Declaration on Plagiarism

Assignment Submission Form

This form must be filled in and completed by the student(s) submitting an assignment

Name(s): **Joanna Talvo**
Programme: **CASE3**
Module Code: **CA341**
Assignment Title: **Comparing Imperative and Object-Oriented Programming**
Submission Date: **06/11/2020**
Module Coordinator: **David Sinclair**

I/We declare that this material, which I/We now submit for assessment, is entirely my/our 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/our work. I/We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I/We have read and understood the Assignment Regulations. I/We have identified and included the source of all facts, ideas, opinions, and 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. This assignment, or any part of it, has not been previously submitted by me/us or any other person for assessment on this or any other course of study.

I/We have read and understood the referencing guidelines found at
http://www.dcu.ie/info/regulations/plagiarism.shtml , https://www4.dcu.ie/students/az/plagiarism

and/or recommended in the assignment guidelines.

Name(s): **Joanna Talvo** Date: **06/11/2020**

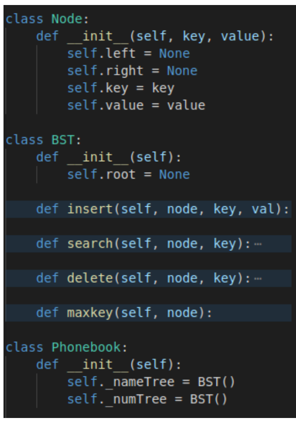
Comparison between Imperative and Object-Oriented Programming

The assignment is to implement a phonebook program using imperative and object-oriented programming languages. The phonebook program must use a binary search tree to store, remove and search entries. Each entry consists of a name, number and address. The program must be able to do a regular search and a reverse search.

Imperative programming is a programming paradigm that tells the computer how to accomplish a task with the use of statements. Functions are coded implicitly in each step required to solve a particular problem. Object-oriented programming is a programming paradigm which is based on the idea of using objects which contain data and code.

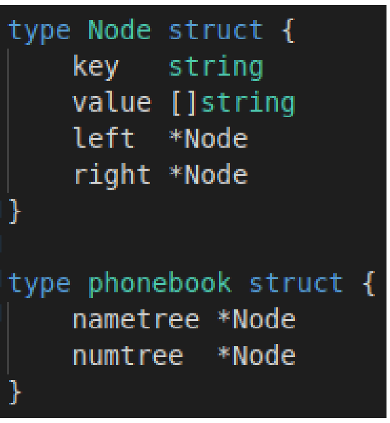
The chosen programming language for the imperative implementation is written in Go while the object-oriented implementation is written in Python. Python is simple and easy to understand. Its syntax alone helped ease the work of creating a phonebook program using a binary search tree. Implementing it with the use of inheritance saved a lot of time and effort. Whereas in Go, the syntax and environment is different which took me a lot of time to implement a phonebook program and Go does not support inheritance.

My approach to this assignment was to use inheritance as it would save me a lot of time and code when implementing 2 BSTs. In my OOP implementation, I have 3 classes; a node class, a BST class and a phonebook class. The phonebook class *(see fig 1)* has 2 objects which are instances of the BST class namely the nametree and numbertree. I chose to take a different approach with my imperative implementation wherein I only have a Node struct and a phonebook struct *(see fig 2)* since inheritance is impossible in Go. Under my phonebook struct are nametree and numbertree fields with *Node as their field type. Both fields are using Node struct which means they can use the fields within the Node struct. This is a similar idea to my BST class where I can use the variables of the Node class. Both implementations operate similarly but in Python, I have a separate BST class which contains all the methods to be used for both objects which I think is better in terms of code readability and code usability.



```python
class Node:
    def __init__(self, key, value):
        self.left = None
        self.right = None
        self.key = key
        self.value = value

class BST:
    def __init__(self):
        self.root = None

    def insert(self, node, key, val):

    def search(self, node, key):⋯

    def delete(self, node, key):⋯

    def maxkey(self, node):

class Phonebook:
    def __init__(self):
        self._nameTree = BST()
        self._numTree = BST()
```

Figure 1

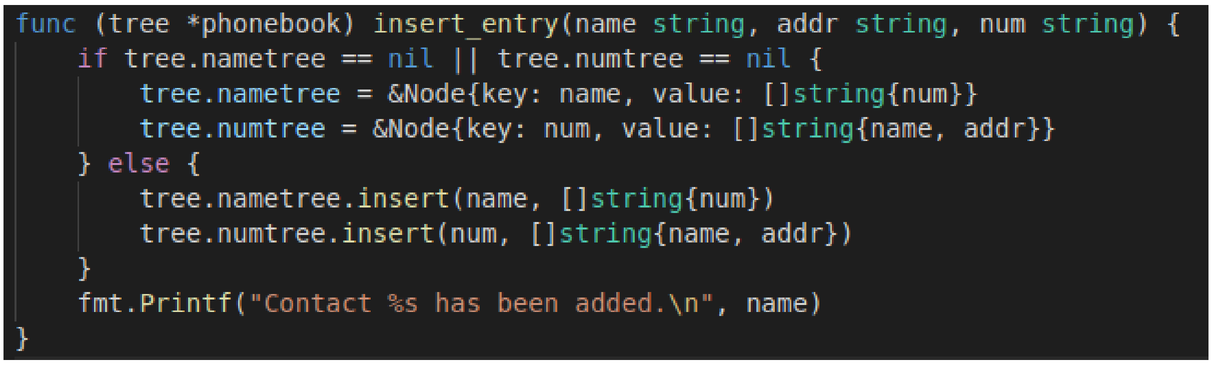


```go
type Node struct {
    key   string
    value []string
    left  *Node
    right *Node
}

type phonebook struct {
    nametree *Node
    numtree  *Node
}
```

Figure 2

In Go, the data fields in my structs have assigned data types. However in Python, I did not set any data types in my parameterized constructor because it knows at the start that my input are strings. However, I think Go in this case is better because it will tell you that you can only use this type of data when using this variable in this struct. You can work your way around that in python but it is better to mention it at the beginning which type of data you want to assign that data to.

