OCR GCSE Computer Science

Your notes

Defensive Design & Testing

Contents

Defensive Design Considerations


Your notes

What is defensive design?

- Defensive design is an approach to software development where **every possible input** from a user is considered
- This is done to **anticipate** all of the ways a user could **misuse** a program
- Defensive design ensures that the final program is **robust** and **reliable** for all users
- Occasionally, some errors can occur in software that can not be foreseen by the developer when writing the software

Defensive Design Considerations

How can a software developer anticipate errors in programs?

- Many errors can occur in a program and some of these can be difficult to anticipate when initially developing the software
- The programmer must ensure the software has a way of dealing with the potential errors to ensure their software does not crash
- Some examples of these errors include
  - **Peripheral** errors
  - **Disk errors**
  - **Communication errors**
- In these instances, a programmer must ensure their software deals with these errors to ensure it is **robust and reliable** for its users

Peripheral errors

- Peripherals commonly don't perform as intended, such as **printers** and this can cause an issue for the end user
- If a printer runs out of **paper**, runs out of **ink** or has a **paper jam**; the user should have the option of reprinting their document
- These considerations must be pre-planned in the software

Disk errors

- Programs such as word processing software must be able to account for errors on a disk drive

- Example errors that may occur on a disk include
  - **Disk running out of space**
  - **Files and folders not being found**
  - **Corrupted files**
- These issues must be preplanned in the software the user has an alternative option, for example, saving their work on another disk


Your notes

Communication errors

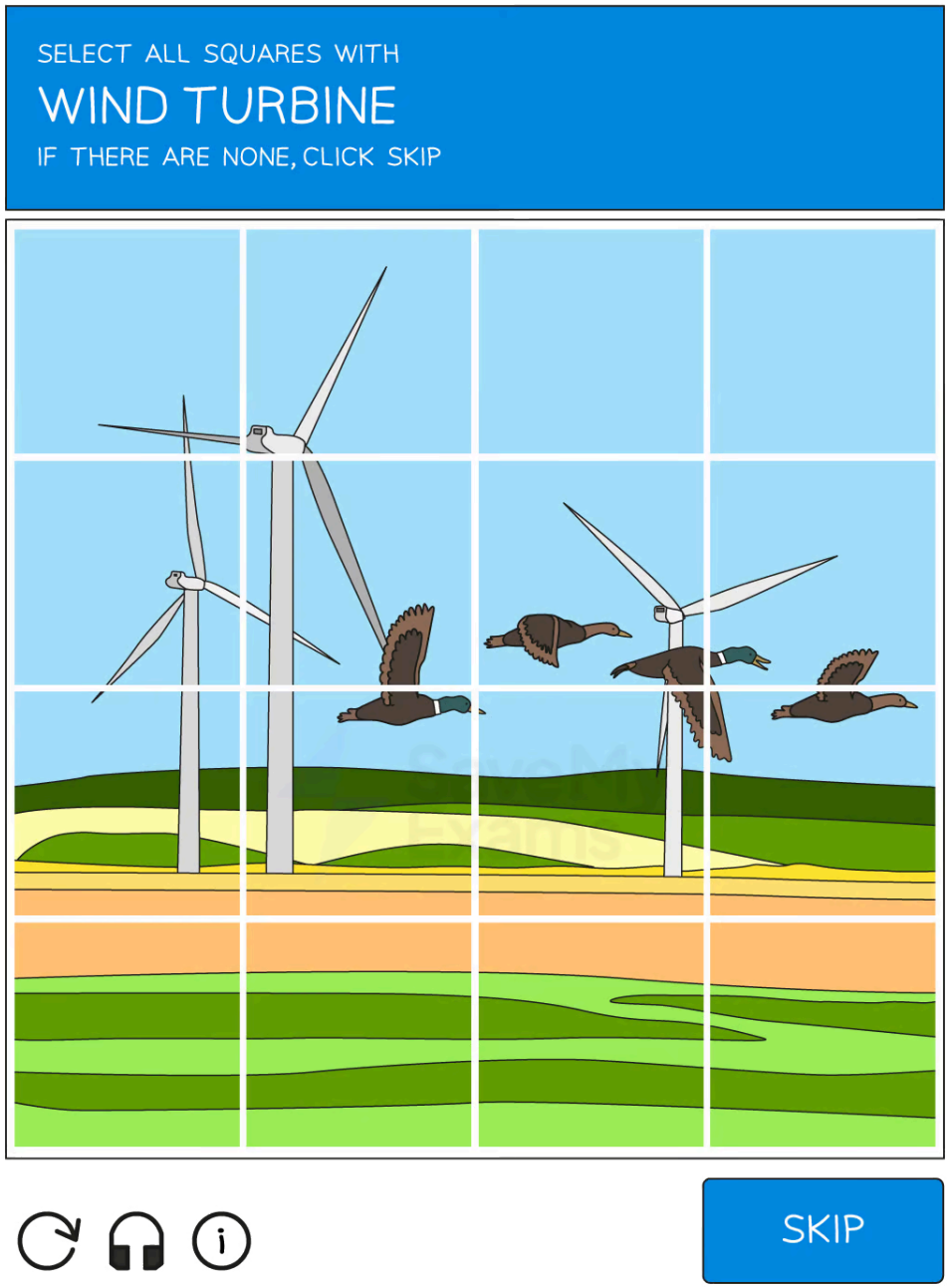
- Applications which use online systems must be connected to a host server
- When a connection is lost, the program should provide a way for the user to cancel their request and try again
- The program may also be able to automatically retry if the connection resumes

