CA341

Imperative and Object-Oriented Programming Comparison

Student Name	Student Number
Kevin Cogan	18421694

Imperative Programming

Imperative programming is a programming paradigm that changes the state of a programme through the execution of commands in a sequential manner. Imperative Programming code focuses on how to do a task. The control flow is very important as the execution of the code in the imperative language is like following a recipe from a cookbook. If not executed in order the state of the output will change, in code, this could be a wrong output.

Briefly, Imperative programming is:

1. Easy to learn for beginners as it follows a logical order like a list instruction.
2. Easy to read as the syntax is basic and functions have to be made to shows each step
3. Code is very similar to the machine language so this gives better control over the execution and memory as well as making it more efficient.
4. Difficult to manage and understand when working on large projects as imperative languages produce a vast amount of code when it explains each step making it difficult for error prevention and detection in code.

Object-Oriented Programming

Object-Oriented Programming is a programming paradigm that makes use of classes and objects to create a structure that allows users to create simple code that is reusable and has the ability to work with code from different developers. In my Object-Oriented Language approach, you can see how I implemented classes in the snippets below.

Object-Oriented Programming has four important features:

1. Abstraction: is used to represent complexity in a simple way that is of use to the user. This can be associated with classes and objects as they hide unnecessary code.
2. Encapsulation: is when private information is held in an object or class and is kept hidden from the outside world. This adds additional security to your code from the outside world.
3. Inheritance: allows classes to inherit attributes from a parent class. This reduces the repetition of code making the code more compact, in turn making it less error-prone
4. Polymorphism: allows a child class to inherit from a parent class and modify that inherited methods to the developer’s needs.

Comparison

I will compare and contrast the Object Oriented Programming language, Python, with an Imperative Programming language, C.

C - Imperative Programming Language Overview

This was developed between 1972 and 1973 and used to run on UNIX operating systems. C became popular due to the availability of C compilers inbuilt on computers. C enabled fast access to the memory with little to no abstraction offering more control for memory management however this is more labour intensive than with a high-level language such as Python. C has low-level features making it close to machine code. As a result, the code is very portable across different operating systems, that has a C compiler, and makes the runtime faster than higher-level languages such as Python. The compiler also helps to increase the efficiency of execution of code as the compiler executes all the code at once wherein python the interpreter executes the code line by line, making execution slower.

Python3 - Object-Oriented Programming Language Overview

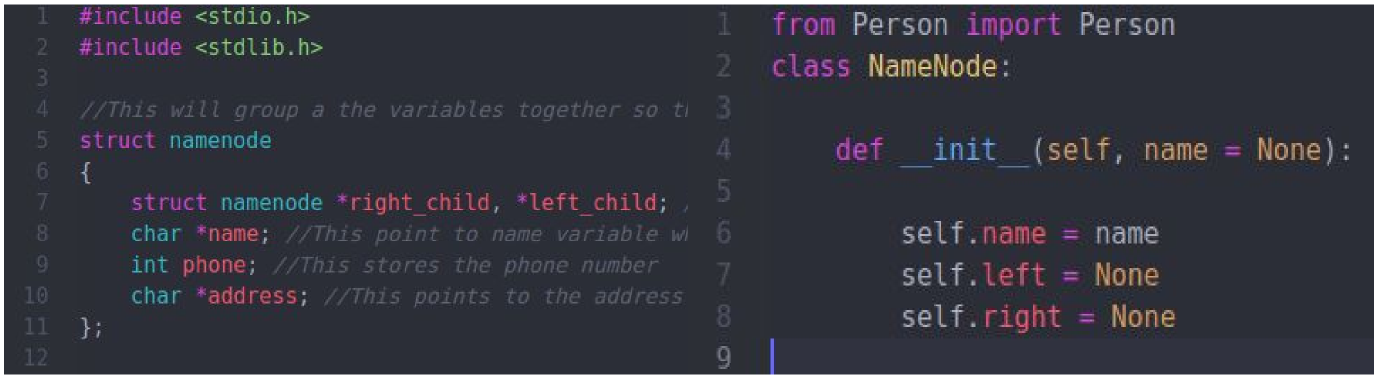
Python was created in the 1980s with the purpose of with a focus on the design of the language's readability and reusability, making it easy to maintain programmes. Python and comes with a large number of libraries that makes the language very powerful to achieve the task. Python helps users by making debugging easier as it tries to avoid segmentation faults with exception handling. This makes python better for debugging as the C language only provides a segmentation fault that can be very difficult to debug in large projects.

C - Statically Type System

On the left, we see that C is a statically typed language due to char and int in front of the variable names. This means that the variable types must be all known before compile time in order to have a compilation without error.

