J277 - 2.2 Programming fundamentals

2.2 – Programming fundamentals

Sub topic	Guidance
2.2.1 Programming fundamentals	
☐ The use of variables, constants, operators, inputs, outputs and assignments	**Required** ✓ Practical use of the techniques in a high-level language within the classroom ✓ Understanding of each technique ✓ Recognise and use the following operators:
☐ The use of the three basic programming constructs used to control the flow of a program: ○ Sequence ○ Selection ○ Iteration (count- and condition-controlled loops)	
☐ The common arithmetic operators	
☐ The common Boolean operators AND, OR and NOT	

Comparison operators		Arithmetic operators	
==	Equal to	+	Addition
!=	Not equal to	–	Subtraction
<	Less than	*	Multiplication
<=	Less than or equal to	/	Division
>	Greater than	MOD	Modulus
>=	Greater than or equal to	DIV	Quotient
		^	Exponentiation (to the power)

J277 - 2.2 Programming fundamentals

Variables, constants, inputs, outputs and assignments

Variable: A value stored in memory that can change whilst the program is running.

Constant: A value that does not change whilst the program is running and is assigned when the program is designed.

Assignment: Giving a variable or constant a value.

Example annotated program: Python

```
#Program to calculate the VAT of an ex-VAT item.

vat_rate = 0.2

cost = input("Enter the ex-VAT price of the item: £")
cost = float(cost)

vat = round(cost * vat_rate,2)

total = cost + vat

print("Ex-VAT price was: £","{:.2f}".format(cost))
print("VAT is: £","{:.2f}".format(vat))
print("Total price is: £","{:.2f}".format(total))
```

constant

input

variable

assignment

output

Example annotated program: OCR reference language

```
//Program to calculate the VAT of an ex-VAT item.

const vat_rate = 0.2

cost = input("Enter the ex-VAT price of the item: £")
cost = float(cost)

vat = round(cost * vat_rate,2)

total = cost + vat

print("Ex-VAT price was: £", (cost))
print("VAT is: £", (vat))
print("Total price is: £", (total))
```

