INTPROG TB2: OBJECT ORIENTED PROGRAMMING

Practical Worksheet 1: Introduction to Objects and Classes

1. Introduction

This practical session will be used to get you started with creating your own classes, and then using them. Work through this worksheet at your own pace, and be sure to ask questions to clarify any concepts you find challenging. Initially, we will be using the shell to test our classes - try to predict the results before you enter the code. Please feel free to experiment until you are confident you understand what is happening and why.

Similarly to all previous practicals in Python, the indents are crucial to the successful running of the program. Ensure that all contents of the class is indented at least once (maybe more depending on the surrounding code).

We will go through the `__init__` definition (known as the constructor) and methods in detail during the lectures. However, as an overview, the constructor is used to set the initial variables for the object when it is created and methods are the functions which are required to be executed.

2. Creating Classes

In the first part of this practical worksheet, we will be designing, developing, and testing classes. In each case, you will be given a scenario, which you then need to use to design the object and its blueprint class.

All development of the classes should be conducted within the editor window, with a separate file for each class. The testing of the class will be carried out within the shell. Next week, we will be developing other files which can be used to test our classes.

2.1. Example 1 – Circle.py

Scenario

You have been asked to create a class which will model a circle. Upon creation of the objects, the user will be able to set the radius of the circle, which can then be used to calculate the area and circumference. In addition, the user would like to print a summary of the circle, with the following information: radius, area, and circumference.

2.1.1. Design

We are going to start designing the class with an empty object definition table (discussed in the lecture).

	Circle Object
Information	
Operations	

The first thing we need to think about is the information which will be required by the object. This will then become the instance variables. The circle class will hold only one piece of information: the radius. This instance variable can then be used throughout to calculate both the area and the circumference.

	Circle Object
Information	Radius
Operations	

The user requires the class to complete three tasks: calculate the area; calculate the circumference; and print information about the circle. These three operations, will become the methods when we are developing the class. The third operation, which prints information about the circle, will utilise the calculations for both the area and the circumference.

	Circle Object
Information	Radius
Operations	Calculate Area Calculate Circumference Print Information

2.1.2. Development

Now that we have a design for our class which tells us the information held (instance variables) and the operations carried out (methods) we can start to write our code.

Open IEP, and create a blank Python file called *Circle.py*. We will start this file with the following class definition:

```
class Circle:
```

Starting our Python file with the keyword class, tells the interpreter that we will now be defining a class. We need to make sure that anything which is part of the class is indented at least once. You may need more indents if the code is part of the constructor or a method.

Firstly, we need to define the constructor for the class. This is completed by defining a special type of method, which is always of the format: `__init__(self, otherParameters)`. In this case, the only parameter is the radius of the circle (as the scenario tells us the user needs to input this value on creation of the circle object). This will be assigned to an instance variable as part of the constructor code.

Note: In all the examples, I have included comments in the code to aid understanding. You do not need to enter these in your programs, unless you feel they will be of benefit to you.

```
def __init__(self, circleRadius):
      # set the instance variable, radius (passed as a parameter)
      self.radius = circleRadius
```

Next, we need to create the methods which are implemented by the class. From the design, we can see that there should be three methods: `calculateArea()`; `calculateCircumference()`; and `retrieveInformation()`. For all the methods, a

value is returned. These values can then be used by other methods, if necessary. As part of this example, the values from `calculateArea()` and `calculateCircumference()` will be used as part of the `retrieveInformation()` method.

```
# method to calculate the area of the circle
def calculateArea(self):
      # calculate the area using the radius of the circle
      area = math.pi * (self.radius ** 2)
      # return the area of the circle
      return area

# method to calculate the circumference of the circle
def calculateCircumference(self):
      # calculate the circumference using the radius
      circumference = 2 * math.pi * self.radius
      # return the circumference of the circle
      return circumference

# method to provide information about the circle
def retrieveInformation(self):
      # create a string containing radius information
      infoString = "A circle of radius {0:0.4f}".format(self.radius)
      # add the area information
      infoString += "\nhas an area of {0:0.4f}".format(self.calculateArea())
      # add the circumference information
      infoString += "\nand circumference of {0:0.4f}".format(self.calculateCircumference())
      # return the string containing the information about the circle
      return infoString
```

Note:

`aString += "another string"` *is the same as typing* `aString = aString + "another string"`

just like:

`counter += 1` *is the same as typing* `counter = counter + 1`

and:

`counter -= 1` *is the same as typing* `counter = counter - 1`

Note 2:

You will notice that some variables have the keyword self in front of them. You will find out more about the 'self' keyword in lecture 2.

