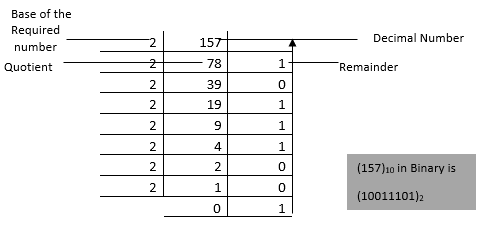
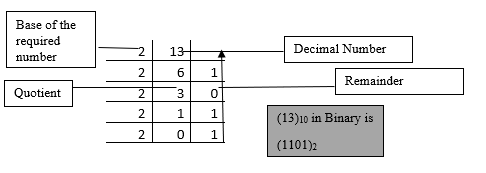
**NCV2 Basic Principles of Computer Programming and Computer Literacy**

# Topic 1:Computer hardware and software

**EXAMPLE 1.1-Replace the diagram with this**



**EXAMPLE 1.2-Reviewers suggested that we add another example in this format as well. Also the base of the result was 12 instead of 2. Typo corrected**

****

**Table 1.1 shows the representation of binary and hexadecimal representation of decimal numbers (As suggested by the reviewer, may you please add this content-Place this before example 1.3 on page 5)**

|  |  |  |
| --- | --- | --- |
| **Decimal** | **Binary** | **Hexadecimal** |
| 0 | 0000 | 0 |
| 1 | 0001 | 1 |
| 2 | 0010 | 2 |
| 3 | 0011 | 3 |
| 4 | 0100 | 4 |
| 5 | 0101 | 5 |
| 6 | 0110 | 6 |
| 7 | 0111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| 10 | 1010 | A |
| 11 | 1011 | B |
| 12 | 1100 | C |
| 13 | 1101 | D |
| 14 | 1110 | E |
| 15 | 1111 | F |

Let us look at the following binary number

010101102. To convert the number to its decimal format, we raise each bit to powers of 2 and multiply by the binary number and then add the total as follows:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |

Let’s add the total (64 x1) + (16 x1) + (4 x1) + (2 x1) =86

## 1.2.8 Typical components of the system unit (Replace all the text in the section with the following)

The system unit is the computer component that contains the essential devices necessary to do computations and generate results. The main components of the system unit are:

* Motherboard
* Processor
* Random Access Memory
* Power Supply
* Video and Audio Card
* Hard Drive

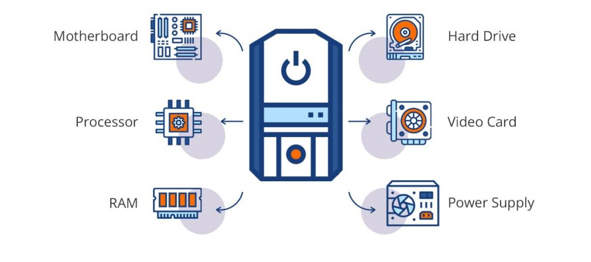


Figure 1. 14: Components of a system unit (source: <https://blog.qatestlab.com/2018/04/13/system-unit-components/>) -As per reviewer’s suggestion, I have replace the diagram of Figure 1.14 with the above-

1.2.5

(As requested by the reviewer to water down the content, I have removed some of the text for this section)

**Primary Storage**

Primary storage (also known as the main memory) is the component of the computer that holds data, programs and instructions that are currently in use.  Main memory refers to physical memory that is internal to the computer. Main memory consists of Random Access Memory (RAM) and Read Only Memory (ROM). In a computer, the RAM stores data temporarily while it is in use. RAM is volatile, i.e. data stored in it is lost when we switch off the computer or if there is a power failure. RAM serves as the ‘working memory’ of the computer, holding data generated by programs.

The read-only memory (ROM) stores the instructions that turn on the computer when a power button is pressed. ROM is a type of memory that can only be read, not written. ROM is non-volatile, i.e. it does not lose instructions when the power is turned off.

**Hard Disk Drives [Reviewer suggested that we trim some of the content**

Place the following sentence above the Figure 1.7 showing hard disk

Take a look at the inside part of a computer hard disk drive shown in

Figure 1.7.

Remove the reminder of the text

In the read/write head, an electromagnet changes the surface of the disk with a positive or negative charge. This is a representation of binary 1 and 0. The circuit boards carefully coordinate the rotation of the disk and swing of the actuator arm so that the read/write head can access the target location quickly. Hard drives eventually wear out and break due to their

moving parts.

## 1.2.9 Modular design of a computer-based system [revisited as suggested]

A modular computer system is made up of easily replaceable parts and standardised interfaces. Computer users can easily upgrade specific aspects of their computers without purchasing a brand-new one if the components use the same standard interface. Computers are an excellent example of **modular design**. In addition, because the design is modular, it allows incremental upgrades. Products or systems with modular designs are easier to customize to meet the needs or preferences of individual customers. Components of a computer can be replaced and upgraded easily.

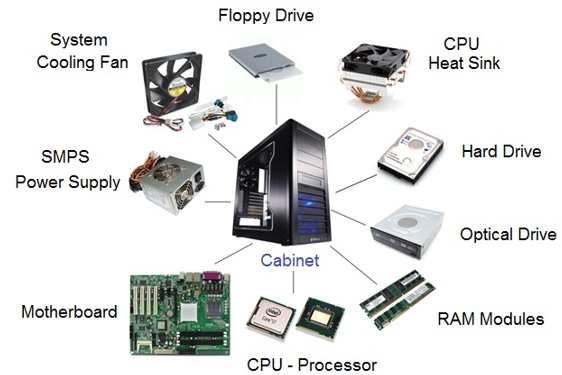


Figure 1.15: Modules of a computer system

1.3.4 Expand a Windows file path and explain each element

Another visual representation of file path is given below.

Diagram

Description automatically generated with low confidence

Add this after 1.3.5 text as part of additional Exercise as per reviewer suggestion

**Task 1.2**

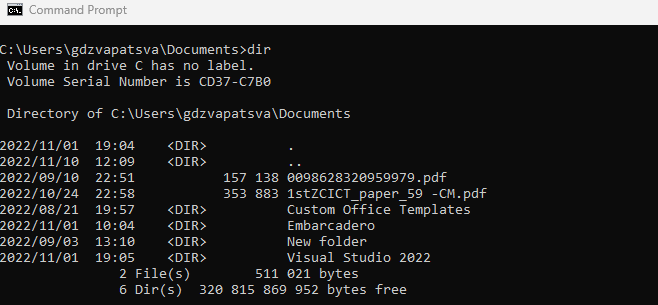
Navigate to the Documents folder and type cmd in the address bar to get to the command prompt. Alternatively, from the terminal, type cd Documents to move to the Documents directory. Use the dir command to list its contents. In my case, my output is as follows:

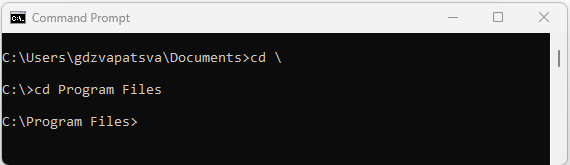
Figure 1.25: Contents of Documents directory

**1.3.6 Change directory location using the command**

Add the following exercise after the section content.