constant

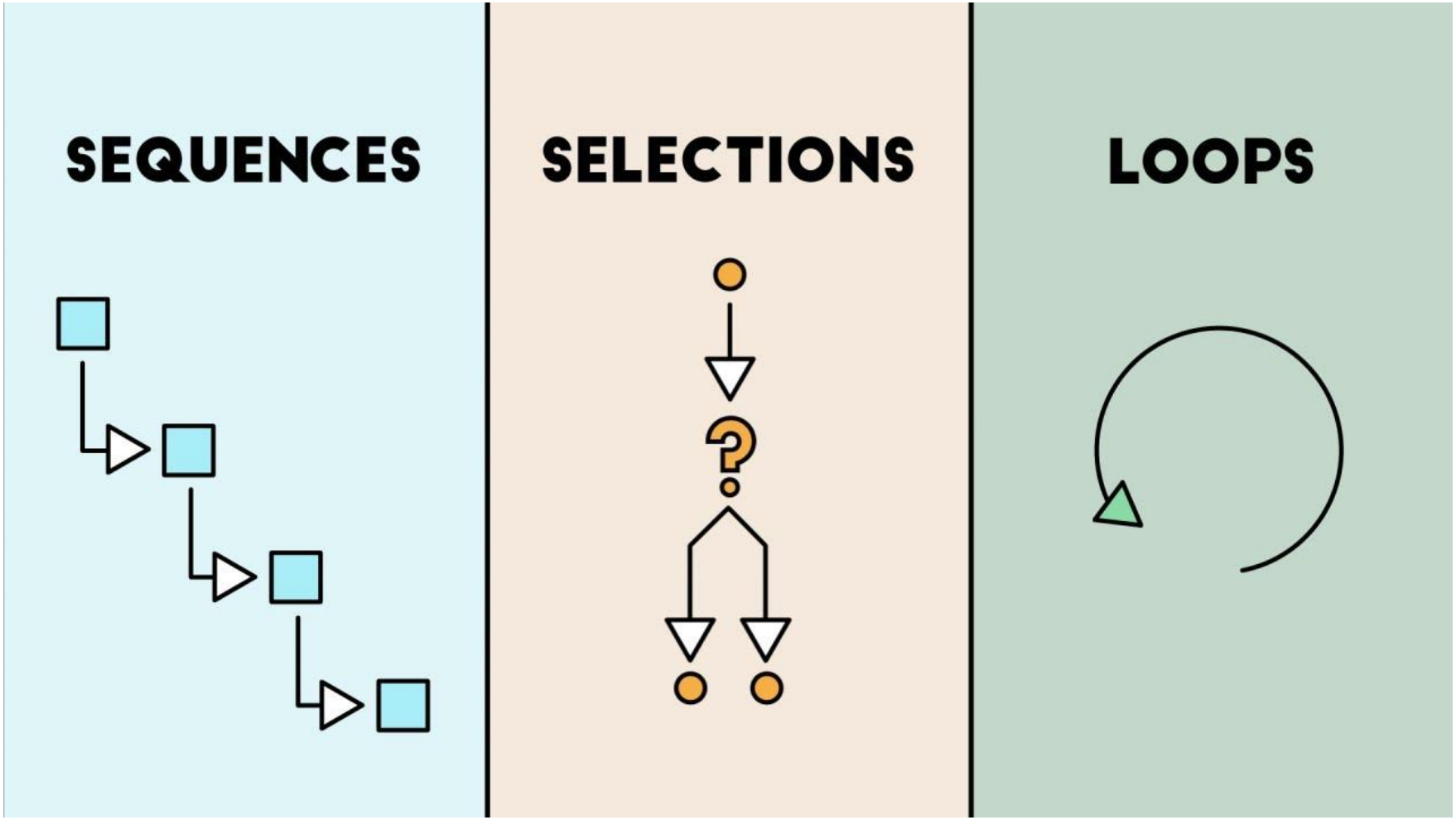
input

variable

assignment

output

The use of the three basic programming constructs: sequence, selection and iteration



J277 - 2.2 Programming fundamentals

The use of the three basic programming constructs: sequence, selection and iteration

Sequence

```
x = 3
MyVariable = 34
New_Variable = x + MyVariable
```

Iteration with a count controlled loop

```
for i in range(10):
  print("Mrs Rollings")
next counter
```

Selection

```
if x > 90 then
        output "a*"
else if x > 80 then
   output "a"
else if x > 70 then
        output "b"
else
        output "fail"
end if
```

Iteration with a condition controlled loop

```
answer = "y"
while answer == "y":
   print("Stay very still")
   answer = input("Is the monster friendly? y/n")
print('Run away!')
end while
```

The use of the three basic programming constructs: sequence, selection and iteration

Selection

```
select case x
    case >90
           output "a*"
    case >80
           output "a"
    case >70
           output "b"
…
    case else
           output "fail"
end select
```

Not supported by all languages, this construct is an alternative to using else if or elif commands.

Iteration with a condition controlled loop

```
do
    name = input("Enter a chosen name: ")
until name.length < 4 or name.length > 12
```

Not supported by all languages, this iteration is different because it will execute the code inside the loop at least once.

J277 - 2.2 Programming fundamentals

Arithmetic, comparison and Boolean operators

Logical operation	Operator	Example
Equivalence	==	`if x == 5`
Less than	<	`if x < 5`
Less than or equal to	<=	`if x <= 5`
Greater than	>	`if x > 5`
Greater than or equal to	>=	`if x >= 5`
Does not equal	<>	`if x <> 5`

Mathematical operation	Operator	Example
Addition	+	`x = x + 5`
Subtraction	-	`x = x - 5`
Multiplication	*	`x = x * 5`
Division	/	`x = x / 5`
Integer division	`DIV` (finds the whole number after the division)	`x = x DIV 5` If `x` is 21, then the result of this is that `x` is 4
Remainder	`MOD` (finds the remainder after the modulus division)	`x = x MOD 5` If `x` is 21, then the result of this is that `x` is 1

Boolean operation	Operator	Example
Both statements must be true for the argument as a whole to be true.	`AND`	`if x>=5 AND x <=20` Returns `TRUE` if `x` is any number between 5 and 20.
Only one of the statements needs be true for the argument as a whole to be true.	`OR`	`if x==2 OR x==5` Returns `TRUE` if `x` is either 2 or 5.
The opposite of the argument is true.	`NOT`	`if NOT(x==10)` Returns `TRUE` if `x` is not 10.
The argument is false if both statements are true. The argument is false if both statements are false. Otherwise the argument is true.	`XOR`	`if x<=10 XOR y<=10` Returns `TRUE` if one of `x` or `y` is greater than 10 and the other is not.

