2.2.1 Programming Techniques

Revision sheet

1 Programming Constructs

There are three fundamental programming constructs which are used throughout all of computing.

1.1 Sequence

This is where the program runs lines of code sequentially. For example, shown below is code which will print `Hello World!` to the screen 5 times.

```
print("Hello World!")
print("Hello World!")
print("Hello World!")
print("Hello World!")
print("Hello World!")
```

1.2 Iteration

This is where the program will iterate around the same lines of code for a set number of times or until a condition is met. It is commonly used for input validation, where an input is continually taken from the user until the user enters a valid input. Both of the examples below will print `Hello World!` 5 times to the screen

1.2.1 Condition Controlled

```
count=0
while count < 5
    print("Hello World!")
    count = count + 1
endwhile
```

1.2.2 Count controlled

```
for count=0 to 4
    print("Hello World!")
next count
```

1.3 Branching

Branching is used to control the flow of a program. It is commonly used to react to a user input or to a value which has been generated within the code.

There are a number of different ways branching can be implemented, each with their own advantages. In the examples below, if the user enters `1` then the program will output `Hello World!`; if the user enters `2` then the program will output `Goodbye World!`; and if the user enters something else then the program will output `Cheesecake`.

1.3.1 If

```
uInput = input("Enter here: ")
if(uInput == 1) then
    print("Hello World!")
else if (uInput == 2) then
    print("Goodbye World!")
else
    print("Cheesecake")
endif
```

1.3.2 Switch

```
switch uInput:
    case 1:
        print("Hello World!")
    case 2:
        print("Goodbye World!")
    default:
        print("Cheesecake")
endswitch
```

2 Recursion

Recursion is the process of defining a problem (or a solution to a problem) in terms of itself. In computer science, this usually means a function or procedure which calls itself. A recursive function will need to have the three following properties: a base case; it must call itself from within; and the stopping condition must be reached after a finite number of calls. The base case is the control case which controls how many runs of the subroutine can be called.

2.1 Why Recurse?

Recursion tends to be more how humans think as it is fundamentally iterating around a problem until a solution is found. However, recursion has a number of disadvantages: if the recursive function is faulty then it may lead to stack overflow; it can be difficult to debug as the program may be many levels deep before errors arise; and it makes heavy use of memory.

2.2 Call Stack

The call stack is the part of memory which holds the variables from the functions when another is called. The size of this can be a problem in recursion especially as every time the program calls the function, the current functions variables are loaded onto the stack - which can get quite big if there are lots of recursive calls and lots of variables in the function.

2.3 Recursion Or Iteration

Both recursion and iteration are valid approaches however different problems will be able to be solved better by one or the other.

3 Variables

There are two main types of variable: *global* and *local*. Each of them operate on different scopes.

3.1 Global Variables

These are declared in the main program and can be used anywhere in the program, including in any subroutine.

3.2 Local Variables

These are declared in a subroutine and are only accessible within that subroutine. They cannot be accessed outside that subroutine. If a local variable is needed to be accessed outside the subroutine then it should be returned to where the calling program.

4 Modular Programs

When writing programs, it is possible to write small blocks of code which can be used again and again - these are called subroutines. The modular approach to programming is particularly applicable to big projects or projects where more than one person is working on it.

4.1 Types Of Subroutines

There are two different types of subroutine.

4.1.1 Function

A function returns a value to the calling program.

4.1.2 Procedure

A procedure does not return a value to the calling program.

4.2 Parameters

When writing subroutines, it might be needed to pass data (variables) into the subroutine. This is achieved using parameters. There are two different ways in which data can be passed to a subroutine.

4.2.1 By Reference

By reference means that the memory address of the variable is passed to the subroutine. This means that any modifications to the variable will be seen in the main program too.

4.2.2 By Value

By value means that a copy of the variable is passed to the subroutine therefore any changes made to it are not reflected in the main program. This is the default for many programming languages. If a variable gets modified and needs to be passed back to the main program then it has to be returned, meaning the subroutine has to be a function.

4.3 Advantages Of Modular Programming

There are a number of advantages to developing a program using a modular approach:

- Subroutines are generally small enough to be understood as a unit of code with a defined purpose. This makes it easy to understand, debug and maintain.
- Subroutines can be tested independently before combining together to make the entire program.

- Once a well-written and well-tested subroutine has been developed, it can be used many times in the program with confidence that it will work.
- It allows multiple people to be working on one project simultaneously as the only thing they have to agree is parameters and what will link to what, not variable names or the exact inner workings of each subroutine.
- Larger projects become easier to monitor and control.

5 IDE

An *Integrated Development Environment* will contain many tools and options which programmers can use to make developing software easier.

5.1 Features Of An IDE

An IDE provides tools which help programmers: write; compile; test; and debug their code.

5.1.1 Writing Code

IDEs provide a text editor within which programmers can enter their code. This text editor will also have line numbers for ease of referencing specific parts of code.

5.1.2 Compiling And Running Code

IDEs will often have built in compiler options, where you can instruct the compiler or interpreter to run with a click of a button. Some IDEs will come with compilers/ interpreters pre-installed. After compilation, the IDE will display the error report or during interpretation, it will display the errors as it comes across them. IDEs also have a number of other tools which can be used to pinpoint errors:

- Breakpoints - this will cause the program to stop when it reaches a certain line so you can see if the program reaches that line or breaks before it.
- Variable Watch - watch the contents of a variable as the program runs, this helps programmers to understand if the variable is being changed at the wrong time.
- Step Through - run the program line by line to observe what is happening after each line.

5.1.3 Testing

It is very important to thoroughly test computer programs during development, ensuring all inputs have been tested with a variety of data so that when it is released to the user, they can't break it. There are three types of data which can be used when testing: normal (data within the range that you would expect and of the data type that you would expect); boundary (data at the ends of the expected range); and erroneous (data outside the expected range). Another test method which can be used is to dry-run a program which means you trace through the program using a trace table.

6 Object Oriented Techniques

See *1.2.4 Types Of Programming Language* for details on how Object Oriented Techniques work.