**Task 1.3**

**In this exercise, you are requested to use the cd command to navigate to the program files folder from Documents folder, which is located on the C drive. You can go to C: drive straight from any path by typing cd \ command as shown in below.**

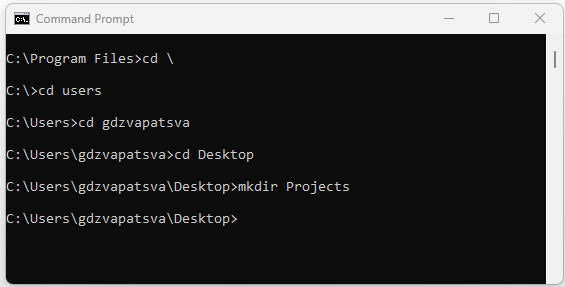
****

**1.3.7 Create a new folder using the mkdir command**

Add the following exercise after the task 1.2 (which should be 1.3) content.

**Task 1.4**

In this exercise, you must create a new directory called "Projects" on the Desktop. You will need to navigate to the Desktop as you will not have permission to create a folder in program files. We have already learnt the two ways of navigating to a specify directory of choice. Here are my steps and these are specific to my machine.



Do a similar exercise and create the suggested folder.

**1.3.8 Remove a folder using the rmdir command**

Add the following task after the content for the section as suggested.

**Task 1.5**

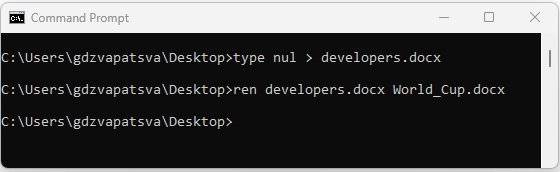
Remove the Projects directory from the desktop using the rmdir command. List the contents of the directory. Try to navigate to the same directory called Projects. What happened? In my case, I got an error message saying, "The system cannot find the path specified." This is to inform you that the directory doesn’t exist anymore, so our rmdir Projects command was successful.

**1.3.10 Rename a file using ren command**

**Add the following task after the content of the section**

Task 1.6 Create a file called developers.txt on the desktop. Rename the file to developers.exe

Here are the commands I typed in my case. Try to do the same and check if the file still exists on the desktop.



**1.3.11 Copy a file using the copy command**

**Add the following task after the content of the section**

**Task 1.7**

**In the Programming folder create a file called Qatar2022.docx. Copy the file Qatar2022.docx to NCV folder. You should see text saying "1 file(s) copied" if the command was successful.**

**1.3.13 Run an executable file from the command line**

**Add the following task after EXAMPLE 1.4**

The other easy way of starting an executable file from the command line is by typing the following in the command prompt:

start [filename.exe] and pressing enter

You can start Microsoft excel from command line: start excel.exe

Once you enter the given command, you will see Microsoft Excel starting.

# Topic 2: Problem solving process and concepts

### 2.1.2 Define the term computational thinking [The correct word is computational-I think this might have been a correction error as main document has computational-Please run control find and replace or calculational]

Computational thinking refers to the methods for solving problems by expressing them in a way that can be understood by a computer. It involves automating processes, as well as using computing to explore, analyse, and understand processes (natural as well as artificial). Computational thinking falls into four steps, regardless of whether it is applied in computer science or another subject area and these are:

* **Decomposition**- To address a complicated problem, you must first divide it into smaller, more manageable sections.
* **Abstraction**- Is the process of obtaining the most relevant information from each deconstructed problem is known as abstraction. This aids in defining or generalizing what has to be done to tackle the problem as a whole.
* **Analysis:** Solution execution and evaluation
* **Algorithmic Thinking**- It involves creating a logical, step-by-step blueprint for solving a problem that is repeatable for a predictable, consistent outcome.

VOCABULARY

Problem solving – defining the problem, determining its root cause, identifying,

prioritising and deciding on options, as well as implementing the solution

Computational thinking refers to the methods for solving problems by expressing them in a way that can be understood by a computer.

2.1.3 The phases of the program development life cycle

**[Changed as per reviewers suggestion]**

Program development lifecycle is part of the software development lifecycle, characterised by the following stages:

Requirements analysis

Design

Coding

Testing

Implementation and Support

Documentation

**Six steps in Program Development Lifecycle**

Figure 2. 1: Program Development Lifecycle

1. Requirement Analysis- The computer user must figure out the problem, then decide how to resolve the problem - choose a program.
2. Design- The development team selects the best or ideal option after investigation. In this phase, the software design document or design document specification is prepared as per the software requirements document. During the design phase, you concentrate on the core objective that the program is attempting to achieve, and then you identify all the elements that contribute to this aim. Modular programming, sometimes known as ***top-down programming***, is a style of programming that starts with a high-level description of what is to be done and this is then broken down into simpler pieces. During this process, a programmer can use tools such as **algorithms** (refer to section 2.1.1), flow charts (refer to section 2.2.8), pseudocodes and decision tables.

**VOCABULARY**

Top-down programming – programming style in which the design begins by specifying complex pieces and then dividing them into successively smaller pieces

Algorithm – procedure used for solving a problem or performing a calculation, an algorithm acts as an exact list of instructions.

1. Coding- The high-level design document is now translated into **modules** and code functions. These are distributed among different developers. Developers will write the source code, which will then be translated into machine code. Compilers, interpreters and tools are used to implement the code at this step. Coding is one of the longest phases of the PDLC, because this is where the actual solution is produced.

**VOCABULARY**

Module – any of several distinct but interrelated units from which a program may be

built up/into which a complex activity may be analysed

**[Added more compilers as per reviewers suggestion]**

**Compilers** take the entire source code and translate it into object code at one go. As soon as the object code has been converted, it can be run at any time without assistance. The process is known as compilation. Examples of compilers include Microsoft Visual Studio, GNU Compiler Collection (GCC), Clang/LLVM , Keil C++ compiler, MinGW, Turbo C, Common Business Oriented Language (COBOL), [Dev-C++](https://www.softwaretestinghelp.com/best-cpp-compiler-ide/#5_Dev-C), [NetBeans IDE](https://www.softwaretestinghelp.com/best-cpp-compiler-ide/#6_NetBeans_IDE), [Cygwin](https://www.softwaretestinghelp.com/best-cpp-compiler-ide/#7_Cygwin) etc. Some compilers come packaged within IDE and some have to be installed separated.