'self' refers to the instance of the class. If you use 'self' before a variable, it becomes an instance variable. If you don't, the variable will remain local to the method it is created in (i.e. when the method finishes running the variable will no longer exist, and it is not accessible outside the method).

To ensure that the `math.pi` function works correctly, import the math package before the class definition.

You will notice that in all the methods an additional variable is created without the "`self`" keyword. Using the "`self`" keyword makes the variable an instance variable. By leaving it out, the variable becomes one which can only be accessed within the method. When the method is finished, the variable is no longer available for use. In this example, we do not need to be able to access the `area`, `circumference`, and `infoString` after the method has finished running. We can use the result, as it has been returned by the method.

Also, the `infoString` uses the methods which have already been created. It does this by calling them using `self.calculateArea()` and `self.calculateCircumference()`.

As the methods only use the instance variable contained within the class, no parameters are required to be passed. However, if the method requires additional information, this can be passed to it through the use of parameters (this will be demonstrated in the next example).

The final Circle class should be as follows (without the extra comments):

```
import math

class Circle:

	# class constructor
	def __init__(self, circleRadius):
		self.radius = circleRadius

	# method to calculate the area of the circle
	def calculateArea(self):
		area = math.pi * (self.radius ** 2)
		return area

	# method to calculate the circumference of the circle
	def calculateCircumference(self):
		circumference = 2 * math.pi * self.radius
		return circumference

	# method to provide information about the circle
	def retrieveInformation(self):
		infoString = "A circle of radius {0:0.4f}".format(self.radius)
		infoString += "\nhas an area of {0:0.4f}".format(self.calculateArea())
		infoString += "\nand circumference of
		{0:0.4f}".format(self.calculateCircumference())
		return infoString
```

2.1.3. Testing

Now that we have our class file, we need to make sure that it works as we intended, and there are no syntax errors. For this example, we will be using the Python shell. Later in the worksheet, we will be developing a main method which can use the class to create objects, and test the methods.

Firstly, run the Circle.py file as a script. This will then load the class definition (and therefore, the object blueprint) into the shell. If all works well, all you will have is a blank shell prompt:

```
Kernel process terminated for restart. (0)

Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) on Windows (32 bits).
This is the IEP interpreter with integrated event loop for TK.
Type 'help' for help, type '?' for a list of *magic* commands.
Running script: "N:\MyDocuments\Teaching\INTPROG 2017-18\TB2\Wk1\Practical\Pyth
on Files\Circle.py"

>>>
```

In the shell, enter the following:

```
circle1 = Circle(4)
```

This will create an object called circle1, which has a radius of 4 stored as an instance variable. We can confirm that this is the case, by typing the following into the shell:

```
print("The radius of circle1 is:", circle1.radius)
```

We can now call the methods which are implemented within this class to ensure the class does carry out all the requirements of the scenario.

Start by calling the `retrieveInformation()` method to print all the key data relating to the circle. In the shell enter:

```
print(circle1.retrieveInformation())
```

You will see that this method will print the radius, area, and circumference of the circle. Before moving on, confirm that the output produced by the method is correct.

We can also call each of the other methods which are implemented within the class using:

```
print("The area of the circle is", circle1.calculateArea())
print("The circumference of the circle is", circle1.calculateCircumference())
```

We will now create another object of type Circle, and confirm that there is no effect on circle1. At the shell prompt enter the following:

```
circle2 = Circle(24)
print("circle2:", circle2.retrieveInformation())
print("circle1:", circle1.retrieveInformation())
```

As you can see from this example, each time we create a new instance of the class (i.e. a new object), there is no change to the previous objects.

2.2. Example 2 – BankAccount.py

Scenario

The bank would like to be able to model a bank account using Python. Upon creation of the bank account, they need to store the customer name, the account number, the current balance (which will initially be set to 0), and the overdraft amount. Once the bank account has been created, they would like to be able to deposit and withdraw money, check the current balance, see what the current overdraft is, and finally, to see how much funds are available (i.e. how much money can be taken out of the account including the overdraft). When money is withdrawn from the account, the overdraft must be taken into account – i.e. they are not able to withdraw more money than the person has available to them.