J277 - 2.2 Programming fundamentals

2.2.2 Data types	
☐ The use of data types: ○ Integer ○ Real ○ Boolean ○ Character and string ○ Casting	**Required** ✓ Practical use of the data types in a high-level language within the classroom ✓ Ability to choose suitable data types for data in a given scenario ✓ Understand that data types may be temporarily changed through casting, and where this may be useful

Data Type	Definition	Examples
Integer	A whole number.	1, 5, -63, 247
Float	A number with a decimal point	1.5, 3.14159, -0.47, 2.0
String	A block of text (designated with ‘ marks in Python). Any numbers contained within a string are treated as text and cannot be used in calculations	‘Python’, ‘ICT’, ‘Hilbre High’, ‘3’
Array (called List in Python)	A list of data stored in a structured way. Represented with [ ] in Python and values separated with a comma.	[‘Python’, ‘programming’, ‘is’, ‘awesome’]
Comment	A comment in the code which is ignored by the computer. Useful for leaving instructions	# This line creates a variable called myBox

J277 - 2.2 Programming fundamentals

2.2.3 Additional programming techniques	
☐ The use of basic string manipulation ☐ The use of basic file handling operations: ○ Open ○ Read ○ Write ○ Close ☐ The use of records to store data ☐ The use of SQL to search for data ☐ The use of arrays (or equivalent) when solving problems, including both one-dimensional and two-dimensional arrays ☐ How to use sub programs (functions and procedures) to produce structured code ☐ Random number generation	**Required** ✓ Practical use of the additional programming techniques in a high-level language within the classroom ✓ Ability to manipulate strings, including: ▪ Concatenation ▪ Slicing ✓ Arrays as fixed length static structures ✓ The use of functions ✓ The use of procedures ✓ Where to use functions and procedures effectively ✓ SQL commands: ▪ SELECT ▪ FROM ▪ WHERE

Casting operations

Casting:

A process that converts a variable's data type into another data type.

Examples

cost = int(cost)

cost = str(cost)

```
# Casting from int to float
x = 5
y = float(x)
print(y)  # Output: 5.0

# Casting from float to int
a = 3.14
b = int(a)
print(b)  # Output: 3

# Casting from int to string
num = 10
text = str(num)
print(text)  # Output: "10"

# Casting from string to int
string_num = "20"
int_num = int(string_num)
print(int_num)  # Output: 20
```

Basic string manipulation

Assuming that a string called name is assigned the value: “Hilbre12345”:

To return the **length** of a string:	l = len(name)	OCR Reference Language: l = name.length
To return the string in **uppercase**:	upperC = name.upper()	OCR Reference Language: up = name.upper
To return the string in **lowercase**:	lowerC = name.lower()	OCR Reference Language: low = name.upper
To return the 5th character of the string:	char = name[4:5]	OCR Reference Language: char = name.substring(4,1)

Basic file handling operations: Opening a file

Creating a file

To create a file, you will use the following code:

```
myFile = open("newFile.txt","wt")
```

The code above creates a file named **newFile.txt** (if it does not already exist).
It is opened in "**wt**" mode, which is short for write. This tells the program we are going to write to a file.
The file is opened into a variable we create called **myFile**. The variable can have any name you like providing it is meaningful.

Basic file handling operations: Reading a file

Reading to a file

To return a string containing all characters in the file, you can use file.read().

```
myFile = open("instructions.txt","r")
print(myFile.read())
myFile.close()
```

prepares it for reading using the “r”.

The **second** line prints out the contents of the file using the **variable name**.

Basic file handling operations: Writing to a file

Writing to a file

```
myFile = open("newFile.txt","wt")
myFile.write("I can add text to this file.")
myFile.close()
```

Notice how the variable MyFile is always called.

The code is continued from the previous slide.

The **second line** writes the sentence "I can add text to this file." to our text file. This uses a method called write.

The **final line** closes the file. It is really important that you close the file after you have finished working with it.

