

Python



Chapter 1

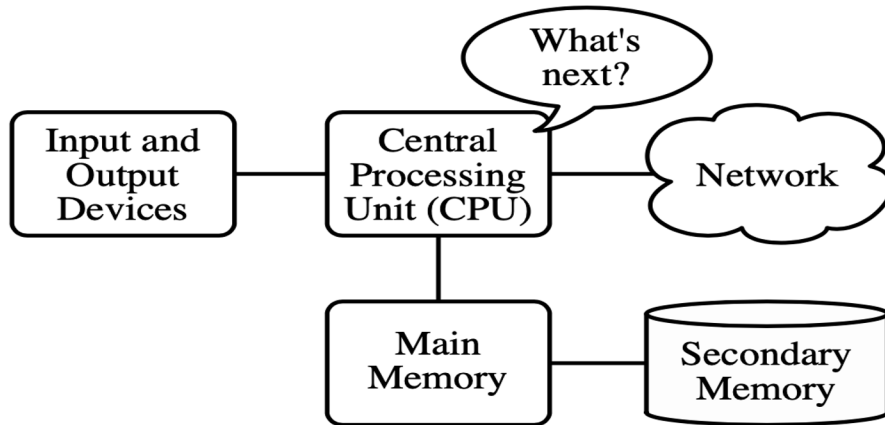



INTRODUCTION

1.1 Computer hardware architecture

How are computers built?

If you were to take apart your computer and look deep inside, you would find the following parts, referred to collectively as **hardware**::






Central Processing Unit (or CPU) also called “CPU” or “the processor”. The heart of any computer. It is built to constantly ask “What’s next?”, asking for the next instruction to execute and then doing whatever that instruction says. Also, it is what runs the software that we write.

Main Memory used to store information that the CPU needs in a hurry. The main memory is nearly as fast as the CPU. The information stored in the main memory disappears when the computer is turned off.

Secondary Memory also used to store information, but it is much slower than the main memory. The advantage of the secondary memory is that it can store information even when there is no power to the computer. Examples of secondary memory are disk drives or flash memory (typically found in USB sticks and cell phones).



Input and Output Devices - An input device is **something you connect to a computer that sends information into** the computer. An output device is something you connect to a computer that has information sent to it (screen, keyboard, mouse, microphone, speaker, touchpad, etc.).

Network Connection describes the extensive process of connecting various parts of a network to one another, for example, through the use of routers, switches and gateways, and how that process works.

1.3 Understanding programming

Programming is a way to **instruct the computer to perform various tasks**.

Instruct the computer - this means that you provide the computer a set of instructions that are written in a language that the computer can understand.

The instructions could be of various types. For example:

- Adding 4 numbers,
- Rounding off a number, etc.

Computers understand instructions that are written in a specific form called a programming language.

Just like we humans can understand a few languages (English, French, Mandarin, etc.), so is the same case with computers.

Perform various tasks: the tasks could be simple ones like we discussed above (adding 4 numbers, rounding off a number) or more complex which may involve a sequence of multiple instructions. For example:

- Calculating simple interest, given principal, rate and time.
- Calculating the average return on a stock over the last 5 years.

The above two examples require complex calculations. They cannot usually be solved in simple instructions like adding 4 numbers.

In summary, Programming is a way **to tell computers to do a specific task.**

1.4 Words and sentences

The Python language reserves a set of **keywords** that designate special language functionality. No object can have the same name as a reserved word. In Python 3.6, there are 33 reserved keywords:

False	class	from	or
None	continue	global	pass
True	def	if	raise
and	del	import	return
as	elif	in	try
assert	else	is	while
async	except	lambda	with
await	finally	nonlocal	yield
break	for	not	

When you type **help('keywords')** to the Python interpreter you can see a list with reserved keywords. Reserved words are case-sensitive and must be used exactly as shown. They are all entirely lowercase, except for **False**, **None**, and **True**.

To create a sentence you have to write a function **print** followed by a string of text that you want to print. Text has to be enclosed in single quotes.

```
print('Hello World!')  
  
print("What's your name?")
```

The strings in the print statements are enclosed in quotes. Single quotes and double quotes do the same thing; most people use single quotes except in cases like this where a single quote (which is also an apostrophe) appears in the string.

1.5 Conversing with Python

To start programing in Python you must first install the software on your computer and learn how to start Python on your computer.

