GCST

2.2 PROGRAMMING FUNDAMENTALS

CONCISE NOTES

GCSE OCR

code is written in OCR Exam Reference Language

2.2.1 PROGRAMMING FUNDAMENTALS

Variable - stores a single piece of data. It is a label for an allocated area of memory. The value of a variable can be changed during the execution of the program.

Constant - is also a label for an allocated area of memory. Unlike a variable, the value of a constant cannot change during the execution of the program.

Variables and constants are given an **identifier** (or name). Their identifiers can be almost anything but must: not contain spaces, not start with a number and not be particular words reserved for use in the programming language.

Variables and constants are **assigned** values using the = operator. Variables can be assigned new values throughout a program, which overwrites the previous value. Constants can only be assigned a value once.

Programming Constructs

- Building blocks programmers use to write code. There are 3 main types:

Sequence - is the execution of statements one after the other, in order. A program runs from top to bottom and each instruction completes fully before the next one is executed.

Selection - is the construct used to make decisions in a program. For example, if statements and switch/case statements. Eg;

```
if answer == "Yes" then
   print("Correct")
elseif answer == "No" then
   print("Wrong")
else
   print("Error")
endif

switch day :
   case "Sat":
     print("Saturday")
   case "Sun":
     print("Sunday")
   default:
     print("Weekday")
endswitch
```

Iteration - is the construct used to repeat sections of code. Iteration is commonly called looping. Two types of iteration, **counted-controlled** and **condition controlled**.

Count-controlled iteration repeats code a defined number of times. **FOR** loops can be used to implement count-controlled iteration.

```
for y = 1 to 20 step 2
   print y
next y
```
^this will print the odd numbers from 1 to 20 inclusive

Condition-controlled iteration checks a condition each time around the loop and decides whether to repeat the code again or continue. **WHILE loops** and **DO UNTIL** loops can be used to implement condition-controlled iteration.

```
while answer != "Correct"
   answer = input("New answer")
endwhile

do
   answer = input("New answer")
until answer == "Correct"
```

Will loop until the user inputs the string "Correct". Check condition is carried out before entering the loop.

Arithmetic operators can be used to carry out mathematical operations on numeric values

Operator	Name	Example
+	Addition	1+2 = 3
-	Subtraction	5-4 = 1
*	Multiplication	4 * 2 = 8
/	Division	10/5 = 2
MOD	Modulus – returns the remainder after division	5 MOD 2 = 1
DIV	Quotient – returns the whole number after division	5 DIV 2 = 2
^	Exponent	2^5 = 32

Data can be input from the user and output back to the user. For example:

```
name = input("Enter your name")
print(name)
```

Comparison Operators are used to evaluate expressions to a Boolean True or False outcome

Operator	Name
==	Equal to
!=	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

Boolean operators allow multiple conditions to be evaluated.

- AND operator requires both conditions to be True for the overall condition tc be True.
- OR operator requires one or the other (or both) of the conditions to be True for the overall condition to be True.
- NOT operator reverses the True or False outcome from a comparison.

2.2.2 DATA TYPES

Integers - are whole numbers, positive or negative.

Real number - is used for numbers, positive or negative, that have (or may have) a decimal or fractional part. For example, 18.779 is a real number. (**floats** are similar but used when precision is required)

Boolean - only store True or False values

Character - is a single item from the character set. E.g., '@' is a character

String - stores a collection of characters. For example, 'Hello world' is a string.

Casting - is the conversion of one data type to another. Not all data can be cast to another data type. E.g. "Hello World" to an integer

OCR Reference Language	Converts to
str()	String
int()	Integer
real()	Real
bool()	Boolean
float()	Float

2.2.3 ADDITIONAL PROGRAMMING TECHNIQUES

String slicing - means to extract individual characters from strings
String concatenation - means to join strings together `(print("hello" + " world")`

Keyword	Use	Example
`.length`	count how many characters are contained in a string.	`s = "hello"` s.length would be 5
`.substring(x, y)`	Extract characters from a string; x is the starting, y is the number of characters required	`s.substring(2,2)` would be ll
`.left(x)`	Extract characters from the left of a string where x is the number of characters required	`s.left(2)` would be he
`.right(x)`	Extract characters from the right of a string where x is the number of characters required	`s.right(2)` would be lo
`.upper`	To convert a string to uppercase	`s.upper` would be HELLO
`.lower`	To convert a string to lowercase	`s.lower` would stay the same
`ASC()`	To find the ASCII Value of a character	`ASC('A')` would be 65
`CHR()`	To find the character that relates to the ASCII Value given	`CHR(97)` would give 'a'

Files can be written to and read from by a program. They must be **opened** before they can be used.