Reviewers suggested that a diagram be added

Source code

Compiler

Machine Code

Error

Output

**Advantages of compilers[Changed as per reviewers suggestion]**

1. **Independence**-The compiler converts commands into machine language binaries; no other program or application is required to run the executable file of source codes.
2. **Optimisation**-The compiler converts source code into a specific machine language for targeted hardware that is well optimized and runs faster.
3. **Data Security**-The compiler generates executable files that can be executed on any other system without the need for actual source code, making the program unhackable, secure, and private.
4. **Speed-**Compiler runs faster than interpreter

**Disadvantages of compilers[Changed as per reviewers suggestion]**

1. **Hardware Specific**- the programs are therefore compiled for a particular OS and architecture.
2. **Time consuming**- Compiler takes time to compile sources code.
3. **Extra memory**- As it creates a new file, the compiler consumes additional memory.
4. **Debugging Difficulty**- Compilers is that it makes it difficult to debug identified errors because it returns all errors at once after reading the source code.

**Interpreters**

**Interpreters** translate source code into object code one instruction at a time. It is the equivalent of a human interpreter who translates what is being said into another language directly, i.e. while the speaker is addressing the audience. The translated object code is then executed. This process is known as *interpretation*. Examples of interpreters include OCaml, List Processing (LISP), Python, etc.

Source code

Output

Interpreter

Error

**Advantages of interpreters[Changed as per reviewers suggestion]**

1. **Cross-Platform** In an interpreted language, source code that can run on any system is shared directly without regard for system incompatibility.
2. **Easy to debug-**Returns error on the spot as it does line by line translation.
3. **Less memory-** interpreters don’t generate new separate files which require extra memory as in compilers.
4. **Execution control-**Programmer has full control of the program and can stop it at any time and make changes.

**Disadvantages of interpreters[Changed as per reviewers suggestion]**

1. **Slower**- Because it reads, analyses, and converts code line by line, the interpreter is slower than the compiler.
2. **Dependencies file required-** **To execute the code, anyone with access to the shared source code must have an interpreter installed on their system.**
3. **Less Secure-** **Since an interpreter does not generate an executable file, we must share our source code with others, which is neither secure nor private.**

**Assemblers Retained as is**

**Assemblers** translate assembly language into object code. In contrast to compilers and interpreters, assemblers create one machine code instruction for each assembly instruction. Examples of assemblers include Fortran Assembly Program (FAP), Macro Assembly Program (MAP) and Symbolic Optimal Assembly Program (SOAP).

**Advantages of assemblers**

1. It is easier to fix errors and alter program instructions.
2. Like machine-level language, it is executable.
3. The symbolic programming is easier to understand.

**Disadvantages of assemblers**

1. It is difficult to maintain.
2. The design of a program can be invalidated by a small change.
3. It is machine-dependent.

[Retained as is]

1. **Debugging/Testing -***Software testing* is the practice of comparing software to its requirements. Testing is used to assure the quality of the final product. Software testing can only assist in the detection of flaws; it cannot ensure the absence of defects. Finding flaws in the product development cycle sooner rather than later is far more cost-effective. The product is tested for bugs by a group of testers. ***Bugs*** are errors in the program. They need to be corrected. *Debugging* is the process of finding the bugs in the computer program. Several testing levels are described in the Table 2.1.

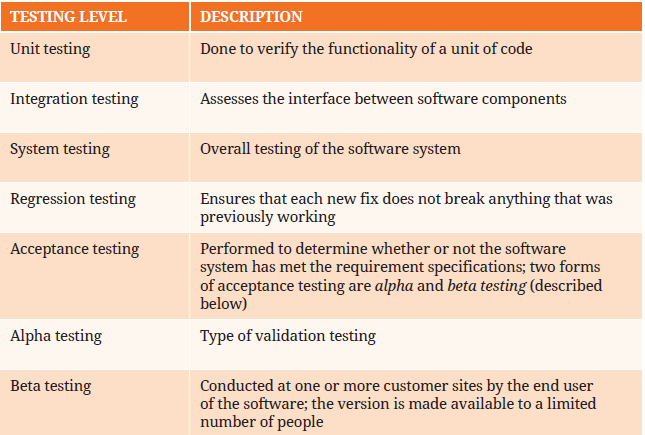


Table 2. 1: Software testing levels

**5: Implementation and Support [changed as per reviewers’ comments]**

One must run the program to make sure there are no syntax and logic errors. Once the solution has been thoroughly tested for logic and syntax errors, **deployment** will commence. The project manager deploys the product live for real-time use by the client. After the implementation, the software requires to be maintained during its life span. Developers/Maintenance teams should keep the product working at its full potential and ensure that the user has no trouble using it.

**6. Documentation [Changed]**

This is the last step in putting everything together. Internal documentation is included in this step because it explains why a change was made to the program or how to write a program.

**VOCABULARY [Retained as is]**

Bug –coding error in a computer program.

Deployment – all the processes involved in getting new software (or hardware) up and

running properly.

### 2.1.4 Describe the purpose of problem solving leading to solutions [Changed as per reviewers suggestions]-Page 37

Problems are part of life and we cannot escape them but we need to make sure that when they occur, we come up with a way to resolve them. Simply put, problem-solving is the process of identifying, defining, analysing, and resolving issues. The goal of problem solving is to overcome obstacles and come up with satisfactory solutions to the initial problem. The ability to solve problems is important for both individuals and organisations, as it allows us to exercise control over our environment.

Here are some of the reasons for problem solving:

* Fixing things that are broken
* Addressing risk
* Improving performance
* Seizing opportunity.

There are tools that help you visualize the problem and organize the information into common categories. Common examples of these diagrams include affinity diagrams, cause and effect diagrams and flowchart diagrams among others.

### 2.1.5 Problem solving steps Add the following just after the first paragraph in this section.

The FOUR stages are:

1. Understand the problem
2. Devise a plan
3. Carry out the plan
4. Look back

Let us look at the scenario below and apply the Polya’s FOUR stage of problem solving.

