



**SKILLS CITY**

Fair Access to Technology Futures

**Welcome**

**IN4.0<sup>TM</sup>**  
Group



**SKILLS CITY**

Fair Access to Technology Futures

# Introduction to Software Engineering

**IN4.0<sup>TM</sup>**  
Group

# Learning Objectives

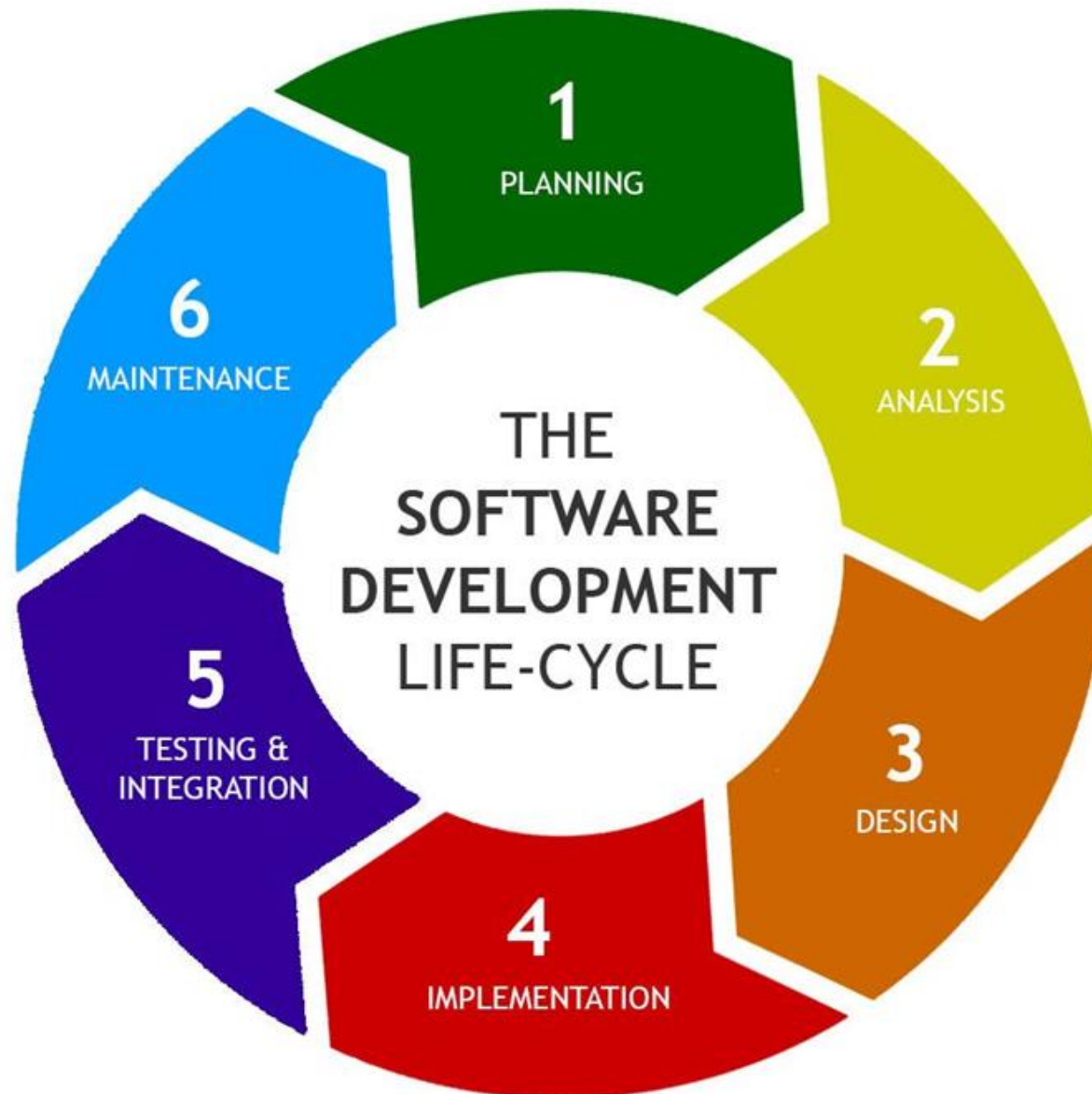
- To understand the purpose of planning when coding.
- To understand and be able to implement a basic flowchart.
- To understand and be able to write basic pseudocode.



