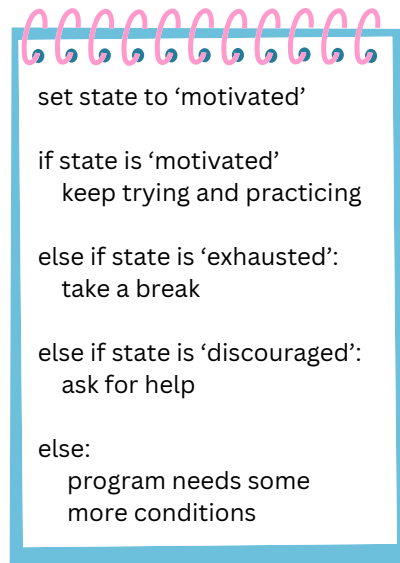


## Introduction

Programs begin with pen, paper, knowledge, good logic, open-minded thinking and the willingness to learn from mistakes and errors, **not** by writing a program without any planning

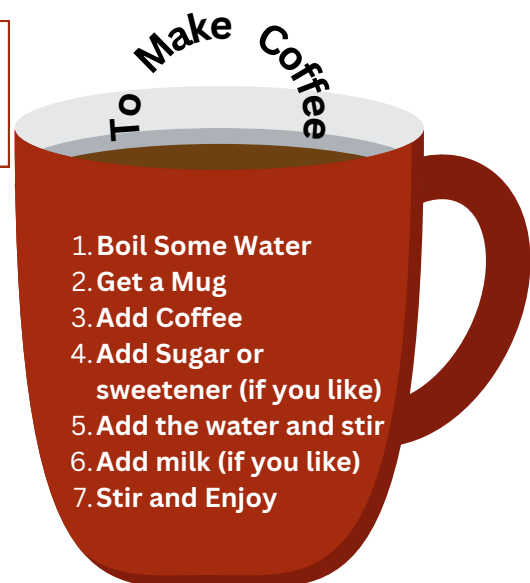
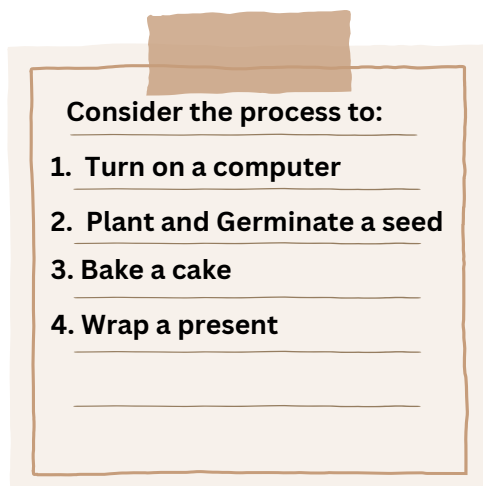
Also remember that a program's functionality and purpose often needs to be explained to clients with little to no understanding of any programming languages or concepts



To create a program, we spend time understanding the program's purpose and the problem our program will solve (the requirements of the program).

We want to create an **Algorithm** (a term that almost everyone has heard, most not in Software or Data Science Fields). An **Algorithm** is simply a set of instructions or steps that are designed and then followed to accomplish some task.

Algorithms are not only used in Software Development! We have seen and used Algorithms for most of our lives





Successful Programs can accomplish tasks from printing “Hello World!” to a console window to running the controls of an aeroplane to managing data for a company

With a powerful computer and the creativity of the programmer, we can make truly powerful and useful software applications

To create a program, we usually want to discuss and create a series of steps for the program to follow that will allow it to perform a task. In other words, we will design an **algorithm** that is implemented by the program.