Keyword	Use
`open()`	`contents = open('text.txt")` opens text.txt into the variable contents
`.readLine()`	`data = contents.readLine()` reads a single line of data from contents
`.writeLine()`	`contents.writeLine("New Line Added")` Writes a single line of data at the end of the file
`.endOfFile()`	`contents.endOfFile()` Return TRUE if the last line of contents has been reached otherwise returns FALSE
`.close()`	`contents.close()` Closes the file
`newFile()`	`newFile("myText.txt")` Creates a new text file called "myText". The file would then need to be opened using the above command for Open.

Example

```
randomFile = open("random.txt")
while NOT randomFile.EndOfFile()
   data = randomFile.readLine()
   print(data)
endwhile
randomFile.close()
```

In the example above, the code prints out all of the data contained in the "random.txt" file.

- opens the text file "random.txt" using the identifier randomFile
- sets up a WHILE loop that runs until the end of the file is reached
- reads one line of data from randomFile into a variable called data
- prints out the contents of the variable data
- this loop repeats for each line in the text file, and prints out the contents of " random.txt" one by one
- once the end of the file is reached the loop stops
- randomFile is closed.

A **record** is a data structure that allows multiple data items to be stored, using field names to identify each item of data. Data is organised using field names and stored in a table.

SQL (Structured Query Language) is a language used to access data stored in a database.

- **SELECT** identifies the fields to return from the database; (* means all fields)
- **FROM** identifies which table(s) the data will be returned from
- **WHERE** is an optional command that allows the programmer to include criteria, with only matching records being returned

For example: Below is a table called "Students" showing three records

FirstName	Surname	YearGroup	Email
Bradley	Cable	9	bcable@lol.com
Jamie	Pegg	10	jpegg@lol.com
Alex	Mcqueen	9	amcqueen@lol.com

```
SELECT FirstName, Email
FROM Student
WHERE YearGroup = 9 or YearGroup = 10
```

Would show

FirstName	Email
Bradley	bcable@lol.com
Jamie	jpegg@lol.com
Alex	amcqueen@lol.com

One-dimensional arrays allow a programmer to store multiple items of data under a single identifier. All items in an array are **indexed** (the first item is at index position 0 and so on). Items in the array must also have the same data type.

```
array colours = ["Blue", "Pink", "Green", "Yellow", "Red"]
```

`colours[2] = "Purple"` - this will change the colour green to purple.
`colours.length` will return 5

To print out all the elements in the array we do:

```
for i = 0 to colours.length -  1
      print(colours[i])
next i
```

Two dimensional arrays are **arrays of arrays**. And follow the same principles as 1D arrays.

```
array scores = [
                [1, 10, 2],
                [3, 9, 1],
                 [0, 2, 5]
                ]
```

where `scores[1,1] = 9`

To access each value, two index numbers are required separated by a comma, for example **`scores[0,2]`**. Any exam question using a table for this will tell you whether you access the array as [row, column] or [column, row]. If it doesn't tell you, just be consistent with whatever way you pick.

To find the total of all the elements in the 2D array we do:

```
total = 0
for I = 0 to scores.length - 1
      for j = 0 to scores[i].length - 1
      total += int(scores[I,j])
      next j
next i
print(total)
```

Random numbers can be generated using the `random()` keyword.
`print(random(1, 100)) // prints a random integer between 1 and 100`

When programs grow in size, they can become hard to manage. Ideally, larger programs should be broken down into **subprograms** (sometimes called **subroutines**).
The **advantages** of using subprograms are:

- They reduce the overall size of the program as code does not need to be repeated in multiple places.
- They make the code much easier to maintain as it is easier to read and understand the purpose of each subprogram.
- They reduce development and testing time as code is much easier to write and debug.
- They allow reuse of code between programs, especially where pre-written and pre-tested subprograms can be used.

Procedures are a type of subprogram that **do not return a value** to the main program.

Functions are a type of subprogram that **return a single value** to the main program. This value then can be stored or used in the main program which is not possible with a procedure.

Example

```
procedure add(x, y)
   total = x + y
   print(total)
endprocedure

function addition(x, y)
   total = x + y
   return total
endfunction
```

The **`return`** keyword must always be used when defining a function, in order to state what value the function will send back to the main program.
Once defined, procedures and functions can be called at any point in the main program and parameters passed into them in brackets:

```
add(1,2) // call the procedure add using the numbers 1 and 2.

x = addition(2,3) // call the function addition using the numbers 2
and 3 and stores the value in the variable x
```

- The variables declared in a procedure or function are called **local variables** (e.g., the variable `total`). Local variables are defined in a subprogram and are accessible only in that subprogram in which they are defined.
- **Global Variables** are defined at the start of the program and exist throughout the whole program and in all subprograms. These variables allow data to be shared between subprograms.