**SKILLS CITY**

Fair Access to Technology Futures

# Software Development Lifecycle



# What is Software Engineering?

*"Software engineering is the process of analysing user needs and designing, constructing, and testing end-user applications that will satisfy these needs through the use of software programming languages. It is the application of engineering principles to software development"*

*"In contrast to simple programming, software engineering is used for larger and more complex software systems, which are used as critical systems for businesses and organizations."*

Techopedia Definition [What is Software Engineering? - Definition from Techopedia](#)



**SKILLS CITY**

Fair Access to Technology Futures

# Think like a Programmer

“Everyone in this country should learn to program a computer, because it teaches you to think.” — **Steve Jobs**

<https://www.freecodecamp.org/news/how-to-think-like-a-programmer-lessons-in-problem-solving-d1d8bf1de7d2/>

# Problem Solving

Are you able to grasp the concepts of programming, but find it difficult to solve problems in programming? This is perfectly normal!

Reading Article:

[The Beginner Programmer's guide to Problem Solving \[With Example\] - CodeProject](#)

# What Code is:

## Problem Solving



One line of code or a thousand lines of code, it's doing the same thing; solving problems!

Your problem-solving skills are essential for you to build solutions to the coding problems.



# What Code is NOT:

Easy to learn



Code isn't one size fits all. There will be different ways to solve a problem

# Types of Programming Languages

# Types of Programming Languages

There are 3 main types of programming languages:

Machine Code

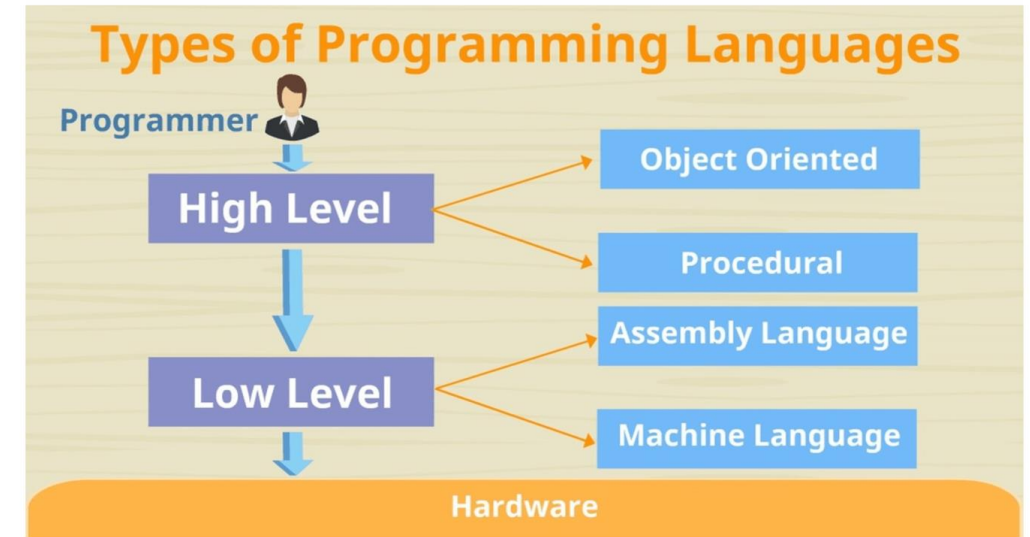
Assembly Language

High-level Languages:

Low-level  
Languages

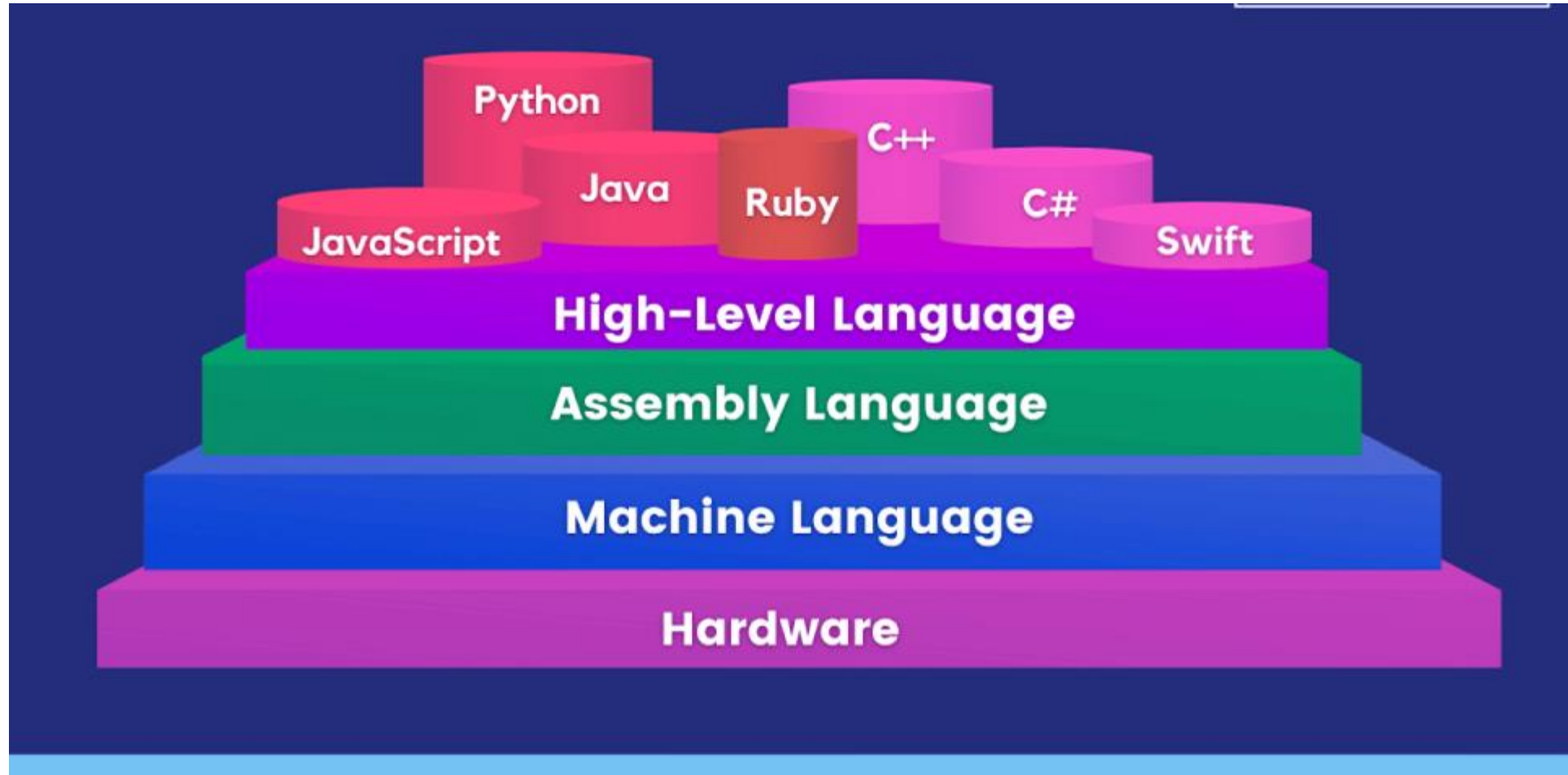
Python, JavaScript, Java, Ruby, C++, C, HTML, CSS  
GW Basic, Cobol, Ada , SQL, API

## High-Level vs Low-Level



**in**programmer

# Types of Programming Languages



# Low-level Programming Languages

## **Machine Code**

Programs written in 0s and 1s are written in Machine Code.

The processor of a computer can only work with binary digits( bits).

This means that all instructions given to a computer must, ultimately, be a pattern of 0s and 1s.

## **Assembly Language**

Machine code is too complex. This led to the development of Assembly Languages, which allowed programmers to use words instead of 0s and 1s.

There are situations when it is useful to use Assembly languages and one of the advantages is that it is executed faster than a higher-level language.

# High-level Programming Languages

Writing machine code and assembly language is too slow for the complex demands made on computers, therefore high-level languages were developed. They are programmer friendly, but that means they must be translated. High Level languages can either be compiled or interpreted.

## Interpreted –

When a programming language is **processed line by line**, it is often referred to as "interpreted." In an interpreted programming language, each line of code is executed one at a time by an interpreter, which reads the code, translates it into machine code or an intermediate representation, and then immediately executes it.