In the internet you can easy find free resources (tutorials/guides) how to install Python software.

Below, I enclose link to website that help you with that task.

- <https://realpython.com/installing-python/>

Also, there is also available online compiler:

- <https://www.programiz.com/python-programming/online-compiler/>

1.6 Terminology: Interpreter and compiler

Python is a **high-level** language designed to be relatively straightforward for humans to read and write and for computers to read and process.

Other high-level languages include: Java, C++, PHP, Ruby, Basic, Perl, JavaScript, and many more.

Hardware inside the Central Processing Unit (CPU) does not understand any of these high-level languages.

The CPU understands a language we call **machine language**. Machine language is very simple and frankly very tiresome to write because it is represented all in zeros and ones:

```
001010001110100100101010000001111  
11100110000011101010010101101101
```

Interpret - it means to execute a program in a high-level language by translating it one line at a time.

Compile - it means To translate a program written in a high-level language into a low-level language all at once, in preparation for later execution.

1.7 What is a program?

The definition of a **program** at its most basic is a sequence of Python statements that have been crafted to do something. Even simple one-line script is a program. It is not particularly useful, but in the strictest definition, it is a Python program.

It might be easiest to understand what a program is by thinking about a problem that a program might be built to solve, and then looking at a program that would solve that problem.

There are some low-level conceptual patterns that we use to construct programs. These constructs are part of every programming language from machine language up to the high-level languages.

input Get data from the “outside world”. This might be reading data from a file, or other sources. Frequently, input will come from the user typing data on the keyboard.

output Display the results of the program on a screen or store them in a file or perhaps write them to a device like a speaker to play music or speak text.

sequential execution Perform statements one after another in the order they are encountered in the script.

conditional execution Check for certain conditions and then execute or skip a sequence of statements.

repeated execution Perform some set of statements repeatedly, usually with some variation.

reuse Write a set of instructions once and give them a name and then reuse those instructions as needed throughout your program.

When programs become increasingly complex, you will encounter three general types of errors:

- **Syntax errors** usually the easiest to spot, syntax errors occur when you make a typo. **Not ending an if statement with the colon** is an example of an syntax error, as is misspelling a Python keyword (e.g. using while instead of while). Syntax errors usually appear at compile time and are reported by the interpreter
- **Logic errors** also called **semantic errors** logical errors cause the program to behave incorrectly, but they do not usually crash the program. Unlike a program with syntax errors, a program with logic errors can be run, but it does not operate as intended.

Logic errors can be caused by the programmer:

- incorrectly using logical operators, eg expecting a program to stop when the value of a variable reaches 5, but using `<5` instead of `<=5`
- incorrectly using **Boolean operators**
- unintentionally creating a situation where an **infinite loop** may occur
- incorrectly using brackets in calculations
- unintentionally using the same variable name at different points in the program for different purposes

- **Runtime error** is an error that takes place during the running of a program.

