Specification Section

1 Fundamentals of Programming

1.1 Programming

3.1.1.1 A data type determines what sort of datum is being stored and how it will be handled by the program. There are several built in data types common to many programming languages:

- Integer: Any positive or negative whole number including zero.

  -2, -1, 0, 1, 2
- Real/Float: Any number with a decimal/fractional part.

  -1, $-\frac{1}{2}$, 0, 0.543543, 1
- Boolean: True or False.
- Character: An individual character (alphanumerical or symbol).

  ”a”, ”1”, ”&”
- String: A sequence of characters.

  ”Hello World”
- Date/Time: A date or time.

  28.05.2016 12:39:40:056
- Records: A collection of items which be of different data types which are related.

  [{Name: ”Nathan”, age: 17} ,{Name: ”Serena”, age: 17}]
- Arrays: A collection of items of the same data type.

  [”Ryan”, ”Helen”, ”Luke”,”Giorgia”]

A user defined data type is a data type that is made of built-in data types (A data type that is provided within the programming language being used). In python you can write your own data types by writing type methods in C, or simply creating a class which does what you want.

3.1.1.2 Within a program there is usually a combination of the following statement types:

- Variable declaration: The process of defining a variable in terms of its data type and identifier (variable name).
- Constant declaration: The process of defining a constant in terms of its data type and identifier (constant name).
- Assignment: Giving a value to a variable or constant.
- Iteration: The principle of repeating processes.
- Selection: The principle of choosing what action to take based on certain criteria.
- Subroutine (AKA Procedure): A named block of code designed to carry out a specific task.
- Function: A subroutine which returns a value.

In python, all of the variables and constants are dynamically typed, meaning you don’t have to worry about declaring a data type, as it is taken care of by the programming language.
Within imperative programs the combining of the principles sequencing, iteration and selection are basic to all of them.
Definite iteration is when a process repeats a set amount of time, for example:

```
1 for x in range(10):
2   print("hi")
```

This would print `hi` 10 times. Indefinite iteration is a process that repeats until a certain condition is met. This can be done in two ways, *Method 1* with the condition at the beginning, and *Method 2* with the condition at the end. The implications of these two methods is that *Method 2* forces the loop to be done once, whereas the loop in method one may never be done. This can be shown using an example using both methods:

Method 1

```
x = 5
while x < 5:
  print(x)
```

Method 2

```
x = 5
while True:
  print(x)
  if not (x<5):
    break
```

Method one results in nothing being printed as `x` starts as `5` so `x<5` `False`, so the loop never runs, so `x` isn't printed. Method two results in `5` being printed because `True` is always `True`, so the while loop runs printing `x` which is equal to `5`, the if statement then resolves to be true (`x<5` is `False` so `not x<5` is true) so the while loop `breaks`.
Nesting is placing one set of instructions within another set of instructions, the most common use of nesting is nested selection (`If...Elif...End`) and nested iteration(A for loop within a for loop).
Within a program, the use of meaningful identifier names is encouraged due to the following reasons:

- It's easier to debug (correct) code.
- Easier for others to understand when working on a large project.
- Easier to update the code.

3.1.1.3 The basic arithmetic operations are:

- Addition `a + b`

  `2 + 3 = 5`
- Subtraction `a - b`

  `2 - 3 = -1`
- Multiplication `a * b`

  `2 * 3 = 6`
- Real/Float Division `a / b`

  `2 / 3 = 0.6666666666`
- Integer Division: The result is the truncated integer of the result. `a // b`

  `17 // 3 = 5`
- Modulus `a % b`

  `17 % 3 = 0`
- Exponentiation `a ** b`

  `2**3 = 8`
- rounding `round(a)`

  `round(0.6666666666)=1`, `round(1.4352534234) = 1`
- truncation `Math.trunc(a)`

  `Math.trunc(0.6666666666)=0`, `Math.trunc(1.4352534234) = 1`

3.1.1.4 Relational operations are expressions that compare two values. Some common Relational Operations are:

- equal to (`==`)
- not equal to (`!=`)
- less than (`<`)
- greater than (`>`)
- less than or equal to (`<=`)
- greater than or equal to (`>=`)