J277 - 2.2 Programming fundamentals

2.2.3 Additional programming techniques	
☐ The use of basic string manipulation ☐ The use of basic file handling operations: ○ Open ○ Read ○ Write ○ Close ☐ The use of records to store data ☐ The use of SQL to search for data ☐ The use of arrays (or equivalent) when solving problems, including both one-dimensional and two-dimensional arrays ☐ How to use sub programs (functions and procedures) to produce structured code ☐ Random number generation	**Required** ✓ Practical use of the additional programming techniques in a high-level language within the classroom ✓ Ability to manipulate strings, including: ▪ Concatenation ▪ Slicing ✓ Arrays as fixed length static structures ✓ The use of functions ✓ The use of procedures ✓ Where to use functions and procedures effectively ✓ SQL commands: ▪ SELECT ▪ FROM ▪ WHERE

Use of records

Record: A data structure that stores related values of different data types.

Field: An element of a record used to store one piece of data.

Most languages let you define your own data structures in the form of records.

An example:

Create a fixed record structure by giving a data type and name for each filed.

```
Record ticket

      string filmName
      int seatNumber
      real price
endrecord
```

Create records by giving values for each field

```
Ticket1 = ticket ("Data Force", 16, 7.50)
```

```
Print (ticket1.seatNumber)
```

Use the variable and field name to access the values

```
16
```

J277 - 2.2 Programming fundamentals

2.2.3 Additional programming techniques	
☐ The use of basic string manipulation ☐ The use of basic file handling operations: ○ Open ○ Read ○ Write ○ Close ☐ The use of records to store data ☐ The use of SQL to search for data ☐ The use of arrays (or equivalent) when solving problems, including both one-dimensional and two-dimensional arrays ☐ How to use sub programs (functions and procedures) to produce structured code ☐ Random number generation	**Required** ✓ Practical use of the additional programming techniques in a high-level language within the classroom ✓ Ability to manipulate strings, including: ▪ Concatenation ▪ Slicing ✓ Arrays as fixed length static structures ✓ The use of functions ✓ The use of procedures ✓ Where to use functions and procedures effectively ✓ SQL commands: ▪ SELECT ▪ FROM ▪ WHERE

J277 - 2.2 Programming fundamentals

The use of records to store data & SQL to search for data

Structured query language (SQL):

SQL is a programming language used for interrogating a database.

Person ID	Title	Forename	Surname	Email address
1001	Mr	Alan	Turing	aturing@bitesize.com
1002	Mrs	Ada	Lovelace	alovelace@gcsecompsci.com
1003	Miss	Grace	Hopper	ghopper@bitesizemail.co.uk
1004	Mr	George	Boole	gboole@bbcbitesize.com

Retrieving data

Data can be retrieved using the **commands** `SELECT`, `FROM` and `WHERE`, for example:

```
SELECT * FROM "personnel" WHERE "Title" = "Mr"
```

Note - `*` stands for wildcard, which means all records. This would retrieve the following data:

1001 Mr Alan Turing aturing@bitesize.com

1004 Mr George Boole gboole@bbcbitesize.com

The `LIKE` command can be used to find matches for an incomplete word, for example:

```
SELECT * FROM "personnel" WHERE "email address" LIKE "%com"
```

This would retrieve:

1001 Mr Alan Turing aturing@bitesize.com

1002 Mrs Ada Lovelace alovelace@gcsecompsci.com

1004 Mr George Boole gboole@bbcbitesize.com

Note - `%com` is also a wildcard which will return any value that contains “com”.

The use of records to store data & SQL to search for data

Structured query language (SQL):

SQL is a programming language used for interrogating a database.

Person ID	Title	Forename	Surname	Email address
1001	Mr	Alan	Turing	aturing@bitesize.com
1002	Mrs	Ada	Lovelace	alovelace@gcsecompsci.com
1003	Miss	Grace	Hopper	ghopper@bitesizemail.co.uk
1004	Mr	George	Boole	gboole@bbcbitesize.com

Boolean operators `AND` and `OR` can also be used to retrieve data.

```
SELECT * FROM "personnel" WHERE "Surname" = "Turing" OR "Hopper"
```

This would retrieve:

1001 Mr Alan Turing aturing@bitesize.com

1003 Miss Grace Hopper ghopper@bitesizemail.co.uk

J277 - 2.2 Programming fundamentals

2.2.3 Additional programming techniques	
☐ The use of basic string manipulation ☐ The use of basic file handling operations: ○ Open ○ Read ○ Write ○ Close ☐ The use of records to store data ☐ The use of SQL to search for data ☐ The use of arrays (or equivalent) when solving problems, including both one-dimensional and two-dimensional arrays ☐ How to use sub programs (functions and procedures) to produce structured code ☐ Random number generation	**Required** ✓ Practical use of the additional programming techniques in a high-level language within the classroom ✓ Ability to manipulate strings, including: ▪ Concatenation ▪ Slicing ✓ Arrays as fixed length static structures ✓ The use of functions ✓ The use of procedures ✓ Where to use functions and procedures effectively ✓ SQL commands: ▪ SELECT ▪ FROM ▪ WHERE

Arrays

An array is a series of memory locations – or 'boxes' – each of which holds a single item of data, but with each box sharing the same name. All data in an array must be of the same **data type**.

For example, imagine that a score table in a game needs to record ten scores. One way to do this is to have a variable for each score:

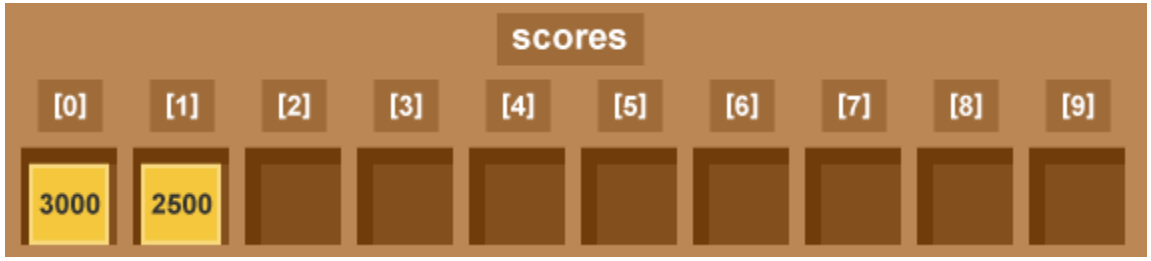
```
score_0
score_1
score_2
score_3
score_4
score_5
score_6
score_7
score_8
score_9
```

Instead of having ten variables, each holding a score, there could be one array that holds all the related data:

```
score(9)
```

All the data in an array must be of the **same data type**. In this example all the data items are integers.

The **size** of an array is declared when the program is **written**.



2D Arrays

A 2D array is also known as a matrix (a table of rows and columns).
To create a 2D array of integers, take a look at the following example:

In this example, we create a 2D array (a list of lists) called **matrix**. Each inner list represents a row, and the outer list represents the entire 2D array.

```
# Creating a 2D array
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

# Accessing elements in the 2D array
print(matrix[0][0])  # Output: 1
print(matrix[1][2])  # Output: 6
print(matrix[2][1])  # Output: 8

# Modifying elements in the 2D array
matrix[1][0] = 10
matrix[2][2] = 15
```

Similarly, **matrix[1][2]** refers to the element at the second row and third column, which is 6.

We can access individual elements of the 2D array using indices. For example, **matrix[0][0]** refers to the element at the first row and first column, which is 1.

We can modify elements of the 2D array by assigning new values to specific indices. In the example, we change **matrix[1][0]** to 10 and **matrix[2][2]** to 15.

J277 - 2.2 Programming fundamentals

2.2.3 Additional programming techniques	
☐ The use of basic string manipulation ☐ The use of basic file handling operations: ○ Open ○ Read ○ Write ○ Close ☐ The use of records to store data ☐ The use of SQL to search for data ☐ The use of arrays (or equivalent) when solving problems, including both one-dimensional and two-dimensional arrays ☐ How to use sub programs (functions and procedures) to produce structured code ☐ Random number generation	**Required** ✓ Practical use of the additional programming techniques in a high-level language within the classroom ✓ Ability to manipulate strings, including: ▪ Concatenation ▪ Slicing ✓ Arrays as fixed length static structures ✓ The use of functions ✓ The use of procedures ✓ Where to use functions and procedures effectively ✓ SQL commands: ▪ SELECT ▪ FROM ▪ WHERE

Sub programs (functions and procedures) to produce structured code

A function:	A sub program that **DOES** return a value.
The purpose of a function:	To create a reusable program component. Returning a value to be used in the program.