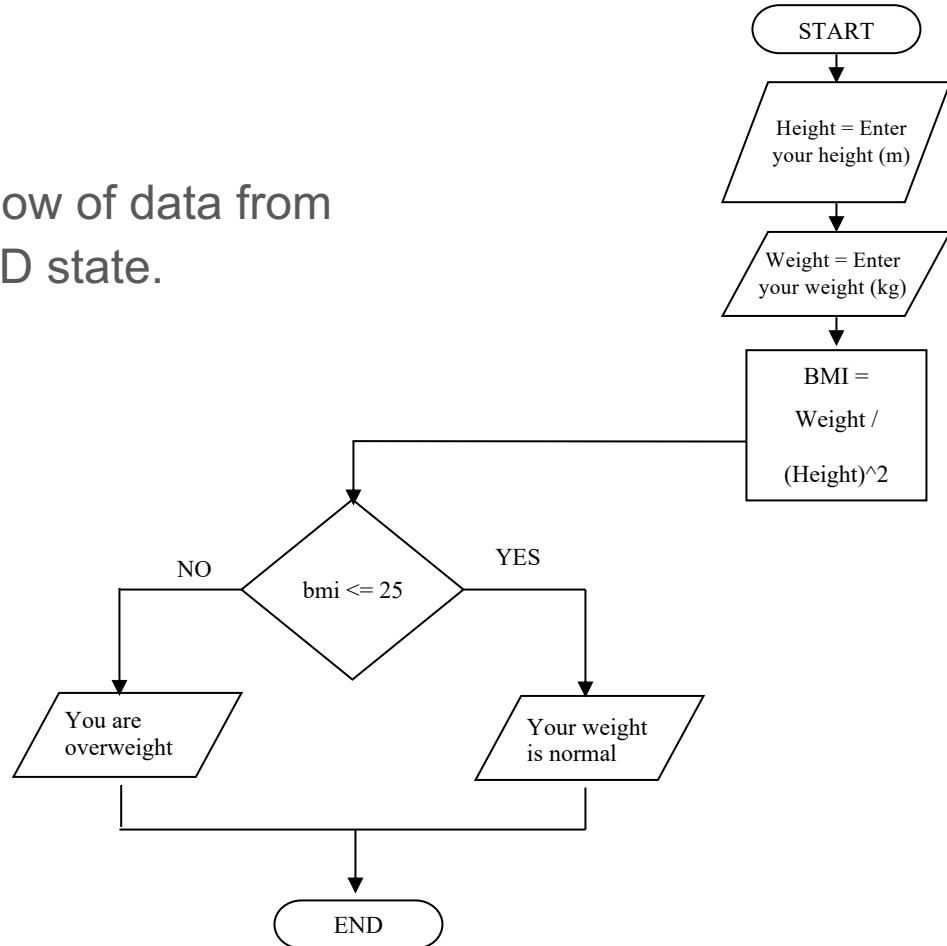
An example is writing a program that tries to access the sixth item in an **array** that only contains five items. A runtime error is likely to crash the program

Algorithm and Flowcharts

- An algorithm is a step-by-step instruction of solving a problem. An algorithm has to be:
 - Unambiguous
 - Terminating
 - Executable
- A flowchart is a visual representation of an algorithm.
 - Only simple executable statement are allowed in a flowchart.

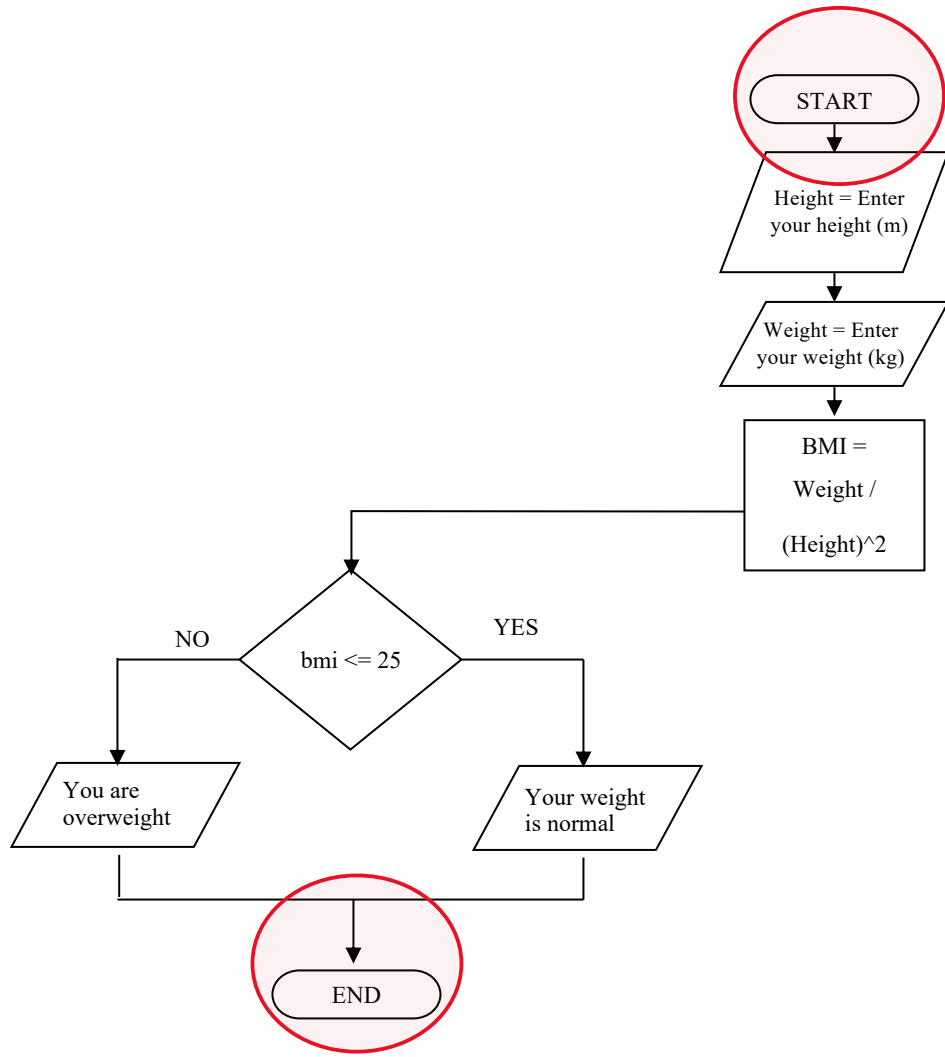
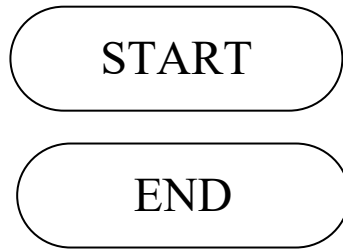
Flowcharts