Python - Dynamically Type System

On the right, we see that Python is a dynamically typed language as each variable is not defined by a type. This means that variables are not assigned types or checked for correct type assignments until run-time.



```c
#include <stdio.h>
#include <stdlib.h>

//This will group a the variables together so t
struct namenode
{
    struct namenode *right_child, *left_child; /
    char *name; //This point to name variable w
    int phone; //This stores the phone number
    char *address; //This points to the address
};

```

```python
from Person import Person
class NameNode:

    def __init__(self, name = None):

        self.name = name
        self.left = None
        self.right = None

```

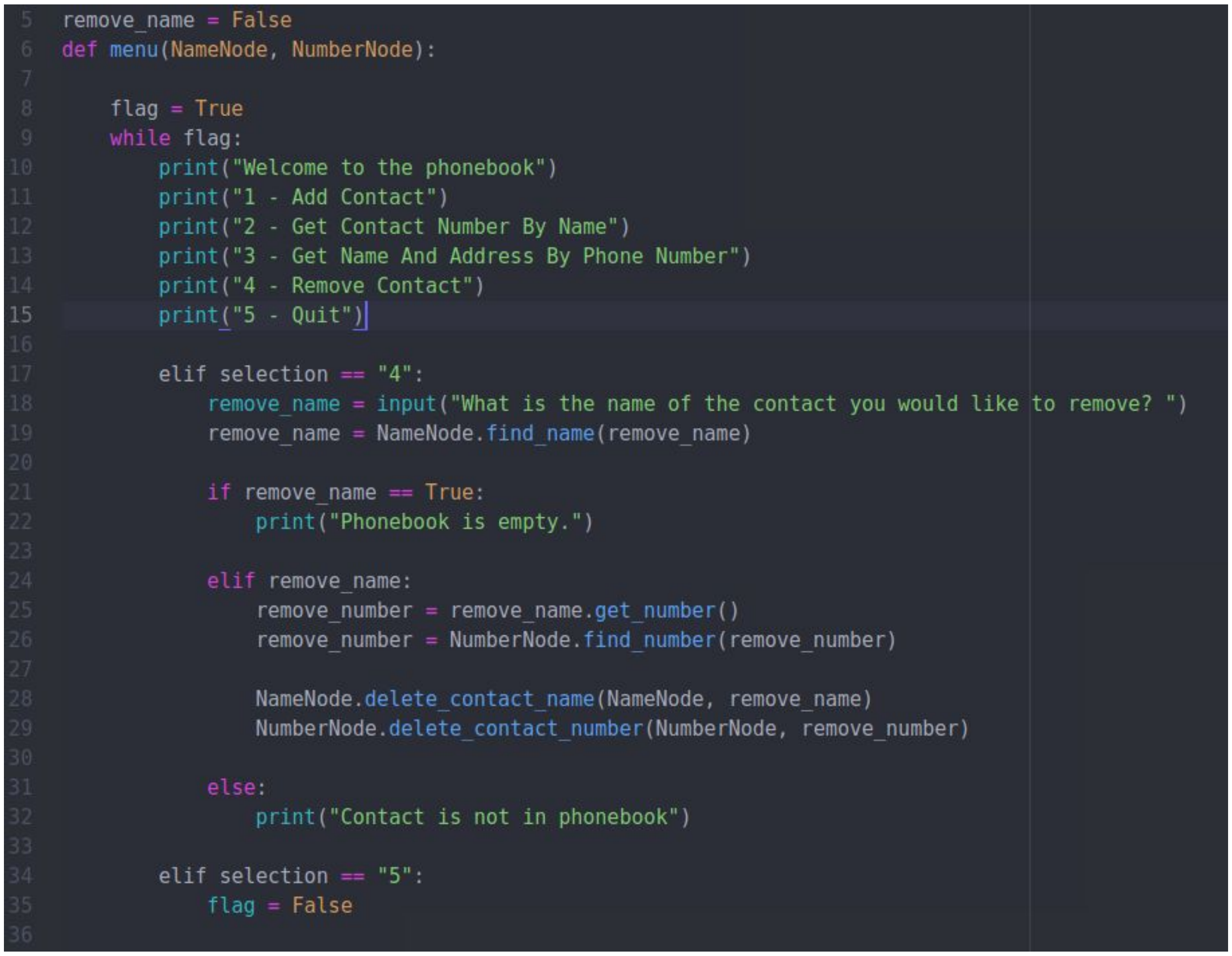
<u>Analysis</u>

1. Statically typed languages already have the types assign so no assignments of types will have to be done which is more efficient than dynamic typing in python as when the types are assigned to values at run-time this gives greater flexibility as they allow the variable type to change however the assignment of type take up greater memory than statically typed language in C.
2. The compiler has already translated the code completely and does not need to interpret the code line by line like the interpreter in python. This makes statically typed languages faster. While dynamically typed languages allow the variable type to change that can often lead to errors.

