Practical Sheet: OOP Programming

This sheet is a set of exercises for introducing OOP in Python using the turtle graphics. The notes assume knowledge of basic Python programming. Also available:

- An overview presentation
- Additional presentations (not used) on OOP features of Python
- A short summary of OOP in python

All materials are at http://www.eecs.qmul.ac.uk/~william/CAS-London-2018.html You need to download the zip file of sample programs.

Contents

Turtle Graphics

The exercises use the Python turtle graphics package. Turtle graphics is a simple way to draw pictures using an on-screen turtle (or cursor) that is navigated (forward, left, right). The turtle holds a pen than can be lifted up to move without drawing. Full documentation is https://docs.python.org/3.3/library/turtle.html

There is a short cheat sheet at the end of these notes.

1 Exercise 1: Using Objects

In this section we focus on using a class to create objects. The aim is to create multiple objects of the same class to reinforce the distinction between a class and objects. The code provided has two files:

1. The file face.py contains a Face class, which draws a face.
2. The drawing.py file uses the Face class to draw several faces.

Exercise 1.1: Run the Program

The following program is provided (slightly simplified):

```
from turtle import *
from Face import Face

#  start of drawing code
#  ---------------------

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

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

#  ---------------------
```

Run the example program. The code creates two Face objects and draws the faces.

Exercise 1.2: List the attributes and methods

The Face class have several attributes. Some attributes are set in the constructor; some can be set using a method. Read the code carefully and complete the following table.

Attribute	Parameter of constructor?	Function to set?
pos	Yes	No
size		Yes

Exercise 1.3: Make a More Complex Drawing

Extend the drawing program to draw more faces. Try to impress your neighbours

Summary: The Face class is a template for Face objects. We can create several instances of the Face class (i.e. Face objects), changing their size to and position to make our first picture. It's exciting isn't it and it helps to reinforce the different between a class and an object (or instance of a class).

2 Exercise 2: Adding Attributes

In this section, we modify the Face class used in Exercise 1.

- We add a method to be able to set the `noseSize` attribute.
- We add more attributes. For each attribute we should
    - Initialise the attribute in the constructor, deciding whether to use a default value or to add a value to the parameters of the constructor
    - Decide whether to provide a method to set (or get) the attribute value

Exercise 2.1: Vary the Size of the Nose

An attribute exists to vary the size of the nose. However, there is no way to set it[1]. We need to add a method to set the `noseSize`. Do this and update the drawing to call it.

Exercise 2.2: Vary the Hair Colour

To vary the hair colour we need to add an attribute to the Face class.

1. Add an attribute in the constructor – give it a default value
2. Create a method to set the constructor and update the value
3. Modify the function drawing the hair
4. Enhance the drawing to draw faces with different hair

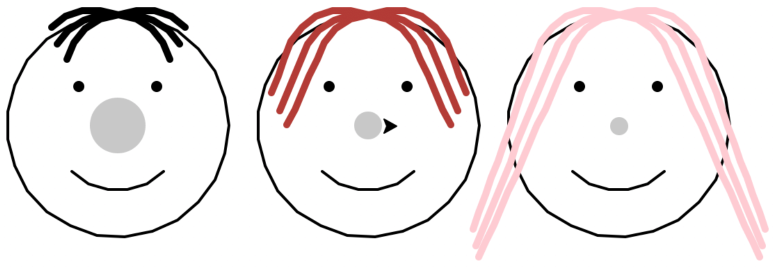
Draw faces with differ colour hair.

Exercise 2.3: Vary the Length of the Hair

The length of the hair can be varied. The main change is the number of segments of each strand of hair. As the direction of the drawing get near to 'south' (i.e. a heading of 270), you may prefer to stop the hair direction changing inwards.

It is suggested that you use a small number of lengths, described by words: e.g. short, normal , ….

Here is a possible solution:



Exercise 2.4: Other Differences

Look around the room and make a list of other possible difference. If this were an exercise for school pupils consider:

- How easy would it be to create this difference?
- What programming principle would be illustrated?

[1] If you know that this statement is not true, please do not tell anyone. We will discuss the disadvantages of using Python to teach OOP at the end of the session.

- Would the pupils find it motivating?

Summary: A class has attributes and methods. Although the class is a template for an instance. This does not imply that all instances of the class are the same. Another way to think of an object is as a box of data values: the class different objects have different values

3 Exercise 3: Multiple Classes

In this section, we create new classes. Our Face class has become quite complex and we want to add more features, so we think that some **decomposition** would be useful. We can separate more complex feature of the face into separate classes.

Initially, only the code organisation will change. Its function will stay the same.

<u>Exercise 3.1</u>: The Hair Class

A new version of the Face class has been provided that uses a separate Hair class.

Check each of the following changes in the Face class:

1. Rather than having attributes 'hairLength' and 'hairColour', a face now has a 'hair' attribute.
2. In the constructor for the hair object, we show which Face the hair is in. Look at how this is done.
3. It is still possible to change the hair colour etc. but now this is done by asking the hair to change its colour. Look at how this is implemented.
4. When drawing the face, we ask the hair to draw itself.