Function Example

- Functions are similar to procedures but always return a value to the main program.

Example

```
Function Hello(FirstName As String, SecondName
As String) As String
      Return "Hello" + " " + FirstName +" "+
SecondName
End Function
Name=Hello("Jonathan", "Weir")
Print(Name)
```

Hello Jonathan Weir

Sub programs (functions and procedures) to produce structured code

A procedure:	A sub program that **DOESN'T** return a value.
The purpose of a procedure:	To produce structured code that is easier to read and debug.

Procedure example

- Procedures are sets of instructions stored under one name (identifier).

```
Example
Procedure name()
      name=Input("What is your name?")
      print("Hello " + name)
End Procedure
```

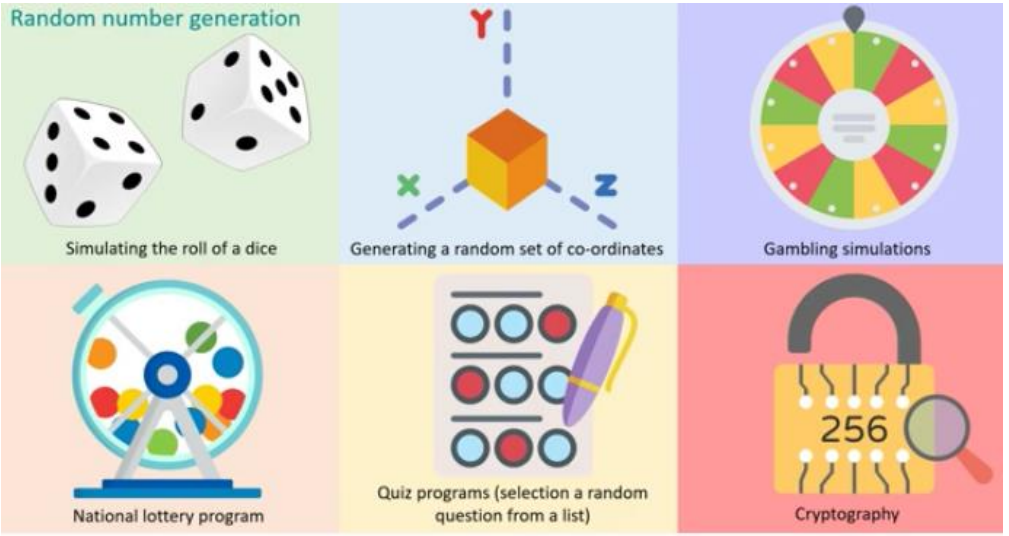
e.g. if we call name() and input Jon it would output:

Hello Jon

J277 - 2.2 Programming fundamentals

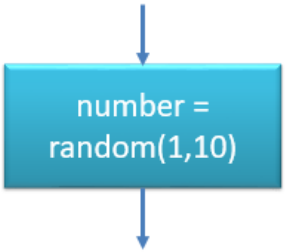
2.2.3 Additional programming techniques	
☐ The use of basic string manipulation ☐ The use of basic file handling operations: ○ Open ○ Read ○ Write ○ Close ☐ The use of records to store data ☐ The use of SQL to search for data ☐ The use of arrays (or equivalent) when solving problems, including both one-dimensional and two-dimensional arrays ☐ How to use sub programs (functions and procedures) to produce structured code ☐ Random number generation	**Required** ✓ Practical use of the additional programming techniques in a high-level language within the classroom ✓ Ability to manipulate strings, including: ▪ Concatenation ▪ Slicing ✓ Arrays as fixed length static structures ✓ The use of functions ✓ The use of procedures ✓ Where to use functions and procedures effectively ✓ SQL commands: ▪ SELECT ▪ FROM ▪ WHERE

Random number generation



Generating random numbers

- A typical method of generating random numbers might look like this:




```
month:= getRandBetween(1,12)
```

```
import random
diceScore = random.randint(1,6)
```

Syntax for exam!

- The exact structure of a random number generator will vary between programming languages.
- Using the OCR Exam Reference Language you can either state the first and last possible values
  - *e.g. diceRoll = random(1,6)*
- Or just the maximum value, if starting from 1
  - *e.g. diceRoll = random(6)*

```
import random
diceScore = random.randint(1,6)
```