## Pseudo Code

To create an Algorithm we want to make use of Pseudo Code. Pseudo Code is a loosely written series of steps that does not formally link to a programming language. This means we do not write it using code, but rather in short statements that can be understood by everyone

In our discussion of Pseudo Code, we will look at:

1. Guidelines for Pseudo Code
2. Pseudo Code Examples
3. Pseudo Code vs Programming Syntax
4. Flowcharts

### Guidelines for Pseudo Code

Whilst Pseudo Code is an informal way to define the logic and steps of an Algorithm, there are a few guidelines to be followed allowing the Pseudo Code to be easily understood and read by others.

1. Use one line to describe each major step
2. Keep the sentences simple
3. Use common words to describe specific aspects of a step such as:
  - ‘input’ when taking input from a user
  - ‘store’ when storing a value in a variable
  - ‘print’ when printing the results back to the user
4. Make use of indentation to show sub-steps (seen in the examples)

● **The best way to learn to code, is to write the code. When writing programs, always remember that code is often read more than it is written** ●

**NOTE:** Some common programming structures such as **if-statements**, **loops**, **variables** and others will be used in these examples. Basic information to describe them will be given, but they are evaluated formally in later notes



### 1. Write a series of steps to determine if a number is even

#### Thoughts:

If we spend some time thinking about even numbers, we can come up with some properties for them such as all even numbers have a final digit as either **0, 2, 4, 6** or **8**. This is useful, but requires us to constantly look at the last digit (and in some cases even isolate this last digit) so we wonder if there may be an easier way to determine if a number is even...

There is one: **All even numbers are divisible by 2**. This means we can divide the number by 2 and if there is no remainder after the division, then we have an even number

#### The Pseudo Code:

##### An if-statement

This purposeful extra spacing is called an **indentation**. It is compulsory in some languages and here it shows us that the printing of the 'odd number' message to the user only happens if the line above is True (correct)

```
request input number from user

store input number into a variable called 'user_input'

if 'user_input' can be divided by 2 without a remainder
    print out 'number is even'

else variable cannot be divided by 2
    print out 'number is odd'
```

Please note that this is **Pseudo Code**. Whilst it has some reserved keywords that are used in many languages such as **'if'**, this is not actual Programming Language Syntax. However, this is the entire point as the steps should be simple enough to understand without any knowledge of programming

## 2. Write an Algorithm to retrieve four preferred movie genres from an application user and then print these genres to the user

### Thoughts:

- We are going to need to ask the user to repeatedly enter different movie genres
- We need a way to count each entered genre until we reach a total of four
- We need a way to stop our algorithm from asking for more genres after four has been entered
- We need to be able to store the entered genres

We can create a 'counter' variable to keep track of the number of entered genres

We can use a **\*list** to store the genres

\* The note-set is considered for Python Developers and thus a list has been mentioned (a data structure used in the Python Language), in other languages an **array** could be used or other types of structures depending on how the data will be used

\* In addition, the word 'list' may be easier for non-programmers to understand which is the main consideration for Pseudo Code (language independent)

### The Pseudo Code:

**while** is a keyword indicating a **while-loop**. The idea is perform some action continuously and after each iteration (loop), check if the condition has been met. If not, then continue to perform the action

Notice that these three lines all have the same indentation, this indicates they are all part of our while statement

create and set 'entered\_genre\_count' to 0

**while** user has not entered four movie genres:

request input from user asking for a movie genre

add 1 to the number of movies entered

store the entered genre in list of genres

print all movie genres in list to the user

### Defensive Programming

**Defensive Programming** is a crucial principle for all software developers to understand. It may be early to discuss it now, but our above example gives us a fantastic opportunity to at least understand the basics of this principle and see the manner in which we can implement it.

Programs deal with inputs and data in a simple sense, but these inputs and the data are often messy; retrieved in incorrect formats or sometimes are simply wrong. Consider asking a user for a number but a letter is received! Programs that do not handle such mistakes (errors) will terminate or produce incorrect and often confusing outputs. Defensive Programming can assist us here.

**Defensive Programming** means to incorporate protective steps into an Algorithm and Program that allow it to handle incorrect inputs amongst other things.

