GCSE COMPUTING
KNOWLEDGE ORGANISER

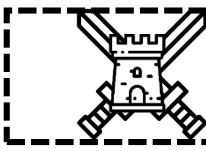
2.3 PRODUCING ROBUST PROGRAMS

2.3.1 DEFENSIVE DESIGN

Defensive design considerations:
- ❑ Anticipating misuse
- ❑ Authentication

Input validation

Maintainability:
- ❑ Use of sub programs
- ❑ Naming conventions
- ❑ Indentation
- ❑ Commenting



DEFENSIVE DESIGN means writing a program anticipating that users might either accidentally or deliberately cause it to fail.

AUTHENTICATION is used to ensure that only authorised users are able to access data in a system. This can involve methods such as **captcha** or biometric technology like finger print or face recognition.

VALIDATION is a set of rules that a program can use to ensure user input is restricted to acceptable values and does not crash the program.

Check	Description
Range check	Checks that a number/date is within a given range
Type check	Checks that the data is of an appropriate type (i.e. text, integer)
Length check	Check that the length of text is neither too short or long
Presence check	Checks that data has been entered
Format check	Checks that the structure of the data is correct – i.e. that a date is written in the correct way

MAINTAINABILITY is a way of producing code in such a way that it is easier to fix bugs and flaws because it is easier for others to read and understand. A range of techniques is available to the programmer in order to ensure maintainability.

SUB PROGRAMS make programs easier to understand. They also make errors easier to correct, since they can be isolated without affecting the main program

```
MAIN
  MENU()
  PLAYER_TURN()
  COMPUTER_TURN()
GAME OVER
```

```
for l in range(1,10):
       v =int(input("Enter your guess"))
       if x = n:
              print("Well done!")
              break
       else:
              print("Bad luck")
              print("Out of goes!")
```

```
for loop in range(1,10):
       guess=int(input("Enter your guess"))
       if guess = number:
              print("Well done!")
              break
       else:
              print("Bad luck")
print("Out of goes!")
```

NAMING CONVENTIONS are important because sensibly named constants and variables make a program easier to understand to programmers who are not familiar with it.

```
for loop in range(1,10):
guess=int(input("Enter your guess"))
if guess = number:
print("Well done!")
break
else:
print("Bad luck")
print("Out of goes!")
```

```
for loop in range(1,10):
       guess=int(input("Enter your guess"))
       if guess = number:
              print("Well done!")
              break
       else:
              print("Bad luck")
print("Out of goes!")
```

INDENTATION helps to identify blocks of code where iteration or selection are taking place. Many programming languages automatically indent code. Without indentation it can be hard to identify where there are blocks of code that do not belong to main 'sequence' of the program.

COMMENTING is used to annotate a program listing in order to explain what the code means. This is useful when more than one developer is working on a program and may not be familiar with aspects of the code.

```
#create loop to run until correct password is entered
#program checks password on file against user input
```

© J Bridgeman 2020

2.3.2 TESTING

The purpose of testing
Types of testing:
- ❑ Iterative
- ❑ Final/terminal

Identify syntax and logic errors
Selecting and using suitable test data:
- ❑ Normal
- ❑ Boundary
- ❑ Invalid
- ❑ Erroneous

Refining algorithms

THE PURPOSE OF TESTING is not only to ensure that the program works , but to ensure that it completes the tasks that it was designed to do. Testing identifies any bugs that are in the program.

ITERATIVE	TERMINAL
Carried out while the program is in development	Carried out at the end of the development process
Uses **TRACE TABLES** and a **TEST PLAN**	Checks against original plan to make sure all parts work and that it works as intended

- **SYNTAX ERRORS** are mistakes that prevent the program from running.
- All programming languages have rules - syntax - (for example, about the use of capital letters) and syntax errors occur when these rules are broken.

Capital 'P'

Missing "

```
Print("Good morning")
Name = input("What is your name?)
print("Nice to meet you" name)
```

Missing ,

REVISION NOTE
You need to be able to identify errors in given code and explain how to correct them

-Programs containing **LOGIC ERRORS** will run. However, they do not produce the output that the programmer intended
- This is because the program does not contain syntax errors, but instead has mistakes in the logic that make it behave unexpectedly.
- Logic errors are often harder to spot (and fix) then syntax errors

```
#program to calculate area of a square

side = int(input("Please enter the length)
area = side + 4
print("The area is",area)
```

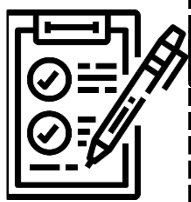
In this example, the program will run but it will not calculate the area of a square because the programmer has used this line…

```
area = side + 4
```

instead of this line…

```
area = side * 4
```

Before a program can be tested, the programmer needs to create a **TEST PLAN**. The programmer will have to identify **TEST DATA** that can be used in the program to see if it generates errors.



```
number = int(input("Please enter a number from 1 – 10))
```

Type of test data	EXPLANATION	EXAMPLE
NORMAL	Data that the program that is designed to handle	1,2,3,4,5,6,7,8,9,10
BOUNDARY /EXTREME	Data on the very edges of what is / isn't acceptable	1,10
INVALID	Data the is outside the limits set by the program	0,11
ERRONEOUS	Data that is unsuitable for the purpose -	A,B,C,#,!

Test Number	Test purpose	Test data	Expected result	Actual result

Typical plan layout

Testing allows the programmer to **REFINE ALGORITHMS.** Once the programmer has tested the program and identified the mistakes, they will need to return to the program to correct or improve the code.