Authentication

What is Authentication?

- Authentication is the process of **ensuring that a system is secure** by asking the user to **complete tasks to prove they are an authorised** user of the system
- Authentication is done because **bots can submit data in online forms**
- Authentication can be done in several ways, these include
  - **Usernames and Passwords**
  - **CAPTCHA**
- Other methods that programmers can do to authenticate the user is include
  - **Allowing users to recover passwords via email links and SMS codes**
  - **Encrypting data**

Your notes



Your notes

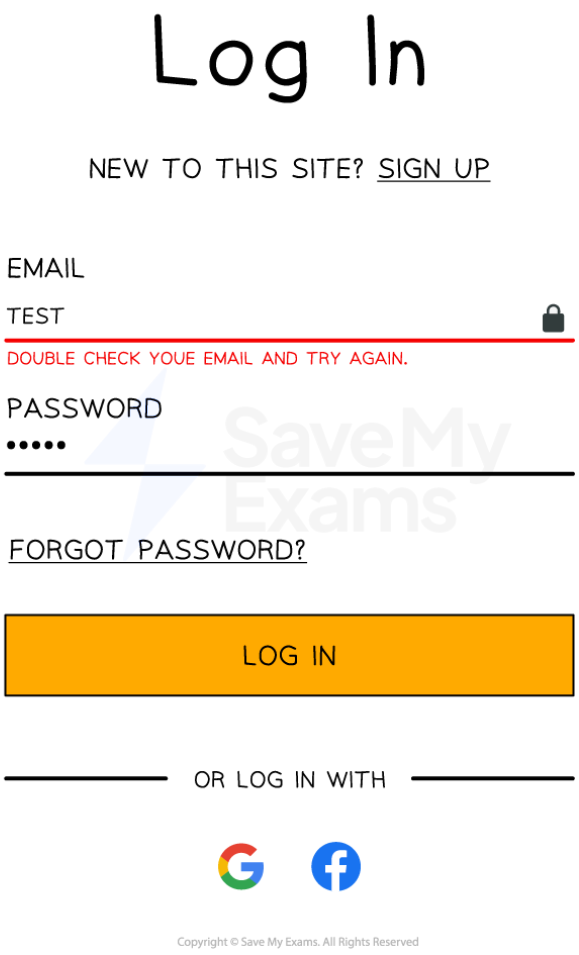
Input Validation

Input Validation

What is Input Validation?