3. An advantage of this method is during compile time the compiler strictly checks if the code has any type errors before the execution of the code and often will not permit the change of variable types. This offers more security as errors are identified early and can be resolved quickly especially for long programmes. Compared to the dynamically typed language in python errors are not identified at runtime. This is a very late stage to identify errors and can cause an issue with large projects.
4. Dynamically typed languages are cleaner looking than the statically typed language in C

C & Python3 - Static Scoping

Both Python and C both use static scoping as the code blocks are set to just allow variables to be called from their own block. Both languages use a nested blocks structure, however, C only allows a nested block without any occurrence of overlapping.



```python
remove_name = False
def menu(NameNode, NumberNode):

    flag = True
    while flag:
        print("Welcome to the phonebook")
        print("1 - Add Contact")
        print("2 - Get Contact Number By Name")
        print("3 - Get Name And Address By Phone Number")
        print("4 - Remove Contact")
        print("5 - Quit")

        elif selection == "4":
            remove_name = input("What is the name of the contact you would like to remove? ")
            remove_name = NameNode.find_name(remove_name)

            if remove_name == True:
                print("Phonebook is empty.")

            elif remove_name:
                remove_number = remove_name.get_number()
                remove_number = NumberNode.find_number(remove_number)

                NameNode.delete_contact_name(NameNode, remove_name)
                NumberNode.delete_contact_number(NumberNode, remove_number)

            else:
                print("Contact is not in phonebook")

        elif selection == "5":
            flag = False
```

In my implementations of C, I do not make use of static scoping however the principle is very identical for the example above. For my python implementation of static scoping I had to cut out the boolean statements just so we can focus on the topic at hand in this example, please refer to phone_book.py for full code. When the menu () method is called and we select the option 4 - Remove Contact we are asked for a name we would like to remove then we go and use the find_name() method to find the object. If find_name() returns an object it will pass the boolean statement by calling the local variable remove_name() in the menu(). However, if the remove_name input does not return an input because it is not in the binary tree then the boolean statement calls the remove_name global variable that is outside the menu() method. This structure clearly demonstrates static scoping in python but would directly apply to C too.

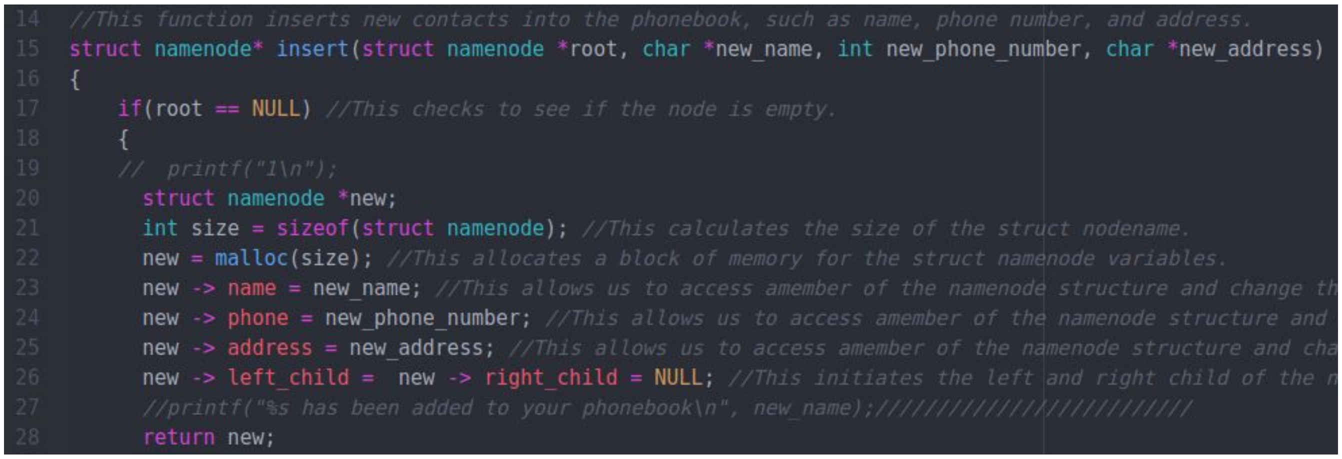
Analysis

1. We are able to easily see how the programme will run just by looking at the code making it quicker and easier for the development of code and debugging for errors such as control flow errors.