In the Hair class, look for the following:

1. The hair knows which face it is part of. What does it use this for?
2. Two functions are used to draw the hair: what are they?

<u>Exercise 3.2</u>: The Eye Class

Since there are two eyes, we can also create a separate class for eyes. This uses the same principles as the Hair class but is rather simpler.

Make the following changes:

1. Create the new class, remembering to import the Turtle library
2. Create a constructor. The eye needs to know:
    a. Which face it is part of (like the Hair)
    b. Which side of the face it is one (left or right)
3. Create a draw function, based on the drawEye function from the Face class but adapted to use the attributes.
4. Modify the Face class to use the Eye class
    a. Remember to import it
    b. In the Face constructor, create two new 'Eye' instance (a left one and a right one)

c. Remove the drawEye function: adapt the Face's draw function to ask the two eyes to draw themselves.

<u>Exercise 3.3</u>: Making Eye More Elaborate

Now that there is a separate Eye class, we can make eyes more variable. For example, you could have a colour attribute.

4 Exercise 4: Inheritance

In this section, we introduce inheritance. We use inheritance to create some specialised faces:

- A girl's face
- A boy's face
- A man's face
- A woman's face

<u>Exercise 4.1</u>: The GirlsFace Class

The GirlsFace class is a subclass of Face. Other ways of saying the same thing include:

- A GirlsFace is a kind of Face
- GirslsFace inherits Face
- Face is super class of GirlsFace
- GirlsFace is a specialisation of Face

A GirlsFace class has been implemented. Points to lookout for:

- It has it's own constructor, but it must also call the constructor of its superclass.
- It can call the method of its superclass – look at how this is done.

<u>Exercise 4.2</u>: Adding Eyebrows

A function to draw eyebrows has been provided but it's not used. Add it in, so that a girl has eyebrows.

<u>Exercise 4.3</u>: Create Other Specialised Faces

Based on the GirlsFace, create other specialised classes.

5 Exercise 5: A Family Class

A family contains is a collection of people (or Faces).

- Keep the people in a list.
- Have a method to add people to the family
- Have a method to draw the family portrait.

Create a family object using the specialised face subclasses to get a nice portrait. You could prompt the user for information (e.g. hair length, number of children) or generate some of it randomly.

6 Appendix: Turtle Graphics Function Reference

The Python turtle graphics package has both

- A functional interface
- An object-oriented interface

We will use the functional interface; a few capabilities are not available as a result. The full documentation is found in chapter 23 of the Python standard library.

Function	Description and Example
Move and Draw	
forward(), backward()	Move the turtle a distance: `forward(10)`
right(), left()	Turn by an angle: `right(90)`
goto()	Move to an (x, y) position: `goto(10, 50)`
home()	Move to the home position: `home()`
circle()	Draw a circle or arc. Examples: • `circle(100)` – draw a circle of radius 100 • `circle(50, 90)` – draw an arc of radius 50, angle 90
dot()	Draw a dot with diameter & colour: `dot(20, ‘blue’)`
Turtle Position	
setheading()	Set the direction of the turtle: `setheading(90)`. In standard mode, 0 is East, 90 is North, 180 is West and 270 is South.
heading()	Get the heading: `angle = heading()`
xcor(), ycor()	Get the x or y coordinates: `currentX = xcor()`
distance()	Calculate the distance from the current position to some point: `dist = distance(50, 75)`
towards()	Calculate the angle from the current position to the given co-ordinates: `towards(100, 100)`
Pen and Turtle	
pendown(), penup()	Put the pen down / up. When the pen is down, moving the turtle draws a line.
pensize()	Set the pen size: `pensize(10)` for a thick pen.
isdown()	Returns true if the pen is up.
showturtle(), hideturtle()	Show or hide the turtle. It is faster to draw without the turtle visible. (*Note: it is also possible to switch off animation altogether*)
pencolor()	Set the pen colour: `pencolor(‘green’)`

Function	Description and Example
Filling and Clearing	
fillcolor()	Set the fill colour; colours most easily entered as string though other formats supported: `fillcolor('blue')`.
filling()	Test whether shapes being drawn are to be filled.
begin_fill(), end_fill()	These command bracket drawing commands in which shapes should be filled.
reset()	Reset everything.
clear()	Clear the picture but do not reset the turtle.
write()	Write a text string: `write("Hello")` writes to the current position (which does not change) and aligns the string so that the turtle is at the left hand of the string.
Screen	
bgcolor()	Set the background colour of the screen: `bgcolor('pink')`
bgpic()	Set a picture as the background, using a file name.
screensize()	Set the screen size to e.g. 500 width and 300 high: `screensize(500, 300)`
textinput	Input text using a dialog: `textinput("Title", "Prompt")`
numinput	Input a number
window_height, window_width	Get the window height or width