Validation

Validation is the process of checking that data entered into a program is appropriate and reasonable. E.g. if a program asks you for your name and you enter 1234 it should recognise that this isn't a valid name, or if you are asked for a number between 1 and 10 and we enter 20 the program should inform us we've made a mistake, rather than just crashing.

Validation is important to prevent errors creeping into our code. Programs should never crash, even if a user does something stupid. The program should recognise that the data entered is inappropriate, should inform the user and request new data.

We can use three simple types of Validation check:
Presence check - this simply double checks that data has been entered.
Range check - this checks that data falls within acceptable boundaries, i.e. if the program needs a number between 1 and 10 it will give an error if the number entered is 0 or less, or 11 or greater.
Length check - this checks that the data entered contains the right number of characters. For instance, if you enter a mobile number it should check that it contains 11 digits.

We should build validation into all programs we write, so that invalid data can't be entered and cause the program to crash, to inform users if they've made an error, and to reduce the risk of security flaws.

Example of a Range check:

```
SET entry to 0
WHILE entry = 0 DO
    RECEIVE entry FROM (INTEGER) KEYBOARD
    IF entry <10 OR entry> 20 THEN
        SEND 'Please enter a number from 10 to 20.' TO DISPLAY
        SET entry TO 0
    END IF
END WHILE
```

Question 1 - What would be two appropriate validation checks to use if a program asked the user to enter their age?

Question 2 - Suggest a validation rule to be applied if a program requires a user to enter their mobile phone number.

Question 3 - it is very difficult to apply a validation check to an address field. Why? What problems might this cause?

Question 4 - if you create a program with no validation built in at all, what problems might a user run into when they try to use it?

Testing and test plans

Making mistakes during programming is more or less inevitable - no matter how smart you are or how good a programmer you are. Therefore, it is essential that all software is tested, and tested thoroughly. We take a **destructive** approach to testing - meaning we must actively look for errors, try to make the software fail in every way we can.

Testing is divided into two phases -

- **Iterative testing** happens during the development process. It is natural to do a little bit of programming, then test what you've written, then either fix mistakes and test again, or develop a little more and test again. We therefore go back and forth between testing and development.
- **Final** or **terminal testing** happens once the development has been completed. When we have finished writing the code, and everything seems to be working right, we should re-test the whole of the program, testing every feature again.

Final testing can further be divided into two types -

- **Alpha testing** is done by the developers themselves
- **Beta testing** is done by a small group of potential users (hence the term 'beta release', meaning a version of a program or game that isn't quite finished, but is made available to some people so that they can report back on any problems or errors)

Question 1 - what does the word iterative mean?

Question 2 - if a program was released to the users without sufficient testing, what might be the impact on the users? What might be the impact on the developers who created the software?

Question 3 - why does Beta testing often reveal mistakes and errors that Alpha testing failed to find?

Test data

In order to be effective, we must use a variety of test data. There are three main types of test data -

Normal data - testing the software under normal circumstances, using valid data entries which should cause the program do what it's supposed to do

Erroneous data - sometimes called invalid data, this is about entering incorrect data and checking that the program doesn't fail or crash, even if the user does something unexpected. E.g. If a program asks for a number, try entering a letter, or if it asks for a number between 1 and 10 try entering 200.
Boundary data - this is test data that is on the edge of what is valid and invalid. Sometimes, this is called extreme data - for instance, if a name must be fewer than 11 characters, you would enter a 10 character name to check it will be accepted. You would also test an 11 character name to make sure it was rejected.

Types of errors

We can divide the errors we make into three types -
Syntax error - syntax means the rules of grammar of a language. A syntax error in code is when you mis-spell a word, or miss out a colon or bracket. This will prevent the code from compiling.
Runtime errors - if your program compiles, but then crashes during running, this is called runtime error. This is usually caused by you asking the computer to do an impossible action, like divide by zero, or add a number to a letter.
Logic error - if your compiles, runs, but produces the wrong result, we call this a logic error. This is when you have written a program accurately, it does what you have told it to, but what you've told it to do is not what you *want* it to do.

Question 1 - write a line of code in Python that would produce a Syntax error.

Question 2 - write a line of code in Python that would produce a Runtime error.

Question 3 - write a line, or lines, of code in Python that would produce a logic error.

Test plan

A test plan is a document, usually drawn up as a table, that lays out the tests a developer intends to perform on a program. It should contain the test data that will be used, the expected outcome, the type of test, and the actual outcome.

E.g. This could be the start of a test plan for a program that averages three numbers.

Test number	Test data	Expected result	Actual result	Test type
1	2,3,4	3	3	Valid data
2	2,3,z	"Invalid entry"	Crashed	Invalid data
3	-3,4,8	3	5	Valid data

Question 1 - why is it important to produce a test plan, and follow it, rather than just working through tests as you think of them?

Question 2 - Logic errors are often the most difficult to find, why is this?

Question 3 - A program has been written to take five integers and sort them into ascending order. Fill out the following test plan for this program. Your test plan should look for more than one logic error.

Test number	Test data	Expected result	Actual result	Test type
1				
2				
3				
4				
5				