5

In our above Algorithm, our solution is not well protected against bad inputs. We do not need actual, syntactically correct code to see this. We can modify our Pseudo Code to better protect our solution.

There are more protective steps that are needed, but here we will just focus on two to demonstrate **Defensive Programming**

1. Add a check that an input has been received
2. Check that an entered genre is different to the ones already entered in the list

```
create and set 'entered_genre_count' to 0

while user has not entered four movie genres:
    request input from user asking for a movie genre
    store user input in a variable 'current_genre'

    if current_genre does not have any information
        print message to user that a movie genre was not entered

    if a movie genre was retrieved from the user

        if current_genre is different from those in the list
            add 1 to the number of movies entered
            store the entered genre in list of genres

        if the current genre is the same as one in the list
            print message to user that the genre they have entered is already in the list

print all movie genres in list to the user
```

**Bold Text** = New Steps

Plain Text = Same Steps from Above

By incorporating **Defensive Programming** into our Algorithm (represented by Pseudo Code), our solution will now only add a movie genre to the list if it is different from those already in the list and if a movie (or at least some input) was actually received from the user

## Pseudo Code vs Programming Syntax

6

This note set focuses on Pseudo Code and thus will not fully examine the syntax of actual code (specifically Python for this section), but it can be helpful to see the two side-by-side to get a sense for how Pseudo Code is actually implemented

**Consider the first Pseudo Code example in which an algorithm to determine if a number was even had been created**

### **Pseudo Code:**

```
request input number from user

store input number into a variable called 'user_input'

if 'user_input' can be divided by 2 without a remainder
    print out 'number is even'

else variable cannot be divided by 2
    print out 'number is odd'
```

### **Python Code:**

```
user_input = input("Please enter a Number:\n")

if int(user_input) % 2 == 0:
    print("number is even")

else:
    print("number is odd")
```

**Take Note** that Defensive Programming has not been used in this example

Don't worry if the above Python Code is confusing at this stage, the main goal is to see how Pseudo Code steps can be applied into an actual Programming language