Tendai was having a birthday party and decided to celebrate with his classmates at KW TVET College. She invited 20 girls and 10 boys. She made one dozen blue muffins and 3 dozen red muffins. At the end of the party there were only 5 muffins left. How many muffins were eaten.

**Solution**

**Understand the problem**

* State the problem in your own words
* Figure out the unknowns
* Figure out what the problem tells you is important
* Identify any irrelevant information to the problem
* So in our case, the information about gender, name of the college and colour of muffins is not important.
* What is important is how many muffins were left from the total made
* So we understand the problem.

**Devise a plan**

To solve a problem, we can:

* Look for a pattern
* Review similar problems
* Make diagrams or charts from the problem presented
* Use guessing and checking
* Identify a sub-goal
* In our scenario, we can identify a sub-goal which is the total number of muffins made. We can then write the equation with the unknown to find the solution as shown

(3 dozen + 1 dozen) – 5 muffins= number eaten

**Carry Out the Plan**

I dozen = 12

(3 x 12) +(1 x 12) – 5=number eaten

36+12-5=number eaten

48-5=number eaten

Number eaten =43

**Looking Back**

Have we calculated the total number of muffins which were eaten. Answer is yes.

Can we verify our answer.

Yes. 48 -5 = 43 muffins.

Let us consider the second scenario

### 2.1.6 Using appropriate tools and techniques to present a solution [Reviewer suggested adding examples]

**Add**

**Additional user Story Insert on page 38 after fourth bullet and before the following sentence “User stories are typically written ….”**

As a developer, I want to code the different classes for this project so that fellow programmers can reuse them in their specific code.

**How to write user stories**

When writing user stories, keep the following in mind:

1. **Definition of “done”- When the user can complete the outlined task, the story is generally considered "done," but make sure to define what that is.**
2. **Outline subtasks or tasks-Determine which specific steps must be completed and who is responsible for each.**
3. **User personas** -Who is it for? Consider creating multiple stories if there are multiple end users.
4. **Ordered Steps** -Create a story for each step of a larger process.
5. **Listen to feedback-There's no need to make up stories when you can get them from your customers.**
6. **Time-Stories should be completed in a single sprint.**

As we have already seen, User stories are often expressed in a simple sentence, structured as follows:

**“As a [persona], I [want to], [so that].”**

Breaking this down:

"As a [persona]": Who are we building this for? We’re not just after a job title, we’re after the persona of the person.

“Wants to”: Here we’re describing the intent — not the features used. What is it we are actually trying to achieve?

“So that”- What is the big problem that needs solving?

User mapping is mostly used in agile software development methodologies. In some cases,

### 2.2.2 Examples of algorithms in life [Page 41]

**Vocabulary**

Recursive – method of solving a computational problem where the solution depends on

solutions to smaller instances of the same problem.

Activity 2.1

2. List the four categories of computational thinking. (4)

### 2.2.3 Construct and devise an algorithm/basic instruction to complete similar

### tasks

Let us move on to use algorithms in solving computational problems. We will start by listing the guidelines of developing the algorithms. [Page 43]

[As per reviewers comments, I have replaced the examples as

**EXAMPLE 2.3**

**Problem: Write a program to find the sum of two numbers entered by the user through the keyboard**

**STEP 1**: Start the program.

**STEP 2**: Read the values of first\_number and second\_number

STEP 3: Compute the sum of the entered numbers ‘first\_number’,’second\_number’,

Sum=first\_number + second\_number.

STEP 4: Print the value of ‘sum’.

STEP 5: Stop the program.

Example 2.3 illustrated how the summation of two numbers is done following algorithm technique. Once a programmer understands the logic and models it correctly using the algorithm, changing it to programming language would be much easier. Values for first\_number and second\_number must be stored in memory locations and these are referred to as variables. We will delve deeper into concepts of variables in chapter 6.

**EXAMPLE 2.4 [Replace the challenge with this one]**

**Problem: Write a program to convert Celsius to Fahrenheit conversion.**

**STEP 1:** Start the program.

**STEP 2**: Read the value of “Celsius “

**STEP 3:** Conversion of Celsius to Fahrenheit by using the formula**:**

Fahrenheit = (1.8\*Celsius) +32

**STEP 4:** Print the value of ‘Fahrenheit’

**STEP 5:** Stop the program.

Add this Before Example 2.5

The previous two examples demonstrated a sequential execution of steps. In some cases, some steps must be repeated several times for a certain condition to be true and then the program continues. This is called looping. More detailed explanation on looping is done in chapter 9. For now, we will design a small algorithm which requires the looping concept.

### 2.2.6 The various parts of an IPO chart [Remove the whole paragraph just after the sub-heading before EXAMPLE 2.6] and replace with the following]Reviewer suggested that we add more flesh to this. However, in section 2.2.9 more examples have been added.

The IPO chart will consist of three columns representing input, process and output. Input Process Output tables, or IPO tables for short, are an effective way to model the important processing going on in your system. Let's consider the three parts of the table:

* **Input**- Data which is required to create the required outputs. This is supplied by the user.
* **Process** - The steps or process involved in creating the outputs from the inputs.
* **Output**- A piece of information which we want as the result. This is normally displayed to the screen or printed out.

Let us start with a simple mathematical challenge to accept two numbers from the keyboard and perform a mathematic calculation. The challenges are normally phrased as in the example below.

EXAMPLE 2.6 [Not to be changed]

Add the following as an additional challenge under Task 2.1 as the second question on page 46

Create an IPO chart for the following problem clearly identifying the inputs, processing, and output (s). A program is required to calculate the total paint required and cost to paint a room.

Show your answer to your lecturer.

### 2.2.9 Exploring and creating algorithms in the form of an IPO chart and a flow chart

Reviewers suggested that we should consider adding more examples.

Add the following examples after EXAMPLE 2:11

**EXAMPLE 2:12**

Program

You are requested to write a program to find whether a number is odd or even.

Use an IPO chart to depict the given scenario.

**Solution**

|  |  |  |
| --- | --- | --- |
| **Input** | **Processing** | **Output** |
| Enter the number to be checked | If number% 2=0  Result=” even”  Else  Result=odd” | Display result |

Use the same program in Example 2:12 and depict the same scenario using an algorithm.

**Step 1**: Start

**Step 2**: Enter the number to be checked

**Step 3**: Check condition.

Result= number% 2

If result=0

Display “even”

Else

Display “odd”

**Step 4**: Stop

**Flowchart**

A picture containing text, businesscard

Description automatically generated

**Flowchart to check if number is odd or even**

**EXAMPLE 2:13**

Problem

You are required to design a program to find the factorial of a number. Use an IPO chart and an algorithm to depict the scenario.