## Compiled –

A compiled language is a type of programming language in which the source code you write is translated into machine code or a lower-level intermediate code by a special program called a compiler. This machine code or intermediate code can be executed directly by the computer's CPU **without the need for an interpreter or further translation steps. All at once.**

# High-level Programming Languages

- Python
- JavaScript
- Ruby (Language) on Rails (Framework) = ROR
- C++
- C#
- Java

And many many more

# Algorithm vs Flowchart vs Pseudocode



# Algorithms

**Algorithms** are sets of step-by-step **instructions** for the computer to follow. They are at the heart of all computer **programs**.

You can think of an algorithm as similar to a food recipe. If you make a sandwich, you follow a set of steps to put the different ingredients together. You bring ingredients together, assemble them as you like, and produce a final product - the sandwich. If you were asked to write down instructions to make a sandwich, this would be considered a written algorithm.

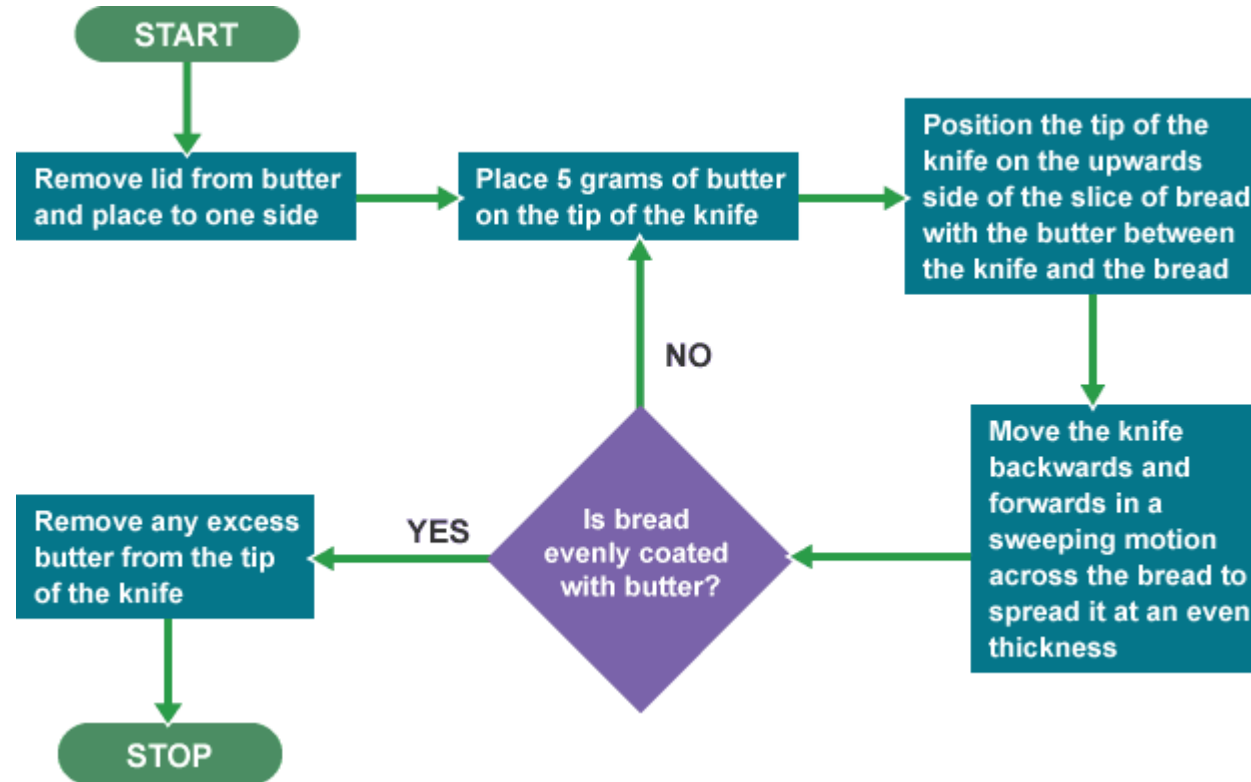
# Designing Algorithms

When designing an algorithm, you need to assess how complex it could be. With a food recipe, a simple command like 'spread butter on bread' could be made much more detailed.