- Input validation is code which is used to **check** that an **input** from a user is **acceptable** and that it matches the requirements of the program
- There are **5 main categories of validation** which can be carried out on **fields** and **data types**, **these are**
  - **Length check**
  - **Type check**
  - **Range check**
  - **Presence check**
  - **Format check**
- There can be occasions where more than one type of validation will be used on a field
- An example of this could be a password field which could have a length, presence and type check on it

Your notes



Length check

- Checks the **length of a string**
- An example is ensuring that a password is **8 or more characters** in length
- Code example

```
password_length = len(password)

while password_length < 8:
  password = input("Enter a password which is 8 or more characters")
```

Type check

- Check the **data type** of a field
- An example is checking a user's **age** has been entered as an **integer**, without creating an integer input
- Code example


Your notes

```
age = input("Enter your age")

while age.isdigit() == False:
    print("enter a number")
    age = input("Enter your age as a number")
```

Range check

- Ensures the data entered as a number **falls within a particular range**
- An example is checking a user's age has been entered and falls between the digits of **0-100**
- Code example

```
age = int(input("Enter your age"))

while age < 0 or age > 100:
    age = int(input("Enter your age, ensure it is between 0-100"))
```

Presence check

- Looks to see if **any data has been entered** in a field
- An example is checking that a user has entered a name when registering for a website
- Code example

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

while name == "":
    name = input("You must enter your name here")
```

Format check

- Ensures that the data has been entered in the **correct format**
- An example would be ensuring that an email includes the @ symbol and a full stop (.)
- Code example

```
email = input("Enter your email address")


while "@" not in email or "." not in email:
    email = input("Please enter a valid email address")
```


Your notes

Worked Example

A car dealership uses a computer system to record details of the cars that it has for sale. Each car has a make, model, age and number of miles driven.

The car dealership only sells cars that have fewer than 15,000 miles and are 10 years old or less.

Write an algorithm that will:

- Ask the user to enter the number of miles and the age of a car
- Validate the input to check that only sensible values that are in the given range are entered
- Output True if valid data has been entered or False if invalid data has been entered **[4]**

How to answer this question

- When answering any algorithm question, ask yourself:
  - What **inputs** and **outputs** do I need?
  - Do I need to do any **calculations** or **comparisons**?
  - Do I need to use **selection** or **iteration?**
  - Do I need to use a **function** or **procedure**?
- Re-read the algorithm question working through the criteria given

Programming Skill	Algorithm
Inputs	▪ Miles ▪ Age
Outputs	▪ True or False
Calculations / Comparisons	▪ Check for valid mileage ▪ Check for valid age
Selection or **Iteration**	▪ Selection is needed (If age <10 and miles < 15000) ▪ Iteration is not needed
Function or Procedure	▪ Not needed

Answer:

i)1 mark per bullet, max 4

- Miles and age input separately
- Checks for valid mileage
- Checks for valid age
- Checks both are greater than / greater than equal to zero
- …correctly outputs both True and False

Example Answer:

```
miles = int(input("enter miles driven"))
age = int(input("enter age of car"))
valid = True
if miles > 15000 or miles < 0 then
        valid = False
elif age > 10 or age < 0 then
        valid = False
endif
print(valid)
```

Your notes

Your notes

Program Maintainability

What is program maintainability?

- Maintainability is used to ensure **programmers can easily understand** what a program is doing months or years after having first written it
- Maintainability is also used when **programming as part of a large team**, so each programmer working on the team understands what each section is doing

How are programs maintained?

- **Commenting of code**: to explain the purpose of the code in a particular section
- **White space**: to make each section clear and easy to see
- **Indentation**: to show each instance of selection and iteration and to make it clear which code belongs to which clause in the program
- **Sensible variable names**: so that the name explains what that variable or data structure does and to prevent confusion later in the development process
- **Use of sub-programs**: functions or procedures split up programs into reusable sections of code which can remove the need for duplicating code and also increase the overall structure of the code

An example of maintainability

- The first image below shows the code for a bubble sort algorithm written in Python
- The code does not include any of the maintainability features mentioned in this section

```
a = ["London", "Madrid", "Newcastle", "York", "Birmingham", "Cambridge"]
print("Items unsorted:")
print(a)
print()
b = True
c = len(a)
while b and c > 0:
c -= 1
b = False
for i in range(0, c):
if a[i] > a[i + 1]:
x = a[i]
a[i] = a[i + 1]
a[i + 1] = x
b = True
print("Items sorted:")
print(a)
```

Your notes

- The second image, below, is the same program however, it clearly shows the use of
  - **comments**
  - **meaningful variable names**
  - **white space**
  - **indentation**
- By including each of these, the code immediately becomes easier to
  - **read**
  - **understand**
  - **debug**
  - **improve**

```
# Define a list of cities to be sorted
cities = ["London", "Madrid", "Newcastle", "York", "Birmingham", "Cambridge"]
print("Items unsorted:")
print(cities)
print()