3.1.1.5 Boolean operations are expressions that return the result `True` or `False`. Some common Boolean Operations are:

- AND: Returns `True` if both inputs are true.
- OR: Returns `True` if either of its inputs are true.
- NOT: Negates (inverses) the input, `True` → `False`, `False` → `True`
- XOR: Returns `True` if either of its inputs are true but not if both are true.

3.1.1.6 A constant is an item of data whose value does not change whereas a variable is an item of data whose value could change while the program is being run. Named constants are useful because you can easily use them throughout the program, and don't have to worry about the initial value, also you can easily change it by changing the assignment and declaration of the constant.

3.1.1.7 There are several ways to manipulate and convert strings from one data type to another. Some examples of string handling functions are:

- Length: Returns the number of characters within a given string
  `len("Hello World")=11`.
- Position: Returns the position of any character or string within another string
  `"Hello World".index("World")=6`.
- Substring: Returns a string contained within another string.
  `"Hello World"[0:5]="Hello"`.
- Concatenation: returns the result of Adding two strings together
  `"Hello"+"World"="HelloWorld"`.
- Character→Character Codes: Converts a character to a character code (a binary representation if a particular letter, number of special character).
  `ord("A")=65`
- Character Codes→Character: Converts a character code to a character.
  `chr(65)="A"`
- String Conversion Operations
  - String to Integer:
    `int("2")=2`
  - String to Real/Float:
    `float("2.432")=2.432`
  - String to Date/Time:
    ```
    datetime.datetime.strptime('5 May 2016', '%d %b %Y')=
                              datetime.datetime(2016, 5, 5, 0, 0)
    ```
  - Integer to String:
    `str(2)="2"`
  - Float to String:
    `str(2.432)=2.432`
  - Date/Time to String:
    `time.strftime("%d/%m/%Y",time.localtime())="28/05/2016"`

3.1.1.8 In python random number generator functions are all contained within the module `random` and therefore requires us to import it using `import random`. There are several functions within this module, but the three most important functions are:

- `random.random()`: produces a random real number between 0 and 1
- `random.randint(a,b)`: produces a random integer between `a` and `b`.
- `random.sample(population, k)`: Chooses `k` unique random elements from a `population`.

3.1.1.9 Exception Handling is the process of dealing with events that cause the current subroutine/ procedure to stop. In general this is done by:

1. An error is thrown causing the current subroutine to stop.
2. The current state of the subroutine is saved.
3. The exception handling (or catch) block is executed to take care of the error.

4. the normal subroutine can continue from where it left off.

In python, exception handling is done by using `try` and `except`. Here is a relatively simple example:

```
Age = input("Please Input Your Age: ")

try:
  Age = int(Age)
except:
  print("Age is not an Integer please try again.")
else:
  print("Your age is %i"%(Age))
```

What this code does is it first asks the user to input their age. We then go into the exception handling part where the code tries to make the input an integer. If an error occurs then the program prints `"Age is not an Integer please try again."`, if no errors occur, then the
3.1.1.10 program prints out the age inputted at the start of the program. A subroutine is self-contained and it carries out one or more related processes, subroutines must be given unique identifiers or names, which means that once they have been written they can be called using their name at any time while the program is being run. Subroutines can be written to handle events (something that happens during runtime).
The benefits of using subroutines are as follows:

- They can be called at any time.
- They allow for an easy overview of the program.
- Can use a top-down approach to develop a project.
- Easier to debug as each subroutine is self-contained.
- Large projects can be developed by multiple programmers