## **For example:**

1. Remove lid from butter tub and place to one side.
2. Place 5 grams of butter on the tip of the knife.
3. Position the tip of the knife on the upwards side of the slice of bread with the butter between the knife and the bread.
4. Move the knife backwards and forwards in a sweeping motion across the bread to spread it at an even thickness.
5. Repeat steps 2 to 4 until one side of the slice of bread is evenly coated with butter.
6. Remove any excess butter from the tip of the knife.

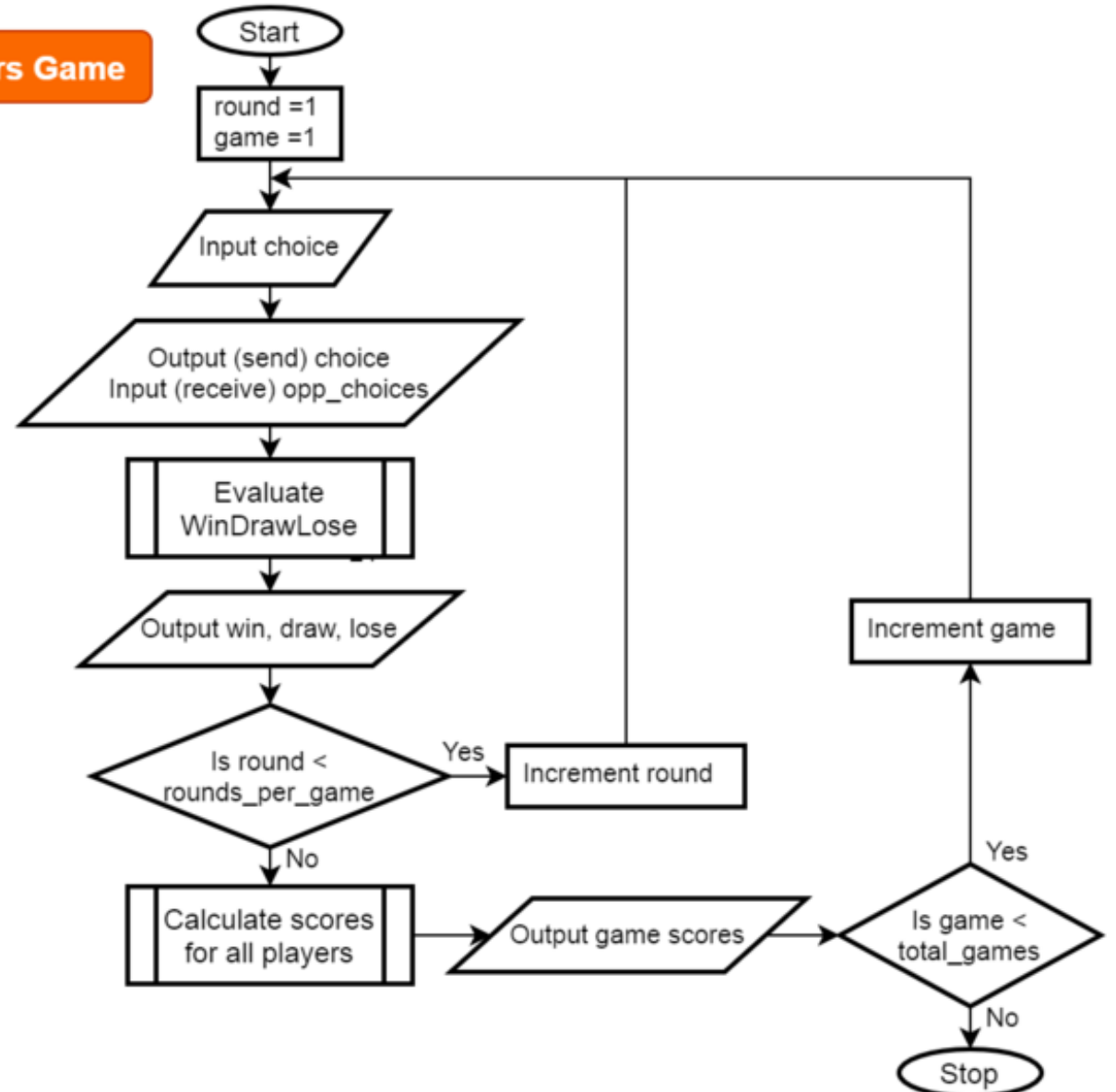
# Designing Algorithms



# Flowcharts

A **flowchart** is a diagram that represents a set of **instructions**. Flowcharts normally use standard symbols to represent the different types of instructions. These symbols are used to construct the flowchart and show the step-by-step solution to the problem.