**NOTE**: Factorial of a positive integer (number) is the sum of multiplication of all the integers smaller than that positive integer.

**IPO Chart**

|  |  |  |
| --- | --- | --- |
| **Input** | **Processing** | **Output** |
| Enter number to be converted *n* | Initialise variables  *factorial*=1  *i*=1  *factorial*=*factorial* \* 1  *i=i+*1 | Display *factorial* |

**Algorithm**

**Step 1**: Start

**Step 2**: Declare variables *n, factorial* and *i*

**Step 3**: Initialise variables

*factorial* =1

*i*=1

**Step 4**: Enter the number n

**Step 5**: Repeat the steps until *i*= *n*

5.1 *factoria*l = *factorial* \* 1

5.2 *i*=*i*+1

**Step 6**: Display *factorial*

**Step 7**: Stop

Diagram, Teams

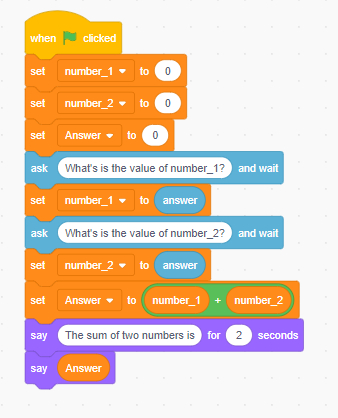
Description automatically generated

**Flowchart to depict factorial of a number**

Adding two numbers page 114 [Reviewer suggested that we should add a scratch program to depict the scenario]

**Adding two numbers using Visual Programming Language.**

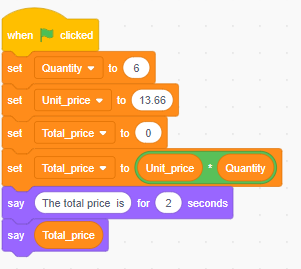
* You have seen how we can use Scratch to solve programming challenges in Chapter 3. Here we re going to show you again how we can solve the same program using Scratch before moving to Python. Here are the steps:
* Add when clicked event block
* Add variables called number\_1, number\_2 and answer and set them to 0
* From the looks select “ask …. wait block” and type the text to tell user to enter the value of number\_1 and number\_2
* From the variables block, select the set…block and add the Answer variable. Select the operators block and drag number\_1 and Number\_2 from the variables and plug them into the two spaces.
* Select the “say … for 2 seconds” block and enter the following text “The sum of the two numbers is”
* Finally, Add a “say ...” block and add the answer variable.
* Run your code and enter the two values. The code will look as the one below:



Your output will be displayed by the default Sprite.

Add the following code after the flowchart on page 125.

Here is the Scratch script for calculating the total price. Take note that the output is formatted correct to 2 decimal places. The values of Quantity and Unit\_price have been preset to 6 and 13.66 respectively.



When you run the script, the result is 81.96. We will now implement the same code in Python.

GLOSSARY PAGE 285 [Remove calculational thinking]

Computational thinking is a set of interconnected skills and practices for solving complex problems, a method of learning topics in many disciplines, and a requirement for fully participating in a computational world.

SOURCES

<https://www.businesscompilerng.com/2022/02/advantages-and-disadvantages-of_21.html>

<https://buggyprogrammer.com/advantages-and-disadvantages-of-compiler-and-interpreter/>

<https://business.tutsplus.com/tutorials/problem-solving-skills-that-lead-to-solutions--cms-39747>

# Topic 3: Concepts of programming for single-board

# microprocessors or microcontrollers

The reviewers suggested that we add similar solutions using scratch for 2.8, 2.9, 2.10 and 2.11

Add the following after Figure 3.14

**Comparing Two numbers using Scratch**

Remember the flowchart for the same problem in example 2.18. You are required to write a Scratch program to compare two numbers and show which one is greater. The program should also tell the user if both numbers are the same. Here is the script for comparing two numbers entered by the user. In this example, you will need to create three variables i.e. number\_1, and number\_2 and then follow the given script.

**Solution**

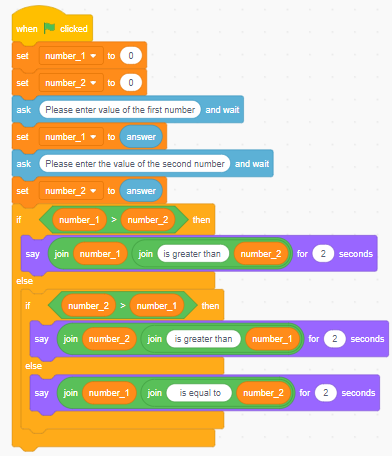


Figure 3. 15: Comparing two numbers

**Comparing Three numbers using Scratch**

Remember the flowchart for the same problem in example 2.11. Here is the script for comparing three numbers entered by the user. In this example, you will need to create three variables i.e. number\_1, number\_2 and number\_3 and then follow the given script

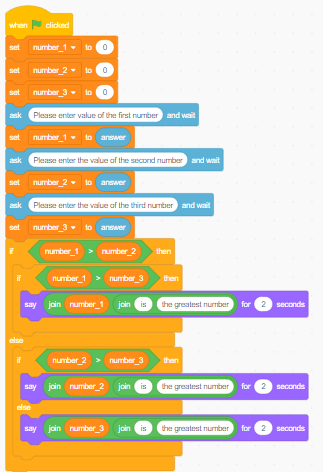


Figure 3. 16: Comparing three numbers

INSERT THIS EXAMPLE on page 71 after the paragraph with the heading Repeating actions

Here is another scenario which was represented using a flowchart in example 2.10.

**Problem**

Design a program that calculates the sum of integer numbers from 1 to 100 numbers from 1 to 100. In this scenario, you will need to make two variables i and sum. The variables are all initialised to 0.- Here is the Scratch script for the challenge.

A picture containing graphical user interface

Description automatically generated

Figure 3. 17: Sum of all integer numbers between 1 and 100

Run your code and the output must be 5050.

Let us move on to another challenge which was represented in example 2.10 using a flowchart.