# Initialise a variable to track whether a swap has occurred
swap_occurred = True

# Get the length of the list
list_length = len(cities)

# Perform the bubble sort algorithm
while swap_occurred and list_length > 0:
    list_length -= 1
    swap_occurred = False

    # Iterate through the list
    for i in range(0, list_length):
        # Compare adjacent elements
        if cities[i] > cities[i + 1]:
            # Swap elements if they are in the wrong order
            temp = cities[i]
            cities[i] = cities[i + 1]
            cities[i + 1] = temp
            swap_occurred = True

# Display the sorted list
print("Items sorted:")
print(cities)
```

Your notes

Worked Example

The area of a circle is calculated using the formula $\pi \times r^2$, where π is equal to 3.142 and r is the radius.

George has written a program to allow a user to enter the radius of a circle as a whole number, between 1 and 30, and output the area of the circle.

```
radius = int(0)
area = float(0.0)
radius = int(input("Enter radius: "))
if radius < 1 or radius > 30:
print("That radius is invalid")
else
area = 3.142 x(radius2)
print(area)
```

Explain, using examples from the program, **two** ways Finn can improve the **maintainability** of the program. **[6]**

How to answer this question

- To score full marks on this type of question you will be awarded
  - 1 mark for identifying a maintainability feature
  - 1 mark for explaining how it aids maintainability
  - 1 mark for applying it to the context in the question

Answers: Maximum of 3 marks per method

- Comments/annotation
  - To explain the key functions/sections
  - E.g. any relevant example, such as line 4 checks the input is valid
- Indentation
  - To show where constructs/sections start and finish
  - E.g. indenting within IF statement
- White space
  - So the code appears easier to read and understand
  - E.g. leaving a line break after the radius input


Your notes

Testing Programs

Your notes

The Purpose of Testing

What is the purpose of testing programs?

- Testing is carried out for many reasons, however, 4 main areas are:
  - To ensure there are **no errors or bugs** in the code
  - To ensure that the **code performs as it was intended**
  - To ensure no one can gain **unauthorised access** to the system
  - To check the program solves the initial problem and **meets all requirements**

Types of Testing

What are the different types of testing?

- There are **two** types of tests that developers and teams will do to ensure their programs are **robust** and **meet the requirements** that have been set out, they are:
  - **Iterative testing**
  - **Final testing**

Iterative Testing

- Each part of a program is tested
- **Every pathway** through the program is tested
  - This includes each branch/pathway inside of **IF statements**
- This is done **during the development** of the program
- Iterative testing means **repeatedly testing the program whilst continuing to make changes** and make improvements
- This method of testing ensures that the program is fully functional and working as intended

Examiner Tips and Tricks

If you get asked anything about iterative testing, just remember each time you ran your code when you were working on a program. Every time you ran the code, you were testing the program using **iterative** testing.


Your notes

Final Testing

- Testing that all parts (**modules**) of a program **work together**
- Checking the program **against real data** to ensure it **meets all of the requirements**
  - Testing the program using **normal**, **boundary** and **erroneous data**
- Final testing is done towards the **end of the development cycle**, once the entire program is complete
- Final testing can include
  - **Alpha testing**
  - **Beta testing**

Worked Example

Describe the difference between iterative testing and final testing. **[2]**

Answer: 1 mark per bullet to max 2

- Iterative is during development // repeatedly testing while making changes
- Final is when the development is (almost) complete // done after iterative testing

Identify Syntax & Logic Errors in Testing

Your notes

What are syntax errors and logic errors?

- Syntax errors and logic errors are thoroughly covered earlier in the course, you can find that revision note here
- A summary of the two error types:
  - **Syntax error:** An error that **breaks the grammatical rules of a programming language** and **stops it from running**
  - **Logic error:** Incorrect code is used that **allows the program to run**, but produces an i**ncorrect or undesired output**

Identify Syntax & Logic Errors in Testing

How can you identify errors?

- As a result of a syntax error breaking the rules of the programming language, the program will not execute
- These are easily identifiable as the IDE will provide information about what the error is
  - This is done to help the programmer be able to fix the issue
- To help practise this skill, a selection of program screenshots will be used

The errors

- In the code below, there is a program which allows the user to enter how many items they wish to add to a shopping list
- The code then allows the user to
  - Enter their chosen number of items into a list
  - Output the list to the screen
- The code contains 3 syntax errors, highlighted with blue boxes