C - Memory Management

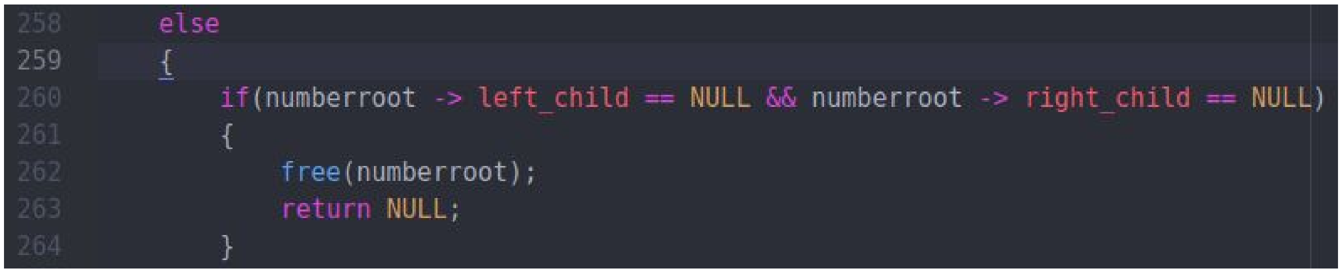
C uses dynamic memory allocation, this enables the programmer to allocate or release memory on demand so no memory is wasted. In C, this is achieved through the inbuilt functions malloc() and free() and pointers. The malloc() creates a memory space for the desired size requested by the user. While in the second snippet of code the free() function is used to release that desired memory location. A pointer to a memory location is denoted by * followed by a variable name this allows us to call the memory location. To access the different memory we use pointer -> to the name of the referenced memory location. However, with more control comes more responsibility which can lead to human errors such as memory getting corrupted by a double-pointer, memory leaks due to not releasing memory when finished or releasing memory too early with using it and dangling pointers.

C - Allocation of memory using malloc()



```c
//This function inserts new contacts into the phonebook, such as name, phone number, and address.
struct namenode* insert(struct namenode *root, char *new_name, int new_phone_number, char *new_address)
{
    if(root == NULL) //This checks to see if the node is empty.
    {
    //  printf("1\n");
      struct namenode *new;
      int size = sizeof(struct namenode); //This calculates the size of the struct nodename.
      new = malloc(size); //This allocates a block of memory for the struct namenode variables.
      new -> name = new_name; //This allows us to access amember of the namenode structure and change th
      new -> phone = new_phone_number; //This allows us to access amember of the namenode structure and
      new -> address = new_address; //This allows us to access amember of the namenode structure and cha
      new -> left_child =  new -> right_child = NULL; //This initiates the left and right child of the n
      //printf("%s has been added to your phonebook\n", new_name);/////////////////////////
      return new;
```

C - Freeing of memory using free()

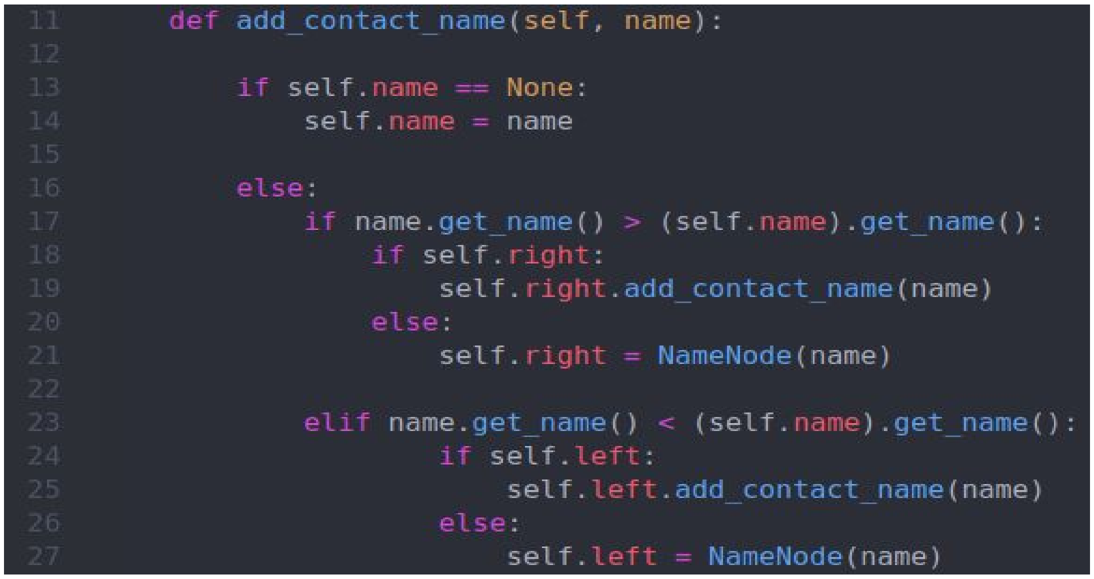


```c
    else
    {
        if(numberroot -> left_child == NULL && numberroot -> right_child == NULL)
        {
            free(numberroot);
            return NULL;
        }
```