## Rock, Paper, Scissors Game








# Flowcharts Symbols

A **flowchart** is a diagram that represents a set of **instructions**.

Flowcharts normally use standard symbols to represent the different types of instructions.

These symbols are used to construct the flowchart and show the step-by-step solution to the problem.

Symbol	Name	Function
	Start/End	Flowchart terminator
	Arrows	A line is a connector that represent relationship between the representative shapes and its flow direction
	Input/Output	A parallelogram represents input or output of your program
	Process	A rectangle represent a process
	Decision	A diamond indicates a decision

# Create a simple flowchart

**Really useful website for creating flowchart diagrams:**

<https://app.diagrams.net/>

# Activity 1 – Flowchart

Create a flowchart to represent an algorithm that asks the user their age and if they are 19 or over, they can enrol on to a course. If they are not, then they can't enrol.

# Pseudocode

**Pseudo** is from Greek '**pseudes**' meaning *the one that is not true*.

This is to say that Pseudocode is **not** a real programming language.

**You shouldn't really worry about Pseudocode, that's the idea behind it!**



**Rules**

It doesn't have a strict syntax or strict rules like programming languages, but it DOES need be LOGICAL.



**SKILLS CITY**

Fair Access to Technology Futures



# Common Pseudocode Notation

There is no strict set of standard notations for pseudocode, but some of the most widely recognised are:

- **INPUT** – indicates a user will be inputting something. Input age. Take age, Enter age
- **OUTPUT** – indicates that an output will appear on the screen
- **WHILE** – a loop (iteration) that has a condition at the beginning)
- **FOR** – a counting loop (iteration)
- **REPEAT – UNTIL** – a loop (iteration) that has a condition at the end
- **IF – THEN – ELSE** – a decision (selection) in which a choice is made
  - any instructions that occur inside a selection or iteration are usually indented



# Using Pseudocode

Pseudocode can be used to plan out programs. Planning a program that asks people what the best subject they take is, would look like this in pseudocode:

```
REPEAT
    OUTPUT 'What is the best subject you take?'
    INPUT user inputs the best subject they take
    STORE the user's input in the answer variable
    IF answer = 'Computer Science' THEN
        OUTPUT 'Of course it is!'
    ELSE
        OUTPUT 'Try again!'
UNTIL answer = 'Computer Science'
```

# How to write Pseudocode

**Write an algorithm using pseudocode that calculates the amount of fuel a train will need to complete a journey. The algorithm must :**

1. Ask the user how many kilometres the journey will be = INPUT
2. Only continue if the user enters a value greater than zero = CONDITION
3. Set the amount of fuel to a number 100 times greater than the number of kilometres = MATH / Logic
4. Not allow the amount of fuel to be less than 1500 = Condition / Logic
5. Finally display the amount of fuel needed = OUTPUT - Solution

```
amountOfFuel = 0
```

```
Kilometres = INPUT ("How many kilometres?") = 100
```

```
IF Kilometres > 0 THEN
```

```
    amountOfFuel = 100 x Kilometres = 100 * 100 = 10000
```

```
    IF amountOfFuel < 1500 THEN
```

```
        amountOfFuel = 1500
```

```
    ENDIF
```

```
    OUTPUT amountOfFuel
```

```
ENDIF
```

# Activity 2 – Pseudocode

Write an algorithm using pseudocode that asks the user their age. If they are aged 19 or over, then output a message that says: “You can enrol on this course” . If they are not, output a message that says: “You are not old enough to enrol”

# Tools Setup -Replit

Before you leave today's session, please ensure you are setup to start coding.

Replit is an online IDE ( Integrated Development Environment)

Please create an account at: <https://replit.com/>

Or

[Install VS Code](#)

# Tools Setup – VSCode

Before you leave today's session, please ensure you are setup to start coding.

VSCode is an IDE ( Integrated Development Environment)

IDEs are coding editors and interpreters - *combined together* - with lots of useful features for developers. There are so many IDEs out there, however, we have chosen VS Code for this course.