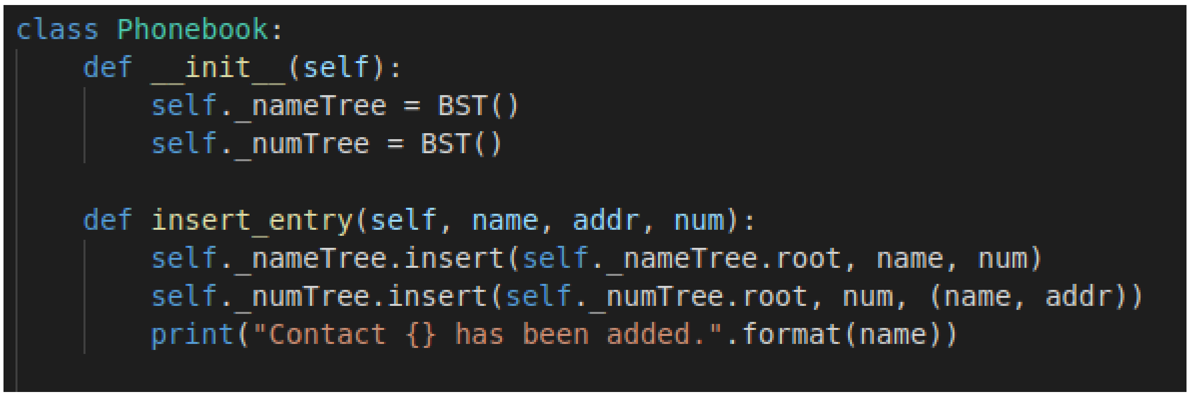
Go does not use ***self*** unlike in Python when defining a method *(see fig 3)*. When defining a method associated with a type, it is used as a named variable. In this case (tree *phonebook), tree is used as a variable within this method. Methods can also modify the values of their pointer receivers to which the receiver points at. In fig 3, I want to give the value of the variables in my phonebook struct. So my nametree and numbertree are assigned values. My arguments also must have argument types in Go which I personally like so my code will return an error in case I use a different data type. Also, methods in Go are independent unlike in python.



```go
func (tree *phonebook) insert_entry(name string, addr string, num string) {
    if tree.nametree == nil || tree.numtree == nil {
        tree.nametree = &Node{key: name, value: []string{num}}
        tree.numtree = &Node{key: num, value: []string{name, addr}}
    } else {
        tree.nametree.insert(name, []string{num})
        tree.numtree.insert(num, []string{name, addr})
    }
    fmt.Printf("Contact %s has been added.\n", name)
}
```

Figure 3 (struct method)

In OOP, when objects are created, they all inherit the methods and variables of the class they are under. Objects are much easier to manipulate and to use. In my numbertree and nametree objects *(see figure 4)*, I can pass arguments and use the variables and methods under the BST class which is very efficient. I don’t have to use any pointers unlike in Go. I also don’t have to tell what type my arguments are in python methods. However it would be better to do so as it could be a problem. In Go, I used a type list for my value since it does not support tuples unlike in Python.

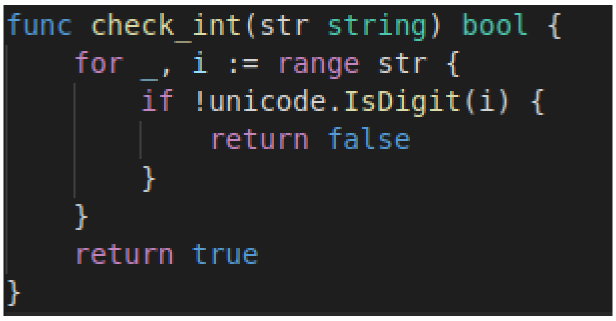


```python
class Phonebook:
    def __init__(self):
        self._nameTree = BST()
        self._numTree = BST()

    def insert_entry(self, name, addr, num):
        self._nameTree.insert(self._nameTree.root, name, num)
        self._numTree.insert(self._numTree.root, num, (name, addr))
        print("Contact {} has been added.".format(name))
```

Figure 4

In imperative style, struct types along with their methods basically serve the same concept of a class in OOP. Struct holds the state and methods provide them behaviour by allowing them to change the state. Code and data are separate. In OOP, classes are blueprints, they contain methods and variables which objects are made from. Code and data are bundled together in an object.

Python isdigit() method which checks if a string is all numerical digits. However in Go, I have to create a method in order to check if all strings are digits (*see figure 5).* Python's built-in methods are practical. It saves lines of code and less errors. IsDigit() in Go can only check one item.



```go
func check_int(str string) bool {
    for _, i := range str {
        if !unicode.IsDigit(i) {
            return false
        }
    }
    return true
}
```

Figure 5

In conclusion, both approaches behave similarly in terms of how they execute and take commands. An imperative approach would be perfect when dealing with smaller programs as it can be very straightforward on what to do. However, it has limitations in terms of larger programs. You have to implement your own methods such as checking if the string is all digit or not. It can take a lot of time and code to be able to implement a program with this approach. An object-oriented approach is much more scalable because it offers a simple and more maintainable approach to programming. OOP is better in larger programs especially it has better concepts that a programmer can use such as inheritance, polymorphism, abstraction and encapsulation. The use of built-in functions are also a bonus when it comes to the OOP approach.