Python3 - Memory Management

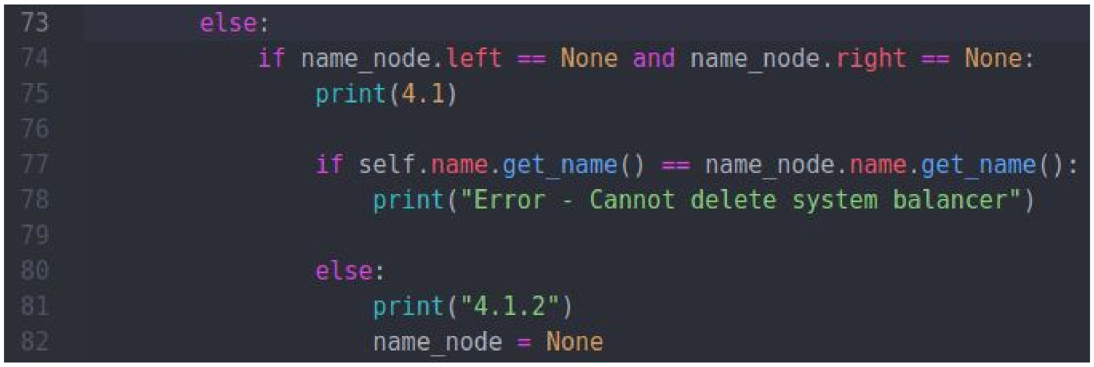
Python3 uses an automatic memory management system called a garbage collector. This automatically frees the memory of objects when the reference counter for an object is equal to zero. The reference counter increases and decreases depending on the assignment of data to the memory location. The garbage collector quietly frees up safe from a memory location at a frequent but random time. In the example above to allocate memory, you simply set a variable name equal to a value. This increments the reference counter, Then to free the memory location we first need to assign the variable None that will put the reference counter to zero so the next time the garbage collector is activated in the background it will free up the undesired memory.

Python3 - Allocation of memory



```python
    def add_contact_name(self, name):

        if self.name == None:
            self.name = name

        else:
            if name.get_name() > (self.name).get_name():
                if self.right:
                    self.right.add_contact_name(name)
                else:
                    self.right = NameNode(name)

            elif name.get_name() < (self.name).get_name():
                    if self.left:
                        self.left.add_contact_name(name)
                    else:
                        self.left = NameNode(name)
```

Python3 - Freeing of memory

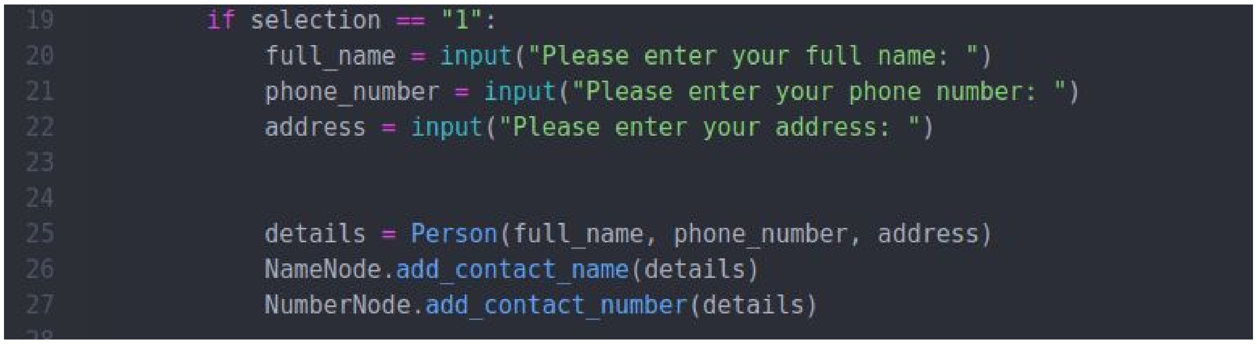

```
        else:
            if name_node.left == None and name_node.right == None:
                print(4.1)

                if self.name.get_name() == name_node.name.get_name():
                    print("Error - Cannot delete system balancer")

                else:
                    print("4.1.2")
                    name_node = None
```

Analysis

1. Garbage Collectors in Python uses additional memory to store a variable so it can keep track of each value compared to the use of dynamic memory allocation in C.
2. The Garbage Collector in Python alleviates the risk of data corruption in the Dynamic memory allocation as the computer operates the freeing and allocation of space which eliminates the human error factor.
3. The Garbage Collector in Python runs quietly in the background at random times this can be an issue for real-time applications that need memory to be free instantly. C's Dynamic memory allocation would be best suited as it allows fast memory management that can be customised to the needs of the application.
4. Comparing assigning memory locations to variables it is evident that the python implementation is clean and requires less knowledge to achieve as we do not have to make use of a pointer or allocate memory for strings.