```
1  #add as many items to a list as you the user wants
2  items=[]
3  numberOfItems = int(input("How many items do you want to add? ")
4
5  #loops through the number of items
6  for i in range(0,numberOfItems)
7      add=input("Add an item: ")
8
9      #adds the item to the list
10     items.append(add)
11
12 print(items)
13
```

Your notes

Error 1

- *Line 3:* **Missing a second bracket** at the end of integer input
  - The error message below is what the IDE provided to help the programmer identify the error
  - A top tip when programming is to look at the line of code above the one given in the error message
    - For example, this error message claims line 6 is the issue, however, the code above line 6 (line 3) is the line that contains the error

```
  File "main.py", line 6
    for i in range(0,numberOfItems):
                                    ^
SyntaxError: invalid syntax
```

Error 2

- Line 6: **Missing a colon** at the end of a **for loop**, while loop or if statement
  - As mentioned earlier, the error message identifies the line after the actual error
  - The error is on line 6, however, the syntax error message identifies the line below it
    - **Always check the line above for any potential errors**

```
  File "main.py", line 7
    add==input("Add an item: ")
    ^
SyntaxError: invalid syntax
```

Your notes

Error 3

- Line 7: **Using ==** which is used for a **comparison** instead of a **single = to declare a variable**
  - This would return the same error as above, this time the user would more easily be able to identify the issue as it is on line 7

```
  File "main.py", line 7
    add==input("Add an item: ")
    ^
SyntaxError: invalid syntax
```

The fix

- Once all fixes are in place, the code should appear as follows and the program should execute as intended

```
1  #add as many items to a list as you the user wants
2  items=[]
3  numberOfItems = int(input("How many items do you want to add? "))
4
5  #loops through the number of items
6  for i in range(0,numberOfItems):
7      add=input("Add an item: ")
8
9      #adds the item to the list
10     items.append(add)
11
12 print(items)
13
```

```
How many items do you want to add? 3
Add an item: apples
Add an item: eggs
Add an item: cheese
['apples', 'eggs', 'cheese']
```

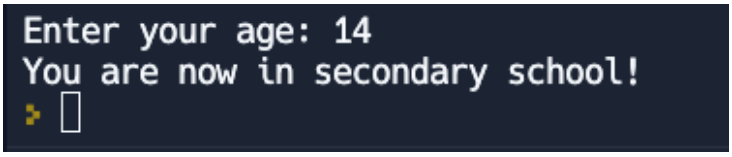
Your notes

Identifying logic errors

- Logic errors can be slightly more challenging to locate compared to syntax errors
- This is because the program will still run but will not produce the expected output that the programmer intended
- The most obvious areas to check for logic errors are:
  - Logical operators (<, >, ==, !=)
  - Boolean operators (AND, OR, NOT)
  - Division by 0
- To help demonstrate this skill, another snippet of program code is used

```
1  age = int(input("Enter your age: "))
2
3  if age > 11 or age < 18:
4      print("You are now in secondary school!")
5  else:
6      print("You are not in secondary school!")
7
```

- In this example, the incorrect Boolean operator has been used, OR instead of AND
- The result means the else clause in the if statement would never be caught

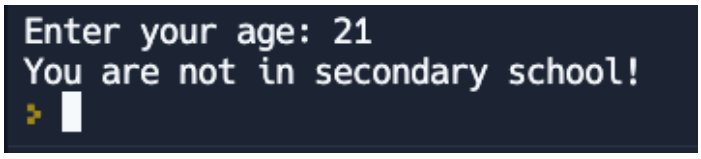


Your notes

- At first glance, entering normal test data such as 14, the program works as intended
- Entering erroneous data or boundary test data which is outside of the range would result in the error
  - When entering the age of 21, it still outputs that the user is in secondary school

- By changing the OR to AND, this corrects the logic error