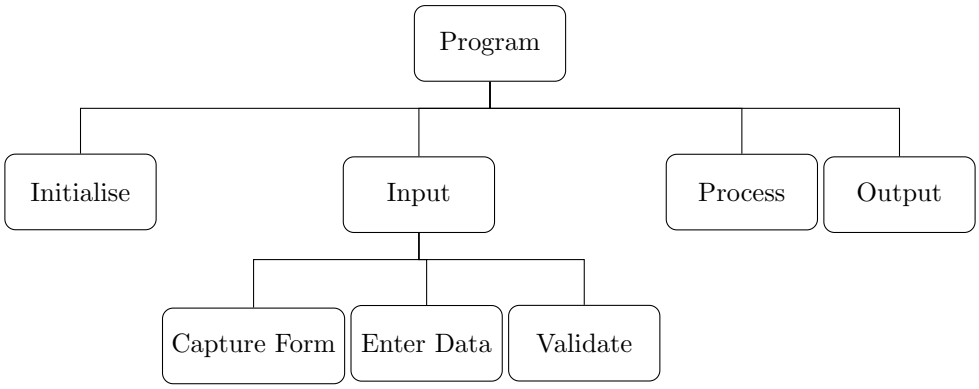
3.1.1.11 A Subroutine often has parameters and Argument. Parameters are pieces of data that represents data to be passed into a subroutine and an argument is a piece of data that is passed into the subroutine. For example if you defined a subroutine `LoadGame(Filename, Board)` Filename and Board are parameters, later when it is called as `LoadGame(TRAININGGAME, Board)` the variables TRAINNINGGAME and Board are the arguments. To pass the arguments into the subroutine a block interface is used, which is code that describes the data being passed into the subroutine.
3.1.1.12 To define a subroutine/ function in python we use the keyword `def` and to add arguments brackets are used after the subroutine name. so if we wanted to define a function named Add_Contact which has the parameters Name, and Address, we would write `def Add_Contact(Name, Address):` for the function to return a value you simply use `return` followed by the data you want the function to return to the calling routine.
3.1.1.13 Within a subroutine (in python) any variable that isn't declared as a global variable, is considered a local variable, meaning that it only exists within the subroutine, and once the subroutine has finished, the variable would no longer exist, so they cannot be accessed outside of the subroutine. There are three main benefits to this which are:

- Can't inadvertently change the value being stored elsewhere in the program.
- Use the same identifier in several places and have them be consider different variables.
- Free up memory as each time a local variable is finished with it is removed from memory.

3.1.1.14 The difference between a local and global variable is that a local variable has a limited existance within a subroutine or function in which it was declared whereas a global variable can be used anywhere in the program.

1.2 Procedural-Oriented Programming

3.1.2.1 Hierarchy or Structure Charts use a top-down approach to explain how a program is put together, meaning it starts from the program name and breaks the problem down into smaller pieces. A Structure Chart differs from a Hierarchy Chart as a Structure Chart shows how data flows through a system, whereas a Hierarchy Chart does not. An example of a Hierarchy Chart is as follows:



A flowchart is a diagram using standard symbols that describes a process or system. A system flowchart is a diagram that shows individual processes within a system. It often possible to create just one flowchart that shows the entire system, but this is not always a good idea as modern programs can be very large and putting every process on to one flowchart might make it too complex to be of any real use. This can be fixed by having multiple flowcharts for the multiple systems.
Pseudo-code is a method of writing code that does not require knowledge of a particular programming language without having to worry about syntax or constructs. The only true rule of Pseudo-code is that it has to be internally consistent, for example if you write `print` in one place and then write `output`, this is considered bad practice and also makes it harder to convert it to a programming language later.
Pseudo-code can be used at many levels of detail meaning it is up to the programmer to decide what level of detail is appropriate to the project they are planning to do. One of the major benefits of using Pseudo-code is it allows the programmer to see how his code may eventually be laid out.
Naming Conventions is the process of giving meaningful names to subroutines, functions, variables and other user-defined features in a program. Before coding, a list of all the variables, including their data type and scope (Global or local) should be made. A similar procedure should also be carried out for all functions and subroutines to be featured within the program.
When writing the actual code, you should try and make your program as programmer-friendly as possible with the use of code layout and comments, examples of this would be:

- Comments to show the purpose of an algorithm.
- Comments to show the purpose of each line.
- Sensible variable names.
- Indenting the contents of loops and subroutines.