Python3 - Reference Parameter Passing Mechanism

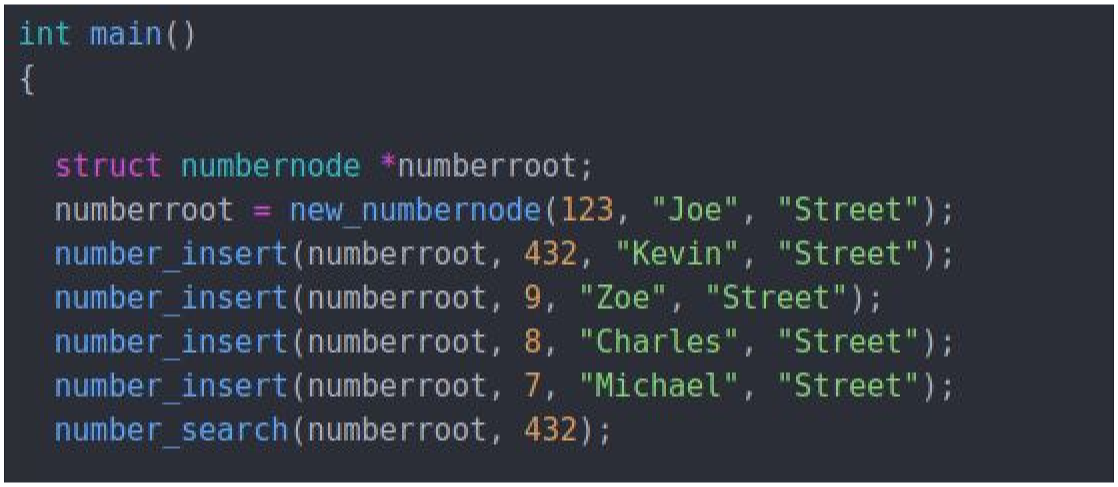
Python3 makes use of parameter passing by reference this means that it has the ability to pass in an object and modify it. Any change to the object passed in will directly modify the original referenced memory location (Copy-Out Parameter).



```python
        if selection == "1":
            full_name = input("Please enter your full name: ")
            phone_number = input("Please enter your phone number: ")
            address = input("Please enter your address: ")


            details = Person(full_name, phone_number, address)
            NameNode.add_contact_name(details)
            NumberNode.add_contact_number(details)
```

C - Copy Parameter Passing Mechanism

C uses the copy parameter passing mechanism to pass arguments into a function. This is achieved by passing a copy/local variable into the function. Any modification to the copy passed into the function will not have any effect on the origami object (Copy-In Copy-Out Parameter).



```c
int main()
{

  struct numbernode *numberroot;
  numberroot = new_numbernode(123, "Joe", "Street");
  number_insert(numberroot, 432, "Kevin", "Street");
  number_insert(numberroot, 9, "Zoe", "Street");
  number_insert(numberroot, 8, "Charles", "Street");
  number_insert(numberroot, 7, "Michael", "Street");
  number_search(numberroot, 432);
```

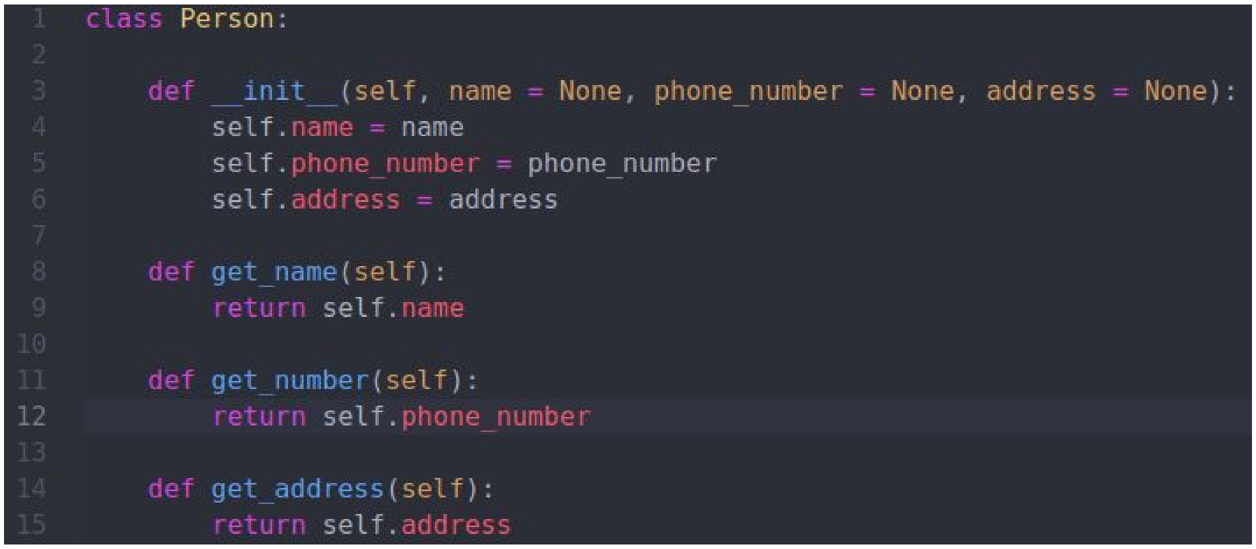
<u>Analysis</u>

1. C's copy parameter passing mechanism requires more memory as we have to create a copy of the memory location to pass the argument into a function. Compared to in Python3 where we use the reference parameter passing mechanism that allows us to pass the reference object to be modified directly.