```
1  age = int(input("Enter your age: "))
2
3  if age > 11 and age < 18:
4      print("You are now in secondary school!")
5  else:
6      print("You are not in secondary school!")
7
```



Worked Example

Dan is writing a program for maths students. To make sure that there are no logic errors in the program, Dan uses a test plan.

Describe what is meant by a logic error. **[2]**

Answer

- The error does not prevent the program from running
- But it does not produce the expected output / it does not do what the programmer intended
- A reasonable example


Your notes

Selecting & Using Suitable Test Data


Your notes

Normal, Boundary & Erroneous Tests

Categories of tests

- There are **three** main categories of tests a programmer or test-user would carry out whilst performing both iterative or final testing on a program
  - **Normal tests**
  - **Boundary tests**
  - **Erroneous tests**
- There is an additional test that users would always carry out to test the robustness of their program, this is known as an **invalid test**
- **To explain the types of tests, some example code will be used**

Example code

```
# Ask for user's name
name = input("What is your name? ")

# Ask for user's age
age = int(input("How old are you? "))

# Check if age is between 12 and 18
if age >= 12 and age <= 18:
    print("Welcome, " + name + "! Your age is accepted.")
else:
    print("Sorry, " + name + ". Your age is not accepted.")
```

Normal tests

- A normal test is when a user enters data that **should be accepted** in the program
- An example would be a user entering their age as 16 into the age field of the program

Boundary tests

- A boundary test is when a user enters **data that is on the edge of what is acceptable**
- An example would be a user entering their age as 12 or 18 into the age field of the program

Erroneous tests

- An erroneous test is when a user enters data that is the **wrong data type**
- An example would be a user entering their age as "F" into the age field of the program

Your notes

Invalid tests

- An invalid test is when a user enters data that is the **right data type but it is outside of what is accepted**
- An example would be a user entering their age as 67 into the age field of the program

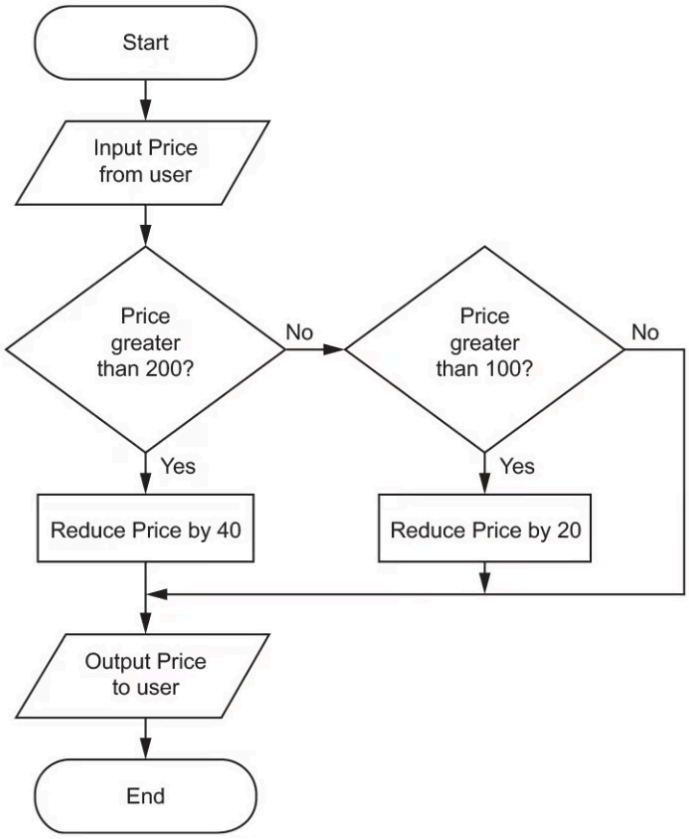
Selecting suitable test data

Type of Test	Input	Expected Output
Normal	14	Accepted
Normal	16	Accepted
Boundary	12	Accepted
Boundary	19	Rejected
Erroneous	H	Rejected
Erroneous	@	Rejected
Invalid	36	Rejected
Invalid	2	Rejected

Worked Example

SME Electronics is an online shop which sells electronic items.
The following flowchart shows an algorithm used to calculate the price of an item when they have a sale.

Your notes



1. Complete the following test plan for the algorithm **[4]**

Price Input	Test Type	Expected Price Output
50	Normal	
100		
150		
200	Boundary	
FFF		

Answer

- **Marked in pairs**
- **1 mark per two correct cells**

Price Input	Test Type	Expected Price Output
50	Normal	50
100	Boundary	100
150	Normal	130
200	Boundary	180
FFF	Erroneous	Error

Your notes