# DCU School of Computing
# Assignment Submission

| | |
|---|---|
| **Student Name:** | Kyrylo Khaletskyy |
| **Student Number:** | 15363521 |
| **Programme:** | BSc. in Computer Applications |
| **Project Title:** | Comparing Imperative and Object-Oriented Programming |
| **Module code:** | CA341 |
| **Lecturer:** | David Sinclair |
| **Due Date:** | 20/11/17 |

## Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying is a grave and serious offence in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion, or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged, and the source cited are identified in the assignment references.

I have not copied or paraphrased an extract of any length from any source without identifying the source and using quotation marks as appropriate. Any images, audio recordings, video or other materials have likewise been originated and produced by me or are fully acknowledged and identified.

This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. I have read and understood the referencing guidelines found at http://www.library.dcu.ie/citing&refguide08.pdf and/or recommended in the assignment guidelines.

I understand that I may be required to discuss with the module lecturer/s the contents of this submission.

Signed: Kyrylo Khaletskyy

Date: 17/11/2017

## Preliminary

I have chosen to implement both of my styles in Python 3. This is a very versatile language which works well with both Object Oriented and Imperative solutions. As mentioned in the comments I read in a set of tasks and events from a separate text file in the OO design, allowing easy addition of new tasks/events. Below you can see the sample input copied from a text file and how to run both programs from terminal. Both solutions add all valid items to a queue, and remove the first 2 items which were added, as a test both solutions print the remaining items in the queue in the correct order, the first 2 items in the input should not be printed as they have been removed previously by using the remove_item method in the OO solution and using list manipulation in the Imperative solution.

```
13/11/17, 13:00, 02:30, [‘Benny’, ‘Steve’, ‘Bob’, ‘Alan’]
17/11/17, 11:00, Henry Grattan
10/11/17, 10:00, 02:00, [‘Mary’, ‘John’, ‘Scott’]
18/11/17, 12:00, Computing Building
```

```
python3 object_oriented_todo.py < todo.txt
python3 imperative_todo.py
```

## Imperative Programming

Imperative programming is a paradigm that executes a sequence of guided steps, in which variables are changed and/or referred to change a programs’ state. Imperative programs define sequences of commands for a computer to perform and the order of each statement is very important as with each iteration of a line the program changes state. The benefits of imperative programming are that it is often much faster to write a small scale program requiring a lot less syntax and often much easier to read follow through the code. The disadvantages of imperative programming are that it is hard to add new code/functionality to your program and your program can usually only do one thing.

## OO Programming

Object Oriented programming is a paradigm which represents everything as an object. Each object contains its own methods which are specific to the data type in question, algorithms are created with the specific object in question and may not suite other objects. Because objects operate undependably, they are encapsulated into modules with contain both local variables and methods. Object Oriented programming follows Imperative programming but rather than stepping through the whole program line by line the data jumps from one chunk of code to another depending on the object and method called. The benefits of object oriented programming greatly outweigh that of Imperative programming. Code can be easily reused and recycled, meaning programmers can use the objects they’ve written in before if their new program contains same or similar objects. Large projects are very hard to write, object oriented programs help programmers with extensive planning and aid in splitting up the program into more manageable parts.

## Comparison of Both Styles

The OO solution begins with a simple piece of code shown below which converts a string imported from an external text file into a list containing each element or in the case of Tasks it converts it to 4 elements and a list of people attending in the last position. Both solutions discard anything that doesn't confine to the format of a Task or an Event. Reading in from a text file allows for a greater versatility in adding new Tasks or events to the list. Once in the correct format the Tasks/Events are added into a larger list of items, now formatted.

```
def main():

    infile = open('todo.txt').read().splitlines()
    act = []

    for line in infile:
        line_list = line.split(', ', maxsplit=3)
        if len(line_list) == 4:
            line_list[3] = line_list[3].strip("[]").split(', ')
            act.append(line_list)
        elif len(line_list) == 3:
            act.append(line_list)
        else:
            continue
```

In the OO style the list of Tasks/Events is passed to the InOut function, which identifies whether an item is an Event or a Task and adds the item to the queue using the add_item accordingly. But before it is added the item in question is passed through to either the Task or Event Class, where it is converted to fit the following formats using a __str__ method:

```
Date: {}, Start Time: {}, Location: {}
Date: {}, Start Time: {}, Duration: {}, Assigned: {}
```

Doing so allows the programmer to quickly and easily add more objects if needed, and overall it is much easier to read and modify objects to fit some new criteria. Furthermore, when working on large scale projects it is easier to assign a programmer to write a Class with the expected inputs and outputs, this prevents programmers from tampering with code that they shouldn't. Examples of both the Event and Task Classes are shown below.

```
class Event():
    def __init__(self, line):
        self.date = line[0]
        self.start = line[1]
        self.location = line[2]

    def __str__(self):
        return "Date: {}, Start Time:
```

```
class Task():
    def __init__(self, line):
        self.date = line[0]
        self.start = line[1]
        self.duration = line[2]
        self.people = line[3]

    def __str__(self):
        return "Date: {}, Start Time:
```

On the other hand, the Imperative solution takes each item for the large list of items and formats it the same style shown above. Using a list, the items retain their order which is very important as we are dealing with queues. Initially we can see that the code itself is more concise and took less time to write but not without its drawbacks. Addition of new objects would be very hard and there is a high chance of error, especially when working on a large scale project. Adding and removing additional items to a list would also prove difficult unlike in the OO Style where I can just call the add_item or remove_item method.

To contrast both solutions, in the OO solution I built a queue, it has 3 main methods, isEmpty which checks if a list is empty, enqueue which inserts an item at the start of a list and dequeue which pops an item from a list as shown below. The Imperative solution on the other hand, uses Pythons list manipulation to add items to a list or remove the first item which was added to the list, retaining the main principles of queues without the use of any Functions or Classes.

**Imperative:**

| | |
|---|---|
| Add: | qu[i] = "Date: " + line[0] + ", Start Time: " + line[1] + ", Location: " + line[2] |
| Remove: | qu = qu[1:] |

**Object Oriented:**

```
class Queue:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def enqueue(self, item):
        self.items.insert(0,item)

    def dequeue(self):
        return self.items.pop()
```

```
class ToDo():
    def __init__(self, queue):
        self.queue = queue

    def add_item(self, item):
        self.queue.enqueue(item)

    def remove_item(self):
        return self.queue.dequeue()
```

## Conclusion

Based on my research and practical approach to writing the same program in both styles I believe that both methods have their benefits and drawbacks, but conclude that unless you are writing a very small program which you would also like to convert to another language (as other languages may not have similar inbuilt functions) with single type objects you should use an Imperative programming style, but for larger projects, even one such as a to do list which I have worked on I feel like you should implement it using Object Oriented programming to aid in better code, ability to add new objects and the ease of modifying current objects and methods to suite the new requirements.

## References

EECS. Major Programming Paradigms [Online].
Available from:
http://www.eecs.ucf.edu/~leavens/ComS541Fall97/hw-pages/paradigms/major.html#imperative
cs.drexel.edu. Introduction of Computer Science Object Oriented Programming [Online].
Available from:
https://www.cs.drexel.edu/~introcs/Fa15/notes/06.1_OOP/Advantages.html?CurrentSlide=3
CA341 David Sinclair Comparative Programming Languages [Online].
Available from:
http://www.computing.dcu.ie/~davids/teaching.shtml