2.2.1. Design

From the scenario, we can see that a class is required. Each bank account that is opened will then become objects. Before we start to code the class, we need to consider its design. Before reading on, try to work out and complete the design for the Bank Account class:

	Bank Account Object
Information	
Operations	

Firstly, we need to decide what information is required by the Bank Account object. The scenario says that four pieces of information are required: the name on the account, the account number, the current balance, and the overdraft limit.

	Bank Account Object
Information	Account name Account number Current balance Overdraft limit
Operations	

Once the information held required by the object is known, we can then move on to deciding what operations are required. In this case, we will require the bank account to: deposit some money, withdraw some money, show the current balance, show the current overdraft limit, and show the current amount of funds which are available to the account owner.

	Bank Account Object
Information	Account name Account number Current balance Overdraft limit
Operations	Deposit some money Withdraw some money Show current amount Show overdraft limit Show current funds

We can now use this design to create the class in Python. The information becomes the instance variables, and the operations are the functions which are available to the object.

2.2.2. Development

The development is started by creating a new class file within IEP called: *BankAccount.py*.

We need to start by declaring the class, and writing the constructor. In the scenario, it is stated that the current balance is initially set to 0. Therefore, there is no need to include this within the constructor parameters. We do need to create the current balance instance variable in the constructor body, but we can automatically set it to 0 (see the currentAccount variable in the below code block).

```
class BankAccount:

	# class constructor
	def __init__(self, name, number, overdraft):
		self.accountName = name
		self.accountNumber = number
		self.currentAmount = 0
		self.accountOverdraft = overdraft
```

Now that we have the constructor for the class, we need to start thinking about the methods which will be required as part of the scenario. From the design, we can see that five methods will be required: deposit money; withdraw money; get current amount; get overdraft limit; and get current amount of funds. When developing the method which will be used to withdraw money, we need to either return True (if the money has been withdrawn) or False (if there is insufficient funds). This will allow us to use the method elsewhere, and know whether it has been possible to withdraw the funds.

```
# method to allow money to be deposited into the account
def deposit(self, depositAmount):
	# increase the amount instance variable by the amount deposited
	self.currentAmount += depositAmount
```

```
# method to allow money to be withdrawn from the account
def withdraw(self, withdrawAmount):
	# calculate the total amount which is available, taking into account the overdraft
	totalAmountAvailable = self.getAvailableFunds()
	if withdrawAmount > totalAmountAvailable:
		# withdrawal not possible due to lack of funds
		return False
	else:
		# withdrawal possible, money in the account should be decreased
		self.currentAmount -= withdrawAmount
		return True

# method to get the current amount of money in the bank account
def getCurrentBalance(self):
	# return the amount of money in the account
	return self.currentAmount

# method to get the current overdraft limit
def getCurrentOverdraft(self):
	# return the overdraft limit
	return self.accountOverdraft

# method to get the current amount of funds available (including the overdraft)
def getCurrentAvailableFunds(self):
	# return the total amount of available funds
	return self.currentAmount + self.accountOverdraft
```

The final BankAccount class should be as follows (without the extra comments):

```
class BankAccount:

	# class constructor
	def __init__(self, name, number, overdraft):
		self.accountName = name
		self.accountNumber = number
		self.currentAmount = 0
		self.accountOverdraft = overdraft

	# method to allow money to be deposited into the account
	def deposit(self, depositAmount):
		self.currentAmount += depositAmount

	# method to allow money to be withdrawn from the account
	def withdraw(self, withdrawAmount):
		totalAmountAvailable = self.getAvailableFunds()
		if withdrawAmount > totalAmountAvailable:
			return False
		else:
			self.amount -= withdrawAmount
			return True

	# method to get the current amount of money in the bank account
	def getCurrentBalance(self):
		return self.currentAmount

	# method to get the current overdraft limit
	def getCurrentOverdraft(self):
		return self.accountOverdraft

	# method to get the current amount of funds available (including the overdraft)
	def getCurrentAvailableFunds(self):
		return self.currentAmount + self.accountOverdraft
```

2.2.3. Testing

In the same manner as we tested the Circle class, we can now check the BankAccount class is functioning correctly, with no syntax errors.