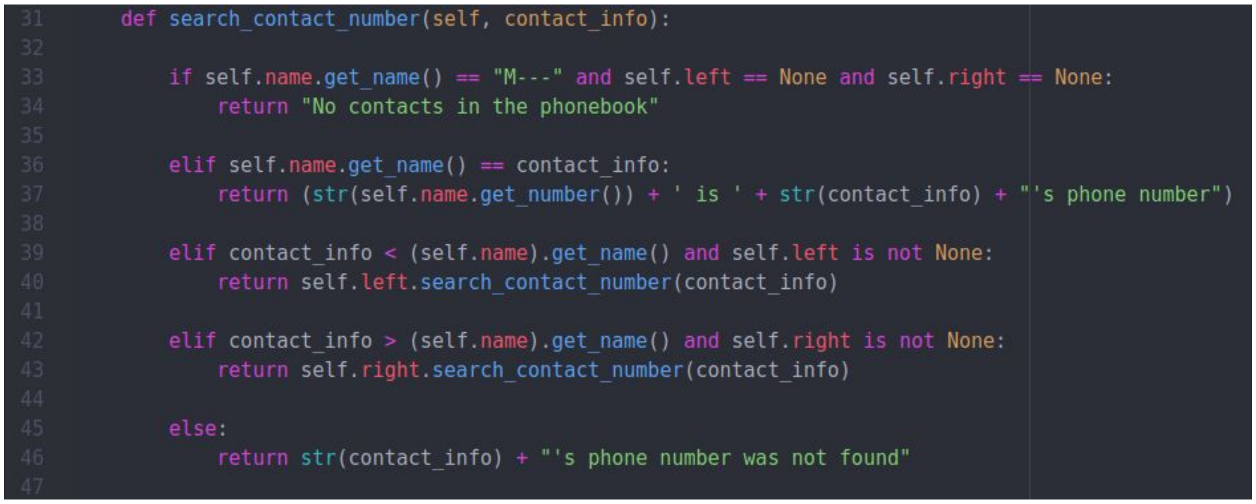
2. Python has to pass a value by copy parameter passing mechanism and reference parameter passing mechanism this makes the python programming more powerful as it has more functionality.
3. Python's use of reference parameter passing mechanism enables the development of complex data structures with ease as there is less code to deal with such as making copies of memory locations.

Python3 - Abstraction

Abstraction in Python3 can be clearly seen in the two code snippets below. The Person Class creates numerous helper functions all allow us to get certain information on a Person object in the binary search tree. In the second snippet below we can see how the helper function creates abstraction, as all the programmer has to do is put the helper functions name in without any knowledge of what the code looks like and we will get the requested information that we need.



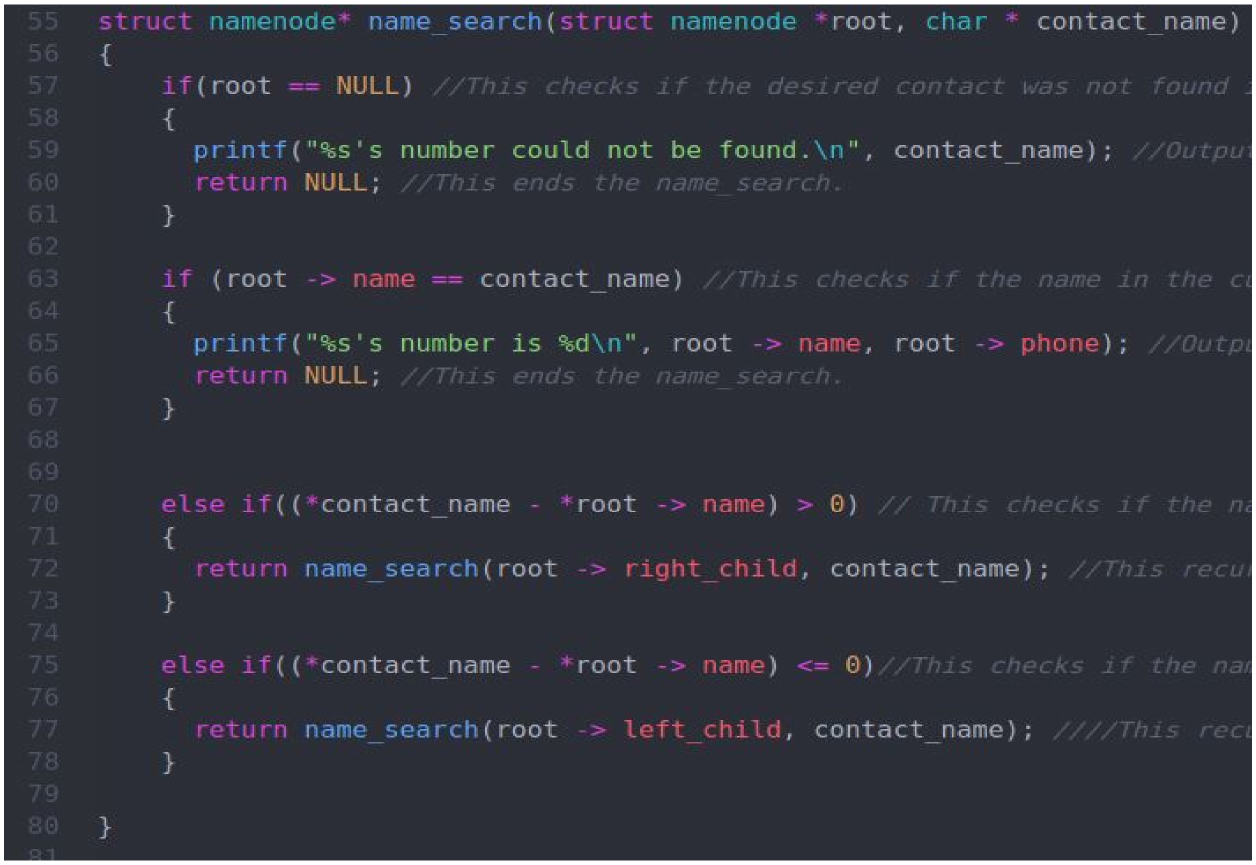
```python
class Person:

    def __init__(self, name = None, phone_number = None, address = None):
        self.name = name
        self.phone_number = phone_number
        self.address = address

    def get_name(self):
        return self.name

    def get_number(self):
        return self.phone_number

    def get_address(self):
        return self.address
```



```python
    def search_contact_number(self, contact_info):

        if self.name.get_name() == "M---" and self.left == None and self.right == None:
            return "No contacts in the phonebook"

        elif self.name.get_name() == contact_info:
            return (str(self.name.get_number()) + ' is ' + str(contact_info) + "'s phone number")

        elif contact_info < (self.name).get_name() and self.left is not None:
            return self.left.search_contact_number(contact_info)

        elif contact_info > (self.name).get_name() and self.right is not None:
            return self.right.search_contact_number(contact_info)

        else:
            return str(contact_info) + "'s phone number was not found"