After the code is initially written, debugging will often have to occur. This can be done using a dry run and a trace table. A dry run is the process of stepping through each line of code to see what will happen before a program is run, a trace table is a method of recording the result of each step that takes place when dry running code.

2 Fundamentals of Data Structure

2.1 Data Structures and Abstract Data Types

3.2.1.1 A data structure is any method used to store data in an organised and accessible format, they normally contain data that are related, different data structures allow for different data manipulations which means different data structures are use for different types of applications. For example an array may be useful to store a list of names whereas a textfile may be used to store
3.2.1.2 information for a database. An array is a set of related data items stored under a single identifier, they can have one or more dimensions, all elements are often of the same type(homogeneous). An array most commonly has either one dimension(which can be useful to represent vectors) which can be visualised using a list, or two dimensions (which is useful for representing a matrix) which can be visualised using a two-dimensional table. In python, instead of Arrays we use lists, which have a few minute differences to standard arrays (python lists are heterogeneous(They can

store dat of different types)) but can be used in the same way as arrays. Some uses of lists are as follows:

```
Studentname = ["Derrick","Gill","Jamal","Lois"]
Studentname[1]
'Gill'

ArrayAdd=[[0,1,2],[1,2,3],[2,3,4]]
ArrayAdd[1][2]
3
```

3.2.1.3 Files are used to store many different types of data meaning that many different file types are needed to store all of these different types of data. Many file types are portable meaning that they can be used on many different platforms, the two most common portable file types when programming are text files (which is a file that contains human readable characters) and binary files (which stores data as 1s and 0s). One line on a text file may be referred to as a record, and the different items of data stored within the record are called the fields.
All files have an internal structure which allows them to store data efficiently, there are two common structures that are used to store data These are:
Tab-delimited text (txt) file:

```
Sara  Phillips  sphillips0@google.co.jp Female  117.135.192.97
Laura Harvey  lharvey1@utexas.edu Female  62.114.62.185
Eugene  Wells ewells2@weibo.com Male  119.176.45.229
Helen Jordan  hjordan3@geocities.jp Female  81.49.64.62
Shirley Weaver  sweaver4@pbs.org  Female  218.20.41.34
```

Comma seperate variable (csv):

```
Sara,Phillips,sphillips0@google.co.jp,Female,117.135.192.97
Laura,Harvey,lharvey1@utexas.edu,Female,62.114.62.185
Eugene,Wells,ewells2@weibo.com,Male,119.176.45.229
Helen,Jordan,hjordan3@geocities.jp,Female,81.49.64.62
Shirley,Weaver,sweaver4@pbs.org,Female,218.20.41.34
```

To read and write to csv, we can use the python module csv:

```
import csv
file=open("Contacts.csv","a+",newline='')
Reader = csv.reader(file) # This reads the contents of the file
Writer = csv.writer(file) # This creates an object which allows us to write to the
    file.
file.write("\n")
Writer.writerow(['Joe', 'Shmuck', 'JShmuck3D@hotmail.com', 'Male','162.148.10.205'
    ])
file.close()
```

Binary files contain binary codes and usually contain some header information that describes what these represent, binary files are not easily readable by a human, but can quickly be interpreted by a program. For example, the PNG image file is a binary file, can be used in a range of applications and requires less memory than some other image formats. Many program files (executables) are binary files so they can be used on other platforms. The two main actions you might want to perform on binary files are to read and write data from and to it.

3 Systematic Approach to Problem Solving

3.1 Aspects of Software Development