**Problem:**

Several employees work in a factory. If an employee is present, calculate the wage for eight hours; if an employee is absent, do not calculate the wage, just enter a zero (no work, no pay). The hourly rate is R250,00. So, depending on the state of the employee (whether present or  not), you are asking the program to do one of two things:

1. When employee is present calculate wage; or
2. When employee is absent put wage as 0.

Key points: Make sure you create three variables:

* hours
* salary
* present

Use scratch to solve the problem. If the worker is present, the salary must be R2000 else the salary is R0.00

Solution



Figure 3. 18: Calculating salary of an employee

Change Figure 3.15 caption to Figure 3.19 and all other subsequent figure numbering will be affected

**Practical activity 3.1 [content below is added after the first practical question on page 77]**

Add the following as second practical exercise

**Question 2**

You have been tasked by JHB Metro Police to write a Scratch program for their new Camera Driver Demerit System. The program should ask the input the driver current speed in km/h and the average allowed speed of the road. The program should conform to the following requirements:

1. If the speed is less than 60, it should print “**OK**”.
2. Otherwise, for every 5km above the speed limit ( e.g 65 km in a 60km/hour zone), it should give the driver one demerit point and print the total number of demerit points. For example, if the speed is 70km/h and the location’s average is 60km/h, it should print: “**Points: 2**”. If the driver is over the limit by less than 5km/h the demerit points will be calculated accordingly.
3. If the driver gets more than 12 demerit points, the function should print “**You are to jail!**”

[15 marks]

**Question 3** Add the following as third practical exercise

Write a Scratch program to enter a test score of a subject. The program should display the grade. The grading and symbol follow the following criteria

# Grading System.

|  |  |
| --- | --- |
| Mark | Grade |
| 90-100 | A |
| 70-89 | B |
| 50-69 | C |
| 40-49 | D |
| 0-39 | E |

[15 marks]

**Question 4** Add the following as fourth practical exercise

The Bright Light Company is increasing the salaries of their employees according to their departments as can be seen in the table below.

|  |  |  |
| --- | --- | --- |
| **Department code** | **Percentage increase** | **Current Salary** |
| A | 7.2 | 45000 |
| B | 6.8 | 30550 |
| Other | 6.3 | 24800 |

Write a Scratch code to calculate the new monthly salary. The program should allow the user to enter the current salary. The program must display the new salary.

[10 marks]

**Question 5** Add the following as fifth practical exercise

Write a program to count the total number of digits in a number in Scratch.

For example, the number is **75869**, so the output should be **5**. In this program, the user enters a number through the keyboard and Scratch program should be able to count the number of digits and display the out as follows:

75869 has 5 digits

[10 marks]

**Question 6** Add the following as sixth practical exercise

Number Guessing game

You are required to code a Scratch program to allow a user to play a number guessing game. The program must allow a user to guess a number between 1 and 100 and if the random number generated by the computer matches the guessed number, the program will display "Correct Guess“. If the number guessed is smaller than the generated number, the program must display "The number guessed is too small“. If the guessed number is greater than the random number, the program must display "The number guessed is too big.“ The program must allow the user to play 6 times, after which the program will display the random number.

[15 marks]

Also Add the solutions to the lecturer guide. These are provided in the other file

#### 3.1.5.6 Adding buttons [Reviewers suggested that we add more text. Add this after the paragraph under subheading Adding Buttons on page 69]

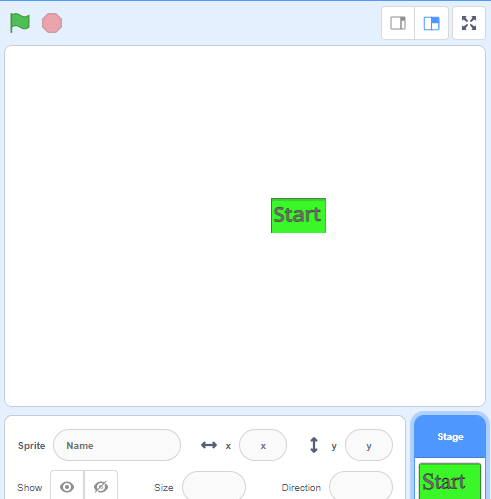
To create a button, select the backdrop tab.

For this example, we are going to create a start button.

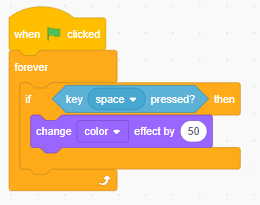
On the right side of the backdrop space, select the rectangle icon. This will bring a rectangle box in the centre of the drawing section. Select the T icon to allow you to write text. Type the word start

You can also apply a colour of your choice to the button by selecting the colour from the fill option. Same applies to the outline.

You will notice our start button in the stage area.



Let us apply some functionality to our button. There are a lot of options you can do to the button. You can make increase its size, rotate, or shrink when you hover over it. In our case we are going to change the button colour effect every time when we a spacebar is pressed. Here is the script to apply the code.



Every time when the space bar is pressed the colour of the button changes.

#### 3.1.5.14 Manipulating strings [Reviewers suggested that we should add more info. Please add the text below on page 71 under the subheading manipulating Strings. It’s a continuation to the existing paragraph]

Examples of strings: “A”, “b”, “I like Pi”, “$400”, “(^\_^)”, “Sentences etc.”. This section demonstrates a couple of string manipulation techniques. There are four string operations as shown in Table 3.1

Table 3. 1: String operations [Note table numbers will be affected. The one on page 72 becomes Table 3.2, the one on page 92 becomes 3.3]

|  |  |
| --- | --- |
| **Operation** | **Output** |
|  | join (I am an) (NCV student) returns I am an NCV Student (there is a space after apple) |
|  | letter (1) of (TVET) returns T |
|  | length of (South Africa) returns 12 |
|  | (Student) contains (Z) returns False |

**Try it out**

Write a Scratch code to accept your name and surname and print out the result as:

My full name is <first name> <surname>

**Reversing a string**

We can accept a string and reverse it. To reverse a string, we extract letter by letter and assemble them backwards.

* erase the result variable
* set the index i to the first character
* repeat for the length of text
* join the i-th letter in front of the result
* increment index i

**Challenge**

Write a Scratch program to accept a string and print out the string in reverse order.

**HINT**

In this example, you will need three variables:

* *word*-to store the text
* *i*- Is the counter. The length of our text will be obtained by using length of.
* *display*-the variable to store output.

Solution

A picture containing chart

Description automatically generated

Repeat a string

The **repeat** function repeats a text n times. Scratch functions do not allow a return value. To return a result, we define a variable which we call **result**. The **split** function will require two return strings, so we define these two variables.

**Exercise**

Write a scratch code to repeat a string 5 times. The code should either accept a letter or word or sentence.

Solution

Chart

Description automatically generated

#### 3.1.5.15 Basic list operations [As suggested by reviewers, add the section below after the sub heading “ Basic list operations on page 71. This should come after Figure 3.15 List stack blocks.]

**How to make list in Scratch?**

List in scratch can be found under variables.

At first, you will see an empty place with the option of adding your list in a grey rectangle saying Make a List.