Load the BankAccount class definition into the shell by running the file as a script.

```
Kernel process terminated for restart. (0)

Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) on Windows (32 bits).
This is the IEP interpreter with integrated event loop for TK.
Type 'help' for help, type '?' for a list of *magic* commands.
Running script: "N:\MyDocuments\Teaching\INTPROG 2017-18\TB2\Wk1\Practical\Pyth
on Files\CashMachine Example\BankAccount.py"

>>>
```

In the shell, create a bankAccount object to allow you to test your BankAccount object.

```
bankAccount = BankAccount("Alice Smith", 29471038, 1000)
```

We can then verify that all the instance variables have been created correctly:

```
print("The account name is: ", bankAccount.accountName)
print("The account number is: ", bankAccount.accountNumber)
print("The amount in the account is: ", bankAccount.currentAmount)
print("The overdraft amount for the account is: ", bankAccount.accountOverdraft)
```

We can then test all the methods which have been included within the class definition. Try to work out what the answer should be before you enter each command into the shell.

```
bankAccount.deposit(1050)
print("The amount in the account after the deposit is: ", bankAccount.currentAmount)

bankAccount.withdraw(50)
print("The amount in the account after the withdrawal is: ", bankAccount.currentAmount)

bankAccount.withdraw(20000)
print("The amount in the account after the withdrawal is: ", bankAccount.currentAmount)

print("The current balance is: ", bankAccount.getCurrentBalance())
print("The current overdraft is: ", bankAccount.getCurrentOverdraft())
print("The current available funds are: ", bankAccount.getCurrentAvailableFunds())
```

Note:

Don't forget that the withdraw account will either return true or false, rather than an error message (you will see why in the next practical sheet, when we complete the CashMachine example).

2.3. Practice Questions

These questions follow the same format as the previous two examples. They all provide a scenario. You will need to design the class required as part of the scenario, and implement the code. Save this file to your N: drive (of wherever you are saving your INTPROG work), then you can highlight the important information and complete the class design diagrams on the worksheet. Make sure you test all of your classes before moving onto the next question (in the same manner as we tested the Circle and BankAccount classes). Make sure that all the instance variables, and method names are appropriate to the task they are required for (i.e. don't forget about code quality and readability!).

1. You have been asked to create a class which models a rectangle. Upon creation of the objects, the user will be able to set the height and width of the rectangle, which can then be used to calculate the area, perimeter, and the diagonal length between the opposing corners. In addition, the user would like to return a summary of the rectangle, with the following information: height, width, area, perimeter, and the diagonal length.
   Note: to calculate the diagonal length use the following formula: $diagonal = \sqrt{height^2 + width^2}$

Information	
Operations	

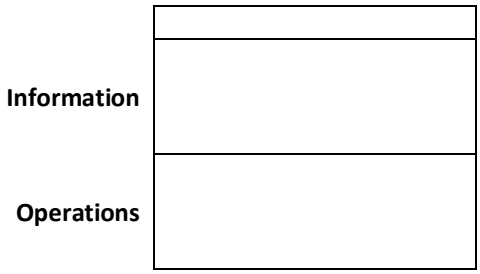
2. You need to create a class which models a coin. Upon creation of the objects, the user should be able to define the value of the coin in pence. In addition, the user should be able to use the class to simulate flipping a coin, returning either heads or tails. You can assume that the coin is fair (i.e. 50% chance of it landing on heads).
   Note: return to week 8 for a reminder of using the random module.

Information	
Operations	

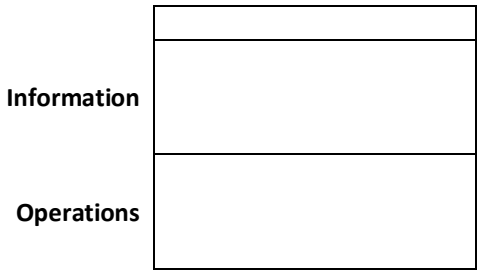
3. Create a class which can be used to model a Dice. Upon creation, the user can define the labels which will be on each side of the dice, using a list. They are not limited to the number of sides the dice will have. The class will be required to simulate throwing the dice, returning the label of the side the dice lands on. You can assume that the dice is fair, therefore, the likelihood of landing on any side is equal.

