Worksheet 6 – Introducing classes and objects

Classes in the real world

Classes create instances which can be saved to variables as objects. They describe what properties objects have, including what properties can change and those that can't. The ones that can't change generally are what define the particular type of object.

For example, for a pie to be a pie, it must have a pastry case and a filling. Having a pastry case cannot be changed, but is still included in the definition of a pie. What can change is the type of filling.

Classes in Python

A class describes what an object will be like and how it will behave. For example, a button is a certain size, can be clicked on, and does something. All buttons share similarities such as these (class), but what the buttons do is different from one to the next (object).

Think of a class as a blueprint or a template. It isn't something in itself; it simply describes how to make something. You can use the blueprint or template to create lots of objects that share properties, but are uniquely different.

Objects in the real world

In the real world, we use objects all the time. For example, you might eat a pie, write with a pen, or comb your hair with a brush. Of course, not all pies are the same; they come in flavours, sizes, etc. Pens come in colours and styles. The same with a brush.

Objects in Python

An object is anything that has common properties and is used multiple times. All objects are not identical, but they are similar. Their similarities come from the class they belong to, and their differences are what make them unique objects.

Let's Get Started

Below is a program that that, when run, creates a red circle every time the user clicks on the canvas.

Type up the program, then read the code breakdowns provided underneath.

```python
from tkinter import *

class Circle():

    def __init__(self, x, y, colour):
        self.colour = colour
        self.x = x
        self.y = y

    def draw(self, canvas):
        canvas.create_oval(self.x-30, self.y-30, self.x+30, self.y+30, fill=self.colour)

def onclick(event):
    newcircle = Circle(event.x,event.y,"red")
    newcircle.draw(canvas)

def main() :
    global canvas
    window = Tk()
    canvas = Canvas(window, width=450, height=300, bg="green")
    canvas.pack()
    canvas.bind("<ButtonRelease-1>", onclick)
    window.mainloop()

main()
```

Class breakdown:

```
class Circle():

    def __init__(self, x, y, colour):
        self.colour = colour
        self.x = x
        self.y = y

    def draw(self, canvas):
        canvas.create_oval(self.x-30, self.y-30, self.x+30, self.y+30, fill=self.colour)
```

`class Circle():` creates a new class called Circle. Think of this class as a blueprint – it does not actually create a circle in and of itself, it only creates a blueprint that can be customised to create one or many individual circles.

`def __init__(self, x, y, colour):` Allows an instance of the class to be created, which takes in three values; `x`, `y`, and `colour`. Think of an instance as the particular object we are making from the class blueprint, or in the case of this code, the particular circle we are creating from our Circle blueprint. **Self** is how we specifically refer to things inside the class from inside itself, such as the object we are creating from the blueprint (though the correct term for referring to an object within a class is **instance**).

- `self.colour = colour`, `self.x = x`, and `self.y = y` take whatever `x`, `y`, and `colour` were given to the class when it was called, and passes them to `self` (i.e. passes them to the new instance of the class we are creating). If we didn't include these lines, the class would be ineffective as we wouldn't be able to create objects using it.

`def draw(self, canvas):` creates a function within the class (note that functions within classes are called **methods**), which takes in the instance of the class (`self`), and a second value (`canvas`).

- `canvas.create_oval(self.x-30, self.y-30, self.x+30, self.y+30, fill=self.colour)` specifies that any instances of this class should be created with the top left corner of the circle at the value of `x-30` and `y-30`, the top right of the circle at `x+30` and `y+30`, and the colour fill set to `colour`.

Object breakdown:

```
def onclick(event):
    newcircle = Circle(event.x,event.y,"red")
    newcircle.draw(canvas)
```

`def onclick(event):` creates a new function which takes in an event to it. The event it takes in is a mouse left click which is specified in the main function by the line `canvas.bind("<ButtonRelease-1>"`.

- `newcircle = Circle(event.x,event.y,“red”)` creates a new object called `newcircle` using the Circle class. The x coordinate that the event takes place at is passed to the `x` of the `__init__` function, the y coordinate that the event takes place at is passed to `y`, and the red is passed to `colour`. Note that until this line of code or one like it, the Circle class has no idea what x, y, and colour are, only that they could potentially be passed to it.

- `newcircle.draw(canvas)` calls the `draw` method of the Circle class, which draws the shape as explained in the class breakdown above.

A note on self:

`self` is the first parameter in any method defined inside a class. Any method or variable created on the first level of indentation (that is, lines of code that start one TAB to the right of where we put class Circle) is automatically put into `self`. To access these methods and variables elsewhere inside the class, their name must be preceded with `self` and a full-stop (e.g. `self.variable_name`).

A better way to program

As we have seen, using classes and objects is good when you have more than one of the same object to create.

We are now going to look at creating the canvas using the class statement, and we do this for two reasons:

a) It is good to separate a program into a number of classes rather than lots of subroutines. This is because using many functions, especially with global variables, can make maintenance of code very difficult. This is due to the fact that it can be hard to trace where an error is coming from when multiple functions are interacting with each other.

b) Passing the assessment requires the use of at least two classes, and if you encapsulate your canvas in a class, you’re halfway there!

Encapsulating the canvas in a class

Compare the two programs below:

Program 1

```python
from tkinter import *

class Circle():

    def __init__(self, x, y, colour):
        self.x = x
        self.y = y
        self.colour = colour

    def draw(self, canvas):
        canvas.create_oval(self.x-30, self.y-30, self.x+30, self.y+30, fill=self.colour)

def onclick(event):
    newcircle = Circle(event.x,event.y,"red")
    newcircle.draw(canvas)

def main() :
    global canvas
    window = Tk()
    canvas = Canvas(window, width=450, height=300, bg="green")
    canvas.pack()
    canvas.bind("<ButtonRelease-1>", onclick)
    window.mainloop()

main()
```

Program 2

```python
from tkinter import *

class Circle():

    def __init__(self, x, y, colour):
        self.x = x
        self.y = y
        self.colour = colour

    def draw(self, canvas):
        canvas.create_oval(self.x-30, self.y-30, self.x+30, self.y+30, fill=self.colour)

class Canvascontrol():

    def __init__ (self):
        window = Tk()
        self.canvas = Canvas(window, width=450, height=300, bg="green")
        self.canvas.pack()
        self.canvas.bind("<ButtonRelease-1>", self.onclick)
        window.mainloop()

    def onclick(self,event):
        newcircle = Circle(event.x,event.y,"red")
        newcircle.draw(self.canvas)

Canvascontrol()
```

Program 1 creates a class then uses the approach we have been used to up until this point, that is, placing the main body of code in a function and calling this function on the last line of code.

Program 2 has the same Circle class as program 1, but also encapsulates the remainder of the code in a class too (the Canvascontrol class).

The detail on the differences between the two programs is:

- Program 2 uses a class with methods instead of a function for the main body of code
- “self” has been added to the start of many statements in the newly created Canvascontrol class (6 times) to refer to the instance of canvas which this class creates
- The global statements are no longer needed and have been removed

From this point onward, you should make it a habit to lay your program out like Program 2, that is, using classes to encapsulate methods. As well as being easier to maintain, this is industry standard practice.

Task 1 – Calculator with classes

In worksheet 5 (completed during week 3 of this course), you created a calculator by making ‘buttons’ which were actually a series of shapes with text in them. This involved many lines of code, many of which were repetitive.

For this task, you are to rewrite the program with each button as a separate object.

If you have your own calculator program, then modify this as you already have an understanding of it. If you don’t have the original calculator program, one has been provided for you on Moodle – *Calculator.py*, under Week 4, Python files.