**Step 1**: Click the **add your list** option, a dialogue box will appear asking you to enter the name of the list you want to add.

**Step 2**: Type name of a list and click OK

The list block with all the related list block generated. With this you are able to add items, delete and replace the element in the list based on specification.

**EXAMPLE**

Write a program to populate a list of numbers from 1 to 10 into the list.

HINT: You will only need one one variable to hold initial value of the starting point. The value is incremented by 1 until its greater than 10.

**Solution**

Graphical user interface

Description automatically generated

Now you have seen how you can create a list of items. When working with numbers you can also skip specific numbers. For instance, if the last set block the incrementing value is set to 2 instead of 1. Try it and compare your answers with your classmates. Did you notice that the list now shows all odd numbers.

You can also replace an element at a specific position with a specific number or text. For instance, if we add the following block:

Graphical user interface

Description automatically generated with low confidence

The value at position 4 will be replaced with text "thing".  Take note that the value 4 indicates the fourth value starting the count from 1 not 0.

**EXERCISE**

Write a Scratch program to display numbers from 100 to 90 and show the length of the list

Solution

Graphical user interface

Description automatically generated

#### 3.1.5.17 Incorporating multiple sprites

Sometimes you want to have one sprite but changing its functionalities or roles. You can make several copies of the same sprites, but it becomes messy in the stage area. What you can do is use the same sprite but configure it to have different colours. This will be done under the costumes tab by duplicating the sprite and giving each copy a different colour.

EXAMPLE

Step 1 : Start new project

Step 2: By default the cat sprite is selected. Click the Costumes tab and right click on the cat sprite. Select duplicate and change the fill colour. Now you have two sprites.

Step 3: Set the sprite 1 and 3 to flip horizontal so that they can face the opposite direction.

Step 4: Under the code tab, select make block and type in clone. Add the two input and type x and y and click OK

Step 5: Under Code tab go to Control and select the create *clone of myself block.* This allows you to position the different clone on different areas inside the stage area.

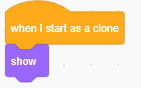
Step 6: Under Motion select the *go to x: y:* block

Step 7: Under the Looks, choose *next custome()* block. Your script should look as follows

Graphical user interface

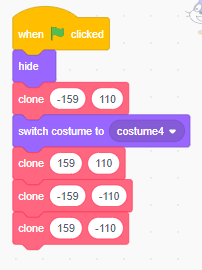
Description automatically generated

Step 8: Add the following script in the code area.



Step 9: Now add the when clicked and hide blocks in the code area. Add the clone and set the x and y to -160 and 110 respectively.

Step 10 : Add the *switch custome* block. Duplicate the clone and set the values as in the code below:

****

The final full script of the code will look as follows:

**Graphical user interface, application, whiteboard

Description automatically generated**

**3.3.6 Drawing a line using the forward function**

Python recognizes many colour names, including standards such as red, green, blue, and cyan, as well as options such as lightgreen, turquoise, skyblue, and so on. The best way to find out if Python recognizes a colour is to give it a shot!

In addition to colour names, Python accepts hex codes. A hex code is a six-character code that describes how to mix various amounts of red, green, and blue to create a specific colour. The code must be followed by a # character. To choose any hex code, you may consider the online link and set a colour of your choice. The platform will generate the hex colour code for you. Here is the link for the website: <https://g.co/kgs/JxzwhP>

# Module 6: Data types, variables and output

6.3 Arithmetic operations

6.3.1 Different arithmetic operators [Insert after the bulleted list in 6.3.1]

Table 6.2 gives a quick explanation of each operator. We are going to use variables num\_1 and num\_2 to give output in num\_3

num\_1=13, num\_2=6

Table 6. 1: Arithmetic operators

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Output** |
| + | Addition | num\_3=num\_1+num\_2  print(num3) | 19 |
| - | Subtraction | num\_3=num\_1 - num\_2  print(num3) | 7 |
| \* | Multiplication | num\_3=num\_1 \* num\_2  print(num3) | 78 |
| / | Division | num\_3=num\_1 / num\_2  print(num3) | 2.1666666666666665 |
| // | Integer division | num\_3=num\_1 / num\_2  print(num3) | 2 |
| % | Modulus | num\_3=num\_1 / num\_2  print(num3) | 1 |
| \*\* | Exponentiation | num\_3=num\_1 \*\* num\_2  print(num3) | 4826809 |

**Activity 6.3**

**10. Consider the following program below.**

#declaring variables

num\_1="13"

num\_2="6"

#initialising variable num\_3

num\_3

#addition

num\_3=num\_1+num\_2

print(num\_3)

**Modify the program to do integer addition and give correct output as 19.**

**(3)**

**TOTAL:25**

**Activity 6.4**

**Activity 6.5**

3. Write a program to create a new word called string\_3 by appending string\_2 in the middle of string\_1. String\_1 and string\_2 are given by the user.

For example:

string\_1=South

string\_2=Africa

string\_3 = SoAfricauth

[12]

**TOTAL :30**

**Activity 6.6**

9. Consider the following program intended to create a text file called robotics.txt and add a single line which reads “ Hello robotics coders”

#This program is supposed to create a file called robotics.txt

#declaring a file object to hold the file and open it

fp = ('robotics.txt', 'w')

fp.write('')

fp.close()

When executed, the program generates errors. Modify the program to make sure that it does the following:

Create a text file called “robotics.txt” (2)

Add one line of text which reads “ Hello robotics coders” (2)

**TOTAL :40**

**SUMMATIVE ASSESSMENT**

10. Consider the following Python Code to create a Text file called robotics\_lesson1.txt

#This program is supposed to create a file called robotics.txt

# create a empty text file

#declaring a file object to hold the file and open it

# writing new content to the file

fp = open("", 'w')

fp.write("" )

print('Done Writing')

fp.close()

# Open the file for writing the new contents

fp = open("robotics\_lesson1.txt", 'w')

print(fp.write())

fp.close()

Modify the program to be able to do the following

: add the following lines to the text file when it is created

text\_1 = "I have learnt to program on a Raspberry Pi"

text\_2="We learnt how to create a program like traffic lights"

text\_3="I enjoyed the lesson so much"

text\_4="Have a good day!"

: The program **should then open the** created file for reading and print the four lines on to the console.