Information	
Operations	

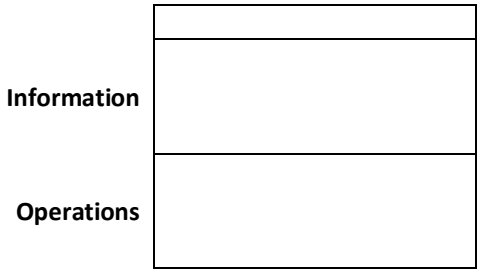
4. A bookshop requires you to create a class which can be used to store information about the books they sell. Upon creation the class will need to contain the title, author, number of pages, ISBN number, price, and the number of copies available. The number of copies can be assumed to be 0 upon creation of the object. The bookshop want to be able to change the price of the book, sell a copy of the book, and restock the book. In addition, they want to be able to retrieve all the information held about the book.



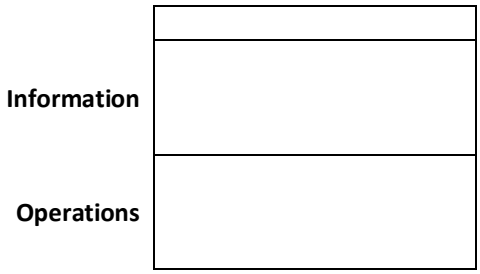
5. A university require a class which can be used to model a lecturer. The university need to hold the following information: first name, last name, salary, and units the lecturer teaches. All this information will be set at the point that the object is created. The university needs to be able to: add new units to the teaching list; remove units which are no longer; and give the lecturer a pay rise. In addition, they want to be able to retrieve all the information they have about the lecturer.



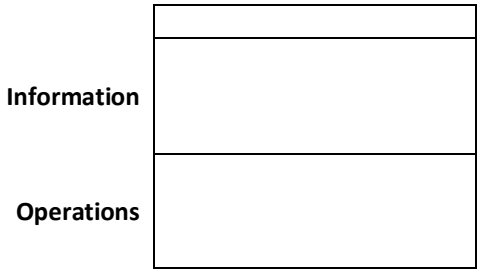
6. You have been asked to create a class which can be used to model a combination padlock, which can use both letters and numbers. Upon creation of the padlock object, an initial code should be set. It is assumed that the padlock will be locked when it is created. The user should be able to open the lock (when the correct code is entered), and close it again. Additionally, the user should be able to change the combination. However, the combination can only be changed once the padlock is open, and the new code should be of the same length as the previous one. When opening the lock, and changing the code, appropriate error messages are required when incorrect parameters are used.
<u>Hint:</u> in the constructor, create an additional instance variable which can be used to support fixing the length of the combination. You will not need to use input statements to get the combination.



7. You have been asked to create a class which models a student. The university need to store the following information about the student: first name, last name, student ID, email address, course, fees payment status, whether they are a current student. The student ID is generated from the first letter of the forename and the surname (e.g. for Bob Smith the ID would be bsmith). All email addresses follow the format studentID@university.ac.uk (e.g. Bob Smith would be bsmith@university.ac.uk). The first name, last name, and course for the student need to be set on creation of the object. It can be assumed at creation that the student is current, and has not paid their fees. The university administrators will need to be able to change the student's course, record their fees as paid (the amount is not required to be stored), withdraw them as a student (i.e. make them not a current student), re-enrol the student (i.e. make them a current student again), and retrieve all the information they hold on the student.



8. You have been asked by the library to create a borrower class. The library would like the object to hold the following information: first name, last name, borrower number, maximum number of books which can be loaned, the number of books currently on loan (assume this to be 0 initially), and the outstanding fine accrued by the borrower (assume this to be 0 initially). The first name, last name, borrower number, and maximum number of books they can loan should be set when the object is created. The library need to be able to change the maximum number of books the person can loan, loan books to the person, return books, associate a fine with the borrower, record that the borrow has paid some of their fine, and retrieve the information held about the borrower. There is no need to store which books have been borrowed, just the number. There needs to be appropriate error messages when inappropriate parameters are entered. In the case of paying the fine, if there is an overpayment, the class needs to print the change.



9. You have been asked to create a class which will model the contents of a till. The following are valid contents of the till: 1p, 2p, 5p, 10p, 20p, 50p, £1, £2, £5, £10, £20, £50. Upon creation, the user should be able to define the quantity of each item in the till. The users will need to be able to add and remove a specific items. In addition, they will need to be able to calculate the total number of coins and the value of everything within the till.
Hint: instead of creating individual variables (in the parameters and constructor), use a list to hold the contents of the till. Look at practical 9 from the last teaching block for a reminder of the methods available for working with lists.