823G5 Programming in Python

Aim

To familiarise yourself with

- Object Oriented Programming
- Creating and using classes
- Classes and objects
- Getting classes to work together
- Class attributes

Exercises

Download the `Lab4Exercises` ZIP file. It contains some basic code for you to develop. The code is a set of Python source code files (`.py`). You will need to open a new PyCharm project and import the `.py` files into it. Refer to the Lab 1 instructions if you cannot remember how to do this.

Hints

Some key points for you to remember:

- A class is a design for code that represents some useful entity that forms a part of a solution to a bigger problem.
- Objects (instances) are instantiated using the notation `x = SomeClass()`. Remember to include the parentheses and any arguments/parameters required.
- Many objects can be created using the same class. Each object is an individual thing in its own right.
- Classes are characterised by attributes and methods. A method is a function that belongs to a class.
- `self` is a self-referential operator that means “this object”. It is used as the first parameter in the definition of methods and to refer to instance variables.
- A class can have an optional constructor method called `__init__` that can set up an object for you. The constructor method is invoked automatically when the object is instantiated.

Coding challenge 1

Open the `challenge1.py` file. It contains the start of two class definitions and a `main()` function. The two class definitions are incomplete. The two classes are:

- `Student`: to represent a university student with attributes `firstname`, `lastname`, `course` and a list of module codes they are enrolled on.
- `Module`: to represent a single study module at the university. A module has a `name`, a `code`, a `tutor` and a list of all students enrolled on that module.

The `Student` class needs to implement the following methods:

- An appropriate constructor.
- `show_details()` to display all information about a student including all the module code for the modules they study.
- `change_course()` that allows a student to change course.

The `Module` class needs to implement the following methods:

- An appropriate constructor.
- `enrol_student()` that takes a student and enrolls them on the module.
- `show_all_enrolled_students()` that displays information about all students enrolled on that module.

Your challenge is to complete the class definitions such that the `main()` function can do its work. A sample output might look something like this:

```
Ken Barlow on course: English
Enrolled on following modules:
A101
E102
Mike Baldwin on course: Business
Enrolled on following modules:
A101
Harold Legg on course: Medicine
Enrolled on following modules:
E102
Enrolled on module: A101
Ken Barlow
Mike Baldwin
Enrolled on module: E102
Ken Barlow
Harold Legg
No students for M105 yet
Ken Barlow on course: Engineering
Enrolled on following modules:
A101
E102
```

A suggested answer is provided on Canvas.

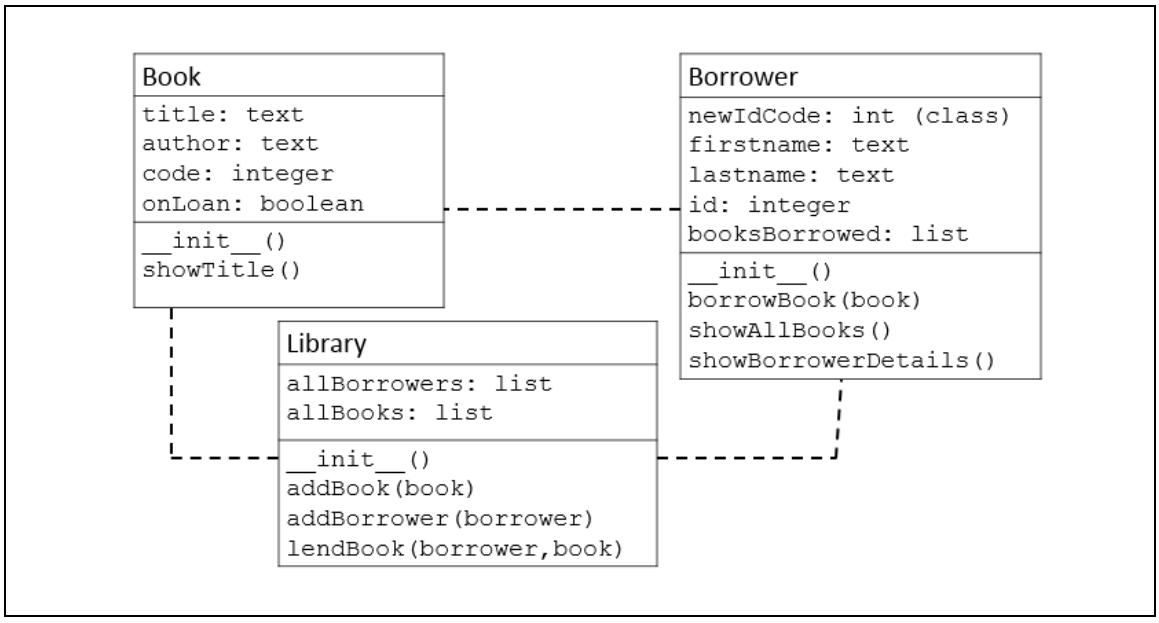
Coding challenge 2

For this challenge, we shall create a 3 class solution to model a lending library. The 3 classes are:

- `Borrower`: a person that can borrow a book.
- `Book`: an item that can be borrowed.
- `Library`: that has borrowers and books and can lend books.

So, the idea is that the `Library` has a collection (a list) of books and a collection of borrowers. It is the library's job to lend books, and the borrower's role to know what they borrowed.

Open the `challenge2.py` file. It has the start of the 3 class definitions and a `main()` method. We can see what the intended structure of this solution will be using a class diagram. A class diagram shows the associations between classes and the key attributes and methods:



There is one new concept we have not seen before. We are used to the idea that classes have attributes. Each object has its own copy of those attributes. We refer to these as the instance attributes. However, we can also have an attribute that is shared (common) to all instances of a class – this is called a class attribute. We can see one in action in the constructor `(__init__)` method in the `Borrower` class:

```
class Borrower:
    """Represents a person who can borrow books."""

    new_id_code = 1  # Class variable

    def __init__(self, firstname, lastname):
        self.firstname = firstname
        self.lastname = lastname
        self.id = Borrower.new_id_code
        Borrower.new_id_code += 1  # Updates for next new borrower ...
        self.books_borrowed = []
```

Notice that the attribute `new_id_code` is outside the scope of any method. This makes it a class attribute. `id` on the other hand is clearly an instance attribute. Every time we create a new `Borrower`, we want the `id` code to be unique, so that every borrower has their own code. When we want to access the `new_id_code` we use the syntax `Borrower.new_id_code`. Note that `Borrower` is the class name rather than an object reference as we have seen before. If any instance of `Borrower` changes `new_id_code`, that value is immediately seen across all instances of the `Borrower` class. There is only one copy of the `new_id_code` attribute. So, in this example, when the constructor method is invoked, it uses the current value of the class attribute `Borrower.new_id_code` to set the value of the instance attribute `id` and then increments the value of `Borrower.new_id_code` so that it is different next time a `Borrower` is instantiated. This is a common usage for class attributes.

Your task is to implement the attributes and methods set out in the class diagram such that the `main()` function can do its job. When you have it working, the output should look something like this:

```
Lending book Kafkas Motorbike to Kevin Wilson
Kevin Wilson id: 1
Title: Kafkas Motorbike
Lending book History of Bagpipes to Rita Shapiro
Rita Shapiro id: 2
Title: History of Bagpipes
Sorry that's already on loan.
```

An example solution is available on Canvas.

Dr. Benjamin Evans
B.D.Evans@sussex.ac.uk