**(10)**

**11. You are required to write a program to produce an invoice in a text file. The invoice must look as the one below.**

**Table

Description automatically generated (10)**

**TOTAL :50**

# Module 7:Math, interactive input, constants and errors

**String formatting in Python [Insert this as the first paragraph on page 179]**

Prior to Python 3.6, there were two ways to embed Python expressions inside string literals for formatting: %-formatting and str.format (). String objects have a built-in operation that uses the% operator to format strings. In practice, this looks like this:

**Using % formatting**

Here is the simplest form of % formatting

first\_name="Timothy"

print("my name is %s" % first\_name)

We can eve use more than  one variable, but they have to be closed in a brackets (tuple) as in the example below:

first\_name="Timothy"

surname="Mpofu"

Age=19

Level="NCV2"

#using %s to format output

print("My name is %s, and my surname is %s" % (first\_name, surname))

print("I am %s years old, and I am in %s" % (Age, Level))

**eLinks**

For more on formatting strings, refer to the Python Docs link:

<https://docs.python.org/3/library/stdtypes.html#printf-style-string-formatting>

**Using the str.format()**

We briefly introduced this in module 5 but did not go in depth. *str.format()* is an improvement on %-formatting. With *str.format(),* the replacement fields are marked by curly braces { }. Take a look at the code below which is similar to the one we did using the % formatting.

first\_name="Timothy"

surname="Mpofu"

Age=19

Level="NCV2"

#using str.format to format output

print("My name is {}, and my surname is {}." .format(first\_name, surname))

print("I am {} years old, and I am in {}." .format(Age,Level))

You can use str.format to refer to variables in any order by referencing their index:

first\_name="Timothy"

surname="Mpofu"

Age=19

Level="NCV2"

#using str.format

print("My name is {2}, and my surname is {1}." .format(Age, surname, first\_name,Level))

first\_name is at index 2 and surname is at index 1.

Although str.format() code is much more readable than %-formatting code, it can still be quite verbose when dealing with multiple parameters and longer strings. A newer way which addresses the weaknesses of % and *str,format*() is the f string.

**Using f string**

f-strings are string literals that begin with a f and contain curly braces that contain expressions that will be replaced with their values. The f string syntax is similar to  the *str.format()* syntax but less verbose. Take a look at how easy this is to read:

first\_name="Timothy"

surname="Mpofu"

Age=19

Level="NCV2"

#using f-string to format output

print(f"My name is {first\_name}, and my surname is {surname}." )

Moving forward, the *f*-string is the most recent technique for formatting output and we recommend you to make use of it.

**datetime module**

continue from the first paragraph on page 179.

**Activity 7.1**

**[Add this as the second question]**

2. Consider the program below that asks the user to enter their name and their age and print out a message addressed to them that tells them the year that they will turn 100 years old.

name **=** input("What is your name: ")

age **=** int(input("How old are you: "))

year **=** 2014 **-** age **+** 100

**print**(name **+** ", you will be 100 years old in the year " **+** str(year))

Modify the solution above implementing use f-strings instead of the + operator to print the resulting output message.

(2)

3. Modify the solution to avoid explicitly writing out the year. Use the built-in Python datetime library to ensure that the code you write works throughout the year, not just the current one.

(5)

# Module 9: Repetition control structure

Practical Activity 9.1 [Add this as the second question pg221]

2. Write a for loop that iterates through a string and prints every letter in uppercase for example the word “Technical Programming”

The output will be Technical Programming each letter in one line

A

F

R

I

C

A

(5)

3. Consider the code below

#declare variable to store number of iterations

n = 10

#condition variable

while n <= 100:

    #print outout

    print(n ,end = ",")

    n = n+10

The output of the program is supposed to be:

10,20,30,40,50,60,70,80,90,100,

However, the program is not stopping. Correct the code so that it will print 10 numbers (multiples of 10)

(2)

**Total :17 marks**

# Module 10: Modularisation and functions

**SUMMATIVE ASSESSMENT [Add this as question 8 of the summative assessment on page 264]**

8. Write a Python program to create a basic calculator using functions. The program must allow the user to enter two numbers from the keyboard and perform any of the following functions:

+ Addition

* + Subtraction

\* Multiplication

/ Division

If the user input is not a number, the program must alert the user and show an error message such as: " The value entered was not a number". If the user makes an incorrect operator, the program must display an error message: "Invalid operator type entry".

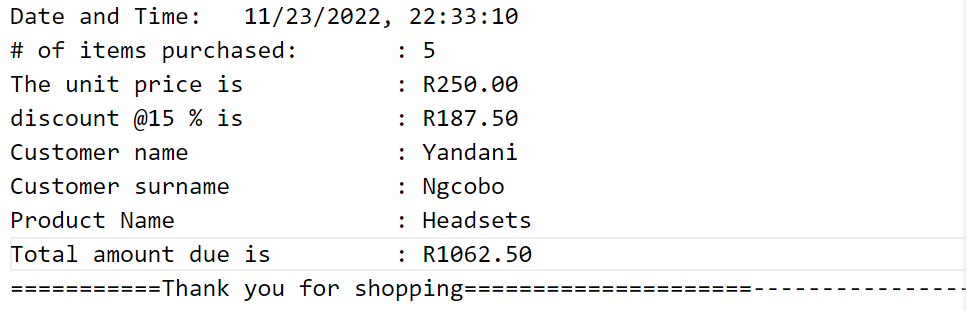
Use try… except block to catch any errors.

Use the f string for displaying output.

**(20)**

**9.** The Osmans Supply Store supplies different products to the public. Customers can buy different products on cash or on credit for their purchases. Customers who pay cash receive a discount, which is determined by the owner and differs from customer to customer. The discount should never exceed 25%. Design a Python program that will calculate the amount payable by customers for their purchases. The program should allow user to enter the type of purchase by entering CS for Cash Sales or CR for Credit purchases. The program should allow the entry in any case ( lowercase or uppercase).

If the discount entered is greater than 25%, the program should display an error message. The program should also display the system's current date and time. The program must generate a receipt, which is displayed in a text file. The output sample of the text file should be as follows:



Below is an IPO-chart for the problem:

|  |  |  |
| --- | --- | --- |
| **INPUT** | **Processing** | **Output** |
| productid  productName  quantity  unitPrice  Credit Sales  CustomerName  CustomerSurname  Cash Sales:  CustomerID  CustomerName  CustomerSurname  discount | CreditSales:  amountPayable=quantity \* unitPrice  CashSales:  amountPayable=quantity \* unitPrice  discountt=amountPayable-discount/100 | CreditSales  Cash Sale  Quantity  unitPrice  customerName  customerSurname  Product Name  amountPayable  Cash Sale  Quantity  unitPrice  discount  customerName  customerSurname  Product Name  amountPayable |

**(25)**

**Total : 67 Marks**