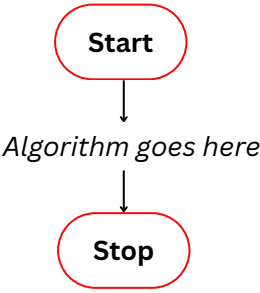

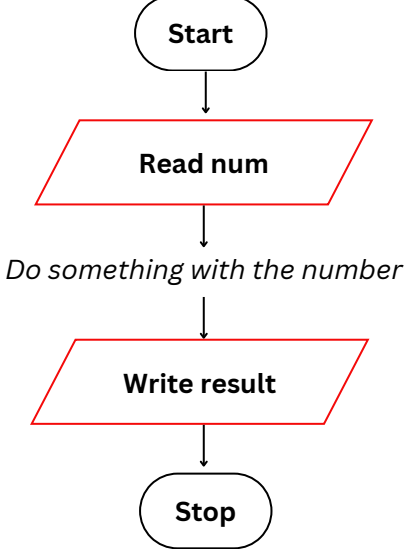

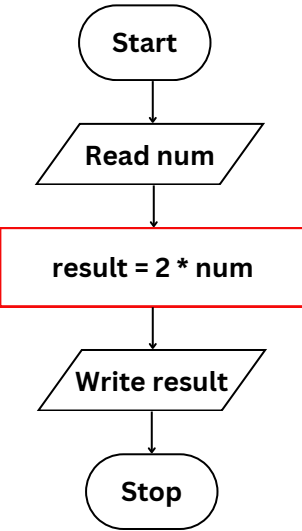
## Flowcharts

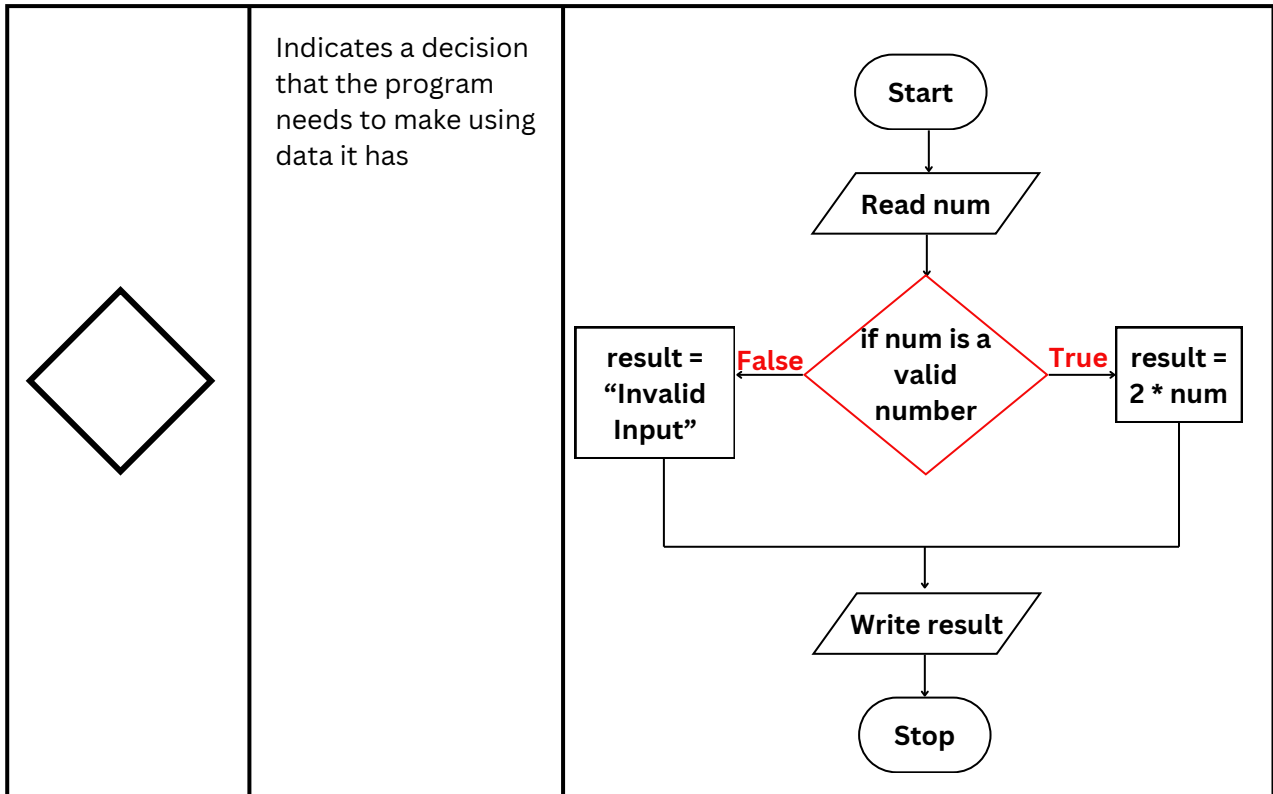
**Pseudo Code** has been demonstrated as a way to create and write a solution that accomplishes a task following a set of steps (an **Algorithm**) in a way that can be understood by everyone. Sometimes, it can also be helpful to show these steps in a more *visual* manner. A visual diagram always allows us to explain a concept or solution in an easy-to-understand and 'follow-along' style.

For Pseudo Code, we can make use of **Flowcharts**. A Flowchart is a diagram that uses symbols (containing boxes linked with arrows) with specific meanings to visually demonstrate how the various steps in our instruction set work together and what function they perform.