```

C - No Abstraction

Abstraction is not seen in the code implementation as imperative programming languages show each instruction step by step. In the code below you can see that each step is explained with no abstraction.



```c
struct namenode* name_search(struct namenode *root, char * contact_name)
{
    if(root == NULL) //This checks if the desired contact was not found i
    {
      printf("%s's number could not be found.\n", contact_name); //Outpu
      return NULL; //This ends the name_search.
    }

    if (root -> name == contact_name) //This checks if the name in the cu
    {
      printf("%s's number is %d\n", root -> name, root -> phone); //Outp
      return NULL; //This ends the name_search.
    }


    else if((*contact_name - *root -> name) > 0) // This checks if the na
    {
      return name_search(root -> right_child, contact_name); //This recu
    }

    else if((*contact_name - *root -> name) <= 0)//This checks if the nar
    {
      return name_search(root -> left_child, contact_name); ////This rec
    }

}
```

Analysis

1. Abstraction in python makes the code more complex to fully understand how everything works in contrast to C. The code clearly explains each step and makes it easy to understand for the programmers.
2. Abstraction reduces visual complexity as the is less code to process this allows developers to debug software easily as there are fewer lines of code and everything is segmented nicely while in C everything is visually dense.
3. Abstraction encourages the Object-Oriented Programming design of reuse ability of code which is clearly evident in the python snippet above while with C the code has to be written out every time you would like to run code.
4. Abstraction increases security as the workings of the code is hidden from the world.

Conclusion

After analysing the Python3 (Object-Oriented Language) and C (Imperative Language) it is evident that both languages have a lot to offer for different tasks. I believe that Python3 can be slow in performance overall compared to C however it is more built around humans needs and difficulties as it does not bombard us with line after line of code. Instead, it offers us tools such as abstraction and classes to help it have much more functionality for large project development and make the life of a programmer much easier with its debugging tools and memory management. On the other hand, C is very much a powerful language in the hands of an expert as C is very similar to machine code, that makes it very difficult to do complex tasks with. However, if this challenge is overcome it enables programmers to customise their code to the needs of the user such as with memory management the C programming language can have total control over its memory helping the development of real-time applications.

References

https://www.computing.dcu.ie/~davids/courses/CA341/CA341.html - Dr.David Sinclair from DCU, I used the topics headings in Dr.David Sinclair's notes as a guide for what topics I should cover in this report.

https://www.youtube.com/watch?v=x8l1AXD1maM - This video was used to get a better understanding of searching and adding on a binary tree.

https://www.codesdope.com/blog/article/binary-search-tree-in-c/ - As a beginner to the C language I used this webpage to get a better understanding of the structure of how a binary tree can be created in C such as the creation of a binary tree.