects

There are many examples of classes and object that are familiar

Using the Face Class

- File class is a familiar example
- Are we aware we use objects?

```
from turtle import *
from Face import Face

f1 = Face(0, 0)
f1.draw()

f2 = Face(-200, 0)
f2.setSize(80)
f2.draw()
```

This is a variable. What is the type of its value?

Draw methods: which face is drawn?

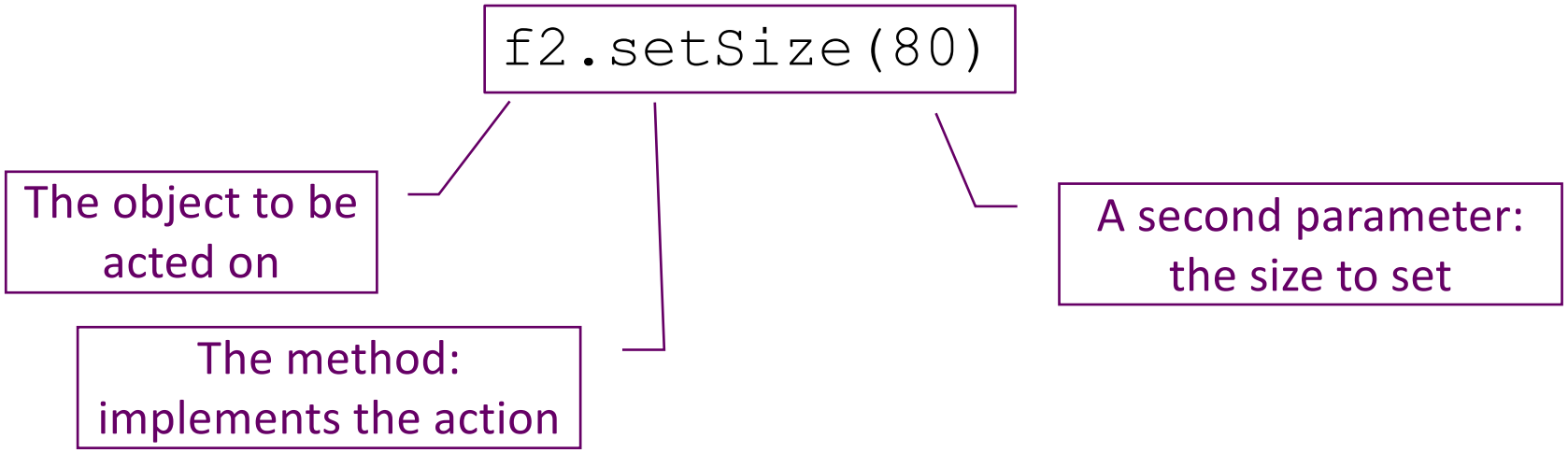
Objects and Methods

Name	Type	Description
f1, f2	Variables; Objects of 'Face' class	A drawing of a face
Face	Class name; constructor	Create a Face object
setSize	Method of 'Face' class	Set size of the face
draw	Method of 'Face' class	Draw the face

'Method' is an OO word for function

Summary: Using Objects

- Face representation is hidden
- Method act (read or update) on objects



Teaching Functional Decomposition

- You have already learnt about functions
  - How they work
  - How to use them

- Is it easy or hard to learn about functions?
  - What aspects are easier?
  - What aspects are harder?

Reflection: Abstraction and Decomposition

Motivation: What are we trying to achieve with classes?

Liskov and Guttag 1986 – Decomposition

- A very small program consisting of no more than a few hundred lines can be implemented as a single monolithic unit.
- However, as the size of the program increases such a ... structure is no longer reasonable ...
- Instead the program must be decomposed into ... modules that together provide the desired function.
- … how to decompose large programming problems into small ones … what kinds of modules are useful … [how] modules can be combined to solve the original problem

Two Different Aims for Learning OOP

How

- How to use classes
  - Create a new object
  - Use objects as variables (e.g. in a list)
- How to create (declare) new classes
  - Add method and attributes
  - … and constructors
- How to create sub-classes (inheritance)

Why

- Decomposing a problem using classes
  - Which classes to use?
  - What makes a good class?
- How to do good abstractions
  - Analysis of the problem
- How classes can interact
  - Software design

Summary

- Emphasis continuity between OOP and previous programming
  - Use of objects and methods explained
  - Abstractions implemented using functions
- Program decomposition; problem abstraction
  - Distinguish between learning syntax and
  - … practicing abstraction and program design
- OOP is a new solution to the goal of decomposition using abstraction
  - Comparison with use of functions

Practical Work

Drawing Faces: Exercises 1 and 2

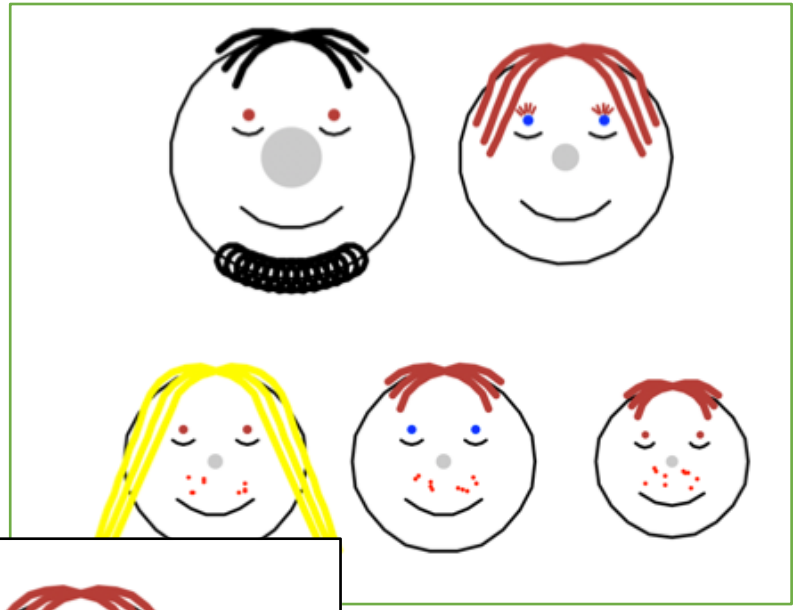
Queen Mary University of London
SUPPORTED BY MAYOR OF LONDON
COMPUTING AT SCHOOL EDUCATE · ENGAGE · ENCOURAGE
KING'S College LONDON

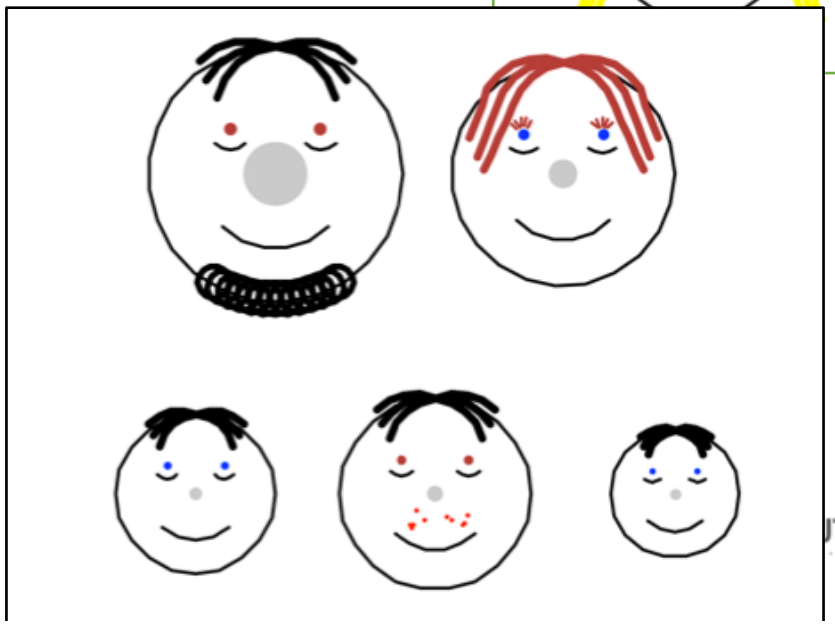
Declaring Your Own Classes

Key concepts

The Faces Example

- Using Python turtle graphics
- Good points
  - Visual and ?? Engaging (creative)
  - Class versus object distinction
  - Incremental
- Limitations
  - Not typical of OO design
  - Complexity of drawing a distraction





Class Declaration

```
from turtle import

class Face:

  def __init__(self, xpos, ypos):
    self.size = 50
    self.coord = (xpos, ypos)
    self.noseSize = 'normal'

  def setSize(self, radius):
    self.size = radius

  def draw(self):
    ...
    self.drawOutline()
    ...
```

Constructor

Attributes

Method

Class Declaration

```
from turtle import

class Face:

  def __init__(self, xpos, ypos):
    self.size = 50
    self.coord = (xpos, ypos)
    self.noseSize = 'normal'

  def setSize(self, radius):
    self.size = radius

  def draw(self):
    ...
    self.drawOutline()
    ...
```

```
  def goHome(self):
    penup()
    goto(self.coord)
    setheading(0)

  def drawOutline(self):
    penup()
    forward(self.size)
    left(90)
    pendown()
    circle(self.size)
    self.goHome()
```

Defining a Constructor

Constructor

- Has a special name
- May have parameters

```
class Face:

  def __init__(self, xpos, ypos):
     self.size = 50
     self.coord = (xpos, ypos)
     self.noseSize = 'normal'
```

Constructor name

Always ‘self’

Constructor parameter

- Don’t forget ‘self’

Initialise attributes

Attributes – Good Practice

- Attributes are not declared
  - In Python, nothing is!
- Good practice to initialise all attributes in the constructor
  - Getters do not fail
  - Clear what the attributes are
  - Do not add more

```
class Face:

  def __init__(self, xpos, ypos):
    self.size = 50
    self.coord = (xpos, ypos)
    self.noseSize = 'normal'

  def setSize(self, radius):
    self.size = radius
```

Practical Work

Drawing Faces: Exercise 3 onwards

Queen Mary University of London
SUPPORTED BY MAYOR OF LONDON
COMPUTING AT SCHOOL EDUCATE · ENGAGE · ENCOURAGE
KING'S College LONDON

Teaching OOP in Python

Queen Mary University of London
SUPPORTED BY MAYOR OF LONDON
COMPUTING AT SCHOOL EDUCATE · ENGAGE · ENCOURAGE
KING'S College LONDON

Program Structure and Complexity

If & loops

```
x =
while x > :
   y =
   print
```

Functions

Function def

Function def

Main program
- Initialise vars
- Call functions

Classes & objects

```
class Friend:
  def __ini
```

```
class Town:
  def __init__()
  def m1(a, b):
```

Main program
- Create obj
- Call methods

- Program grows **more complex** in structure
- Simpler elements remain
  - If & loop → part of function
  - Method → part of class

OOP Concepts

Concept	Details
Basic mechanics	• Calling a method of an object • Class as a template for data • Class as a collection of methods
Constructors	• Definition and use
Interaction	• Object as a value (variable, list item, …) • Object as an attribute value (has-a relationship) • Object passed as a parameter
Abstraction and modelling	• Class as a domain concept • Methods (and constructor) have parameters
Inheritance	• Superclass and subclasses • Constructor called using super() • Method inherited or overridden

Prerequisite knowledge: functions & parameters

Prerequisite knowledge: basic mechanisms

Misconception	Possible Evidence
Attributes in the wrong scope	• Omission of self (assignment or use)
Confusion between class and object	• No objects created • Only one instance • Inheritance rather than instance
Confusion between class and attribute	• Many classes – all simple
Objects only contain data	• No encapsulation • Only get and set methods
Objects do not interact	• All code in single class • Classes defined but not imported • Objects not used as attributes • Objects never passed as parameters
Believing objects are copied not shared	• Unnecessary copying • Unexpected sharing

Also lack of prerequisite knowledge: functions & parameters

Python Issues for Teaching OOP

Usual OOP

- The attributes are declared
- A class has a fixed set of attributes
- Attributes can be hidden: access only by methods

Python

- Nothing is declared
- Attributes appear when assigned to
- Hiding is not enforced

Use Python to teach OOP

- Avoid some Python tricks
- Use only a subset
- … explain later

Python versus Java

- No declarations
- Values are typed
  - Variable types are dynamic
- Run time type checking
- Syntax with indentation
- Permissive philosophy

- Declarations
- Static typing of variables
- Compile time type checking
- Syntax with braces { }
- Rigid philosophy

Summary

- Object-oriented programming
  - Builds on more basic programming
  - A approach to program decomposition (decomposition take practice)
  - Previous experience learning decomposition
- Progression: concepts not syntax
  - Proficiency with functions essential
  - Class versus object
  - Classes have attributes and methods
  - Constructor
  - Relationships between classes; objects as values
  - Inheritance
- Python – some disadvantages