- A flowchart represent the flow of data from the START state to the END state.



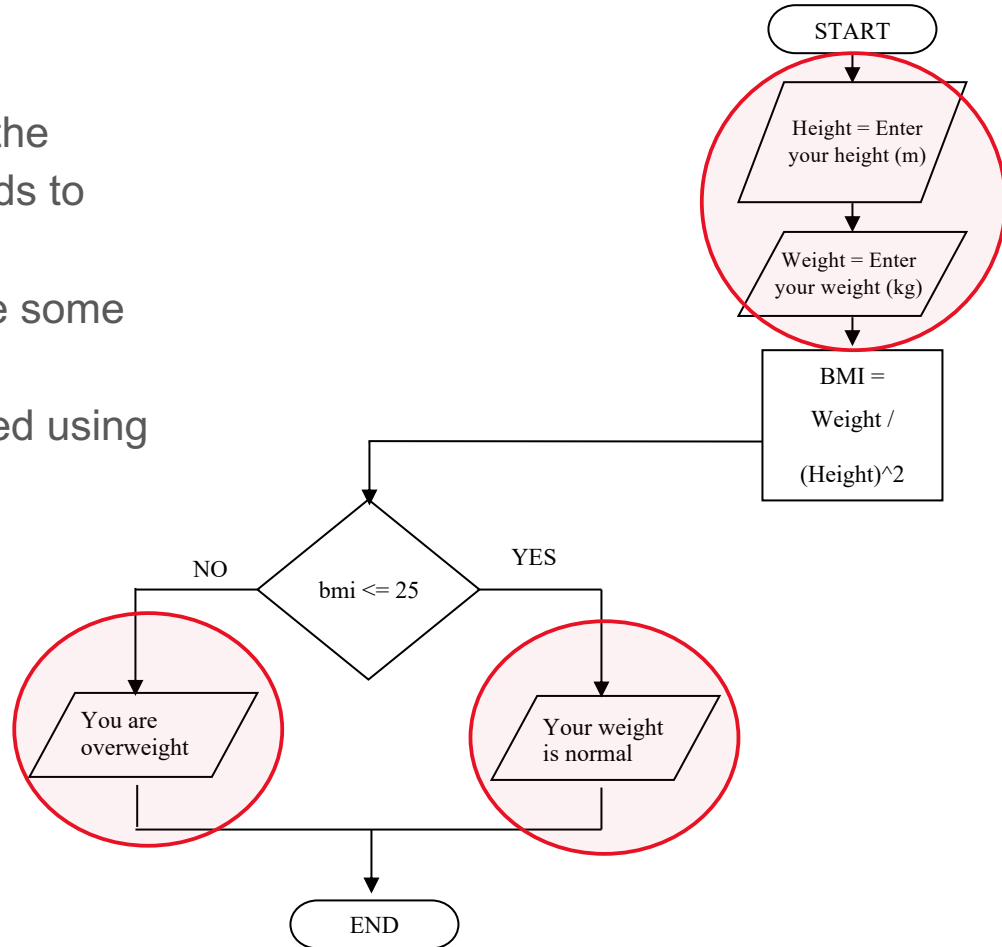
Start and End

- Start and End: Terminals
 - Start and End are the terminal states.
 - There is only and only one START state.
 - There is at least one End state.
 - START and END symbols are shown using a Circle or rounded rectangle.