Like most things in Programming, it is best to see them in action to understand how they work

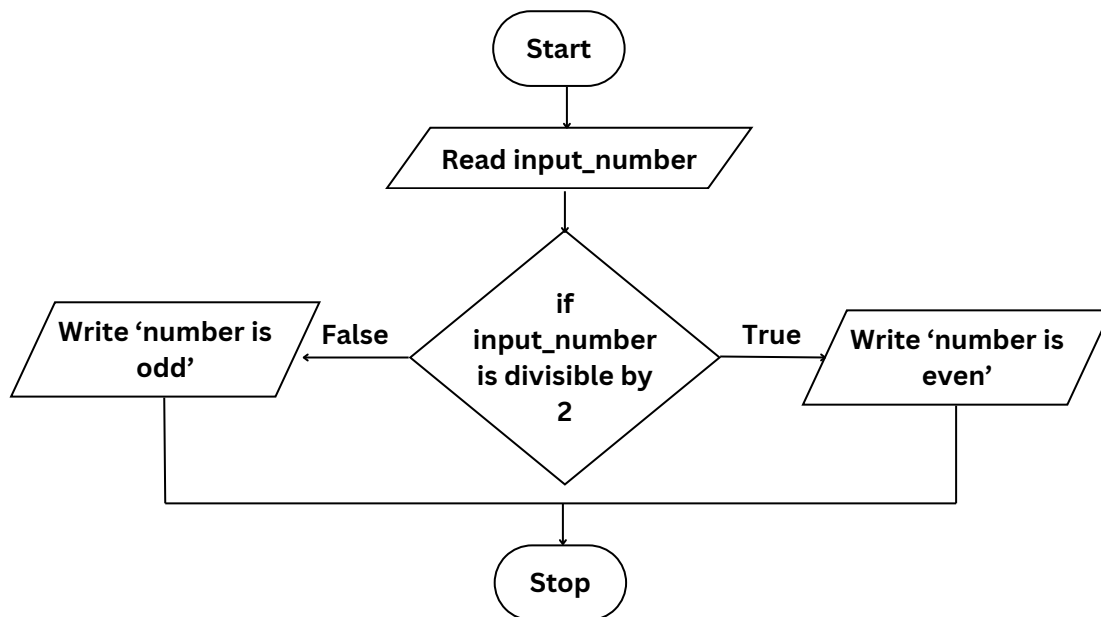
## FlowChart Symbols

Symbol	Explanation	Example
	Indicates the start and end of an algorithm and would contain the words "Start" and "Stop"	 <pre> graph TD     Start([Start]) --&gt; Process[Algorithm goes here]     Process --&gt; Stop([Stop]) </pre>
	Indicates an input to the program or an output from the program	 <pre> graph TD     Start([Start]) --&gt; Read[/Read num/]     Read --&gt; Process[Do something with the number]     Process --&gt; Write[/Write result/]     Write --&gt; Stop([Stop]) </pre>
	Indicates a calculation or process that must be completed	 <pre> graph TD     Start([Start]) --&gt; Read[/Read num/]     Read --&gt; Process[result = 2 * num]     Process --&gt; Write[/Write result/]     Write --&gt; Stop([Stop]) </pre>



### Flowchart Example

Consider the previous example in which we created an Algorithm to determine if a number was even or not (odd). Here we will create a flowchart for this example to add a visual representation of the solution





## More on Flowcharts

Flowcharts are useful tools for breaking down and designing the steps that an Algorithm will follow. Considering they use Pseudo Code, we note their range of uses extends far beyond that of Software Development and they can be applied to many fields.

Larger Software Projects can be broken down into smaller, simpler components and steps. Generally we create a series of these components that (with good design) will 'fit' together to create the entire application. **Flowcharts** can be used at the design phase of a project to help us understand and highlight major features of an application in a very simplified sense.

Consider a simple to-do list project, allowing a user to create and save their tasks in a database. We can design a flowchart for this during the design phase of the project. There are many other features we would want to add on such as functionality to view recorded tasks, to edit a task and to mark tasks as completed. For this example we will just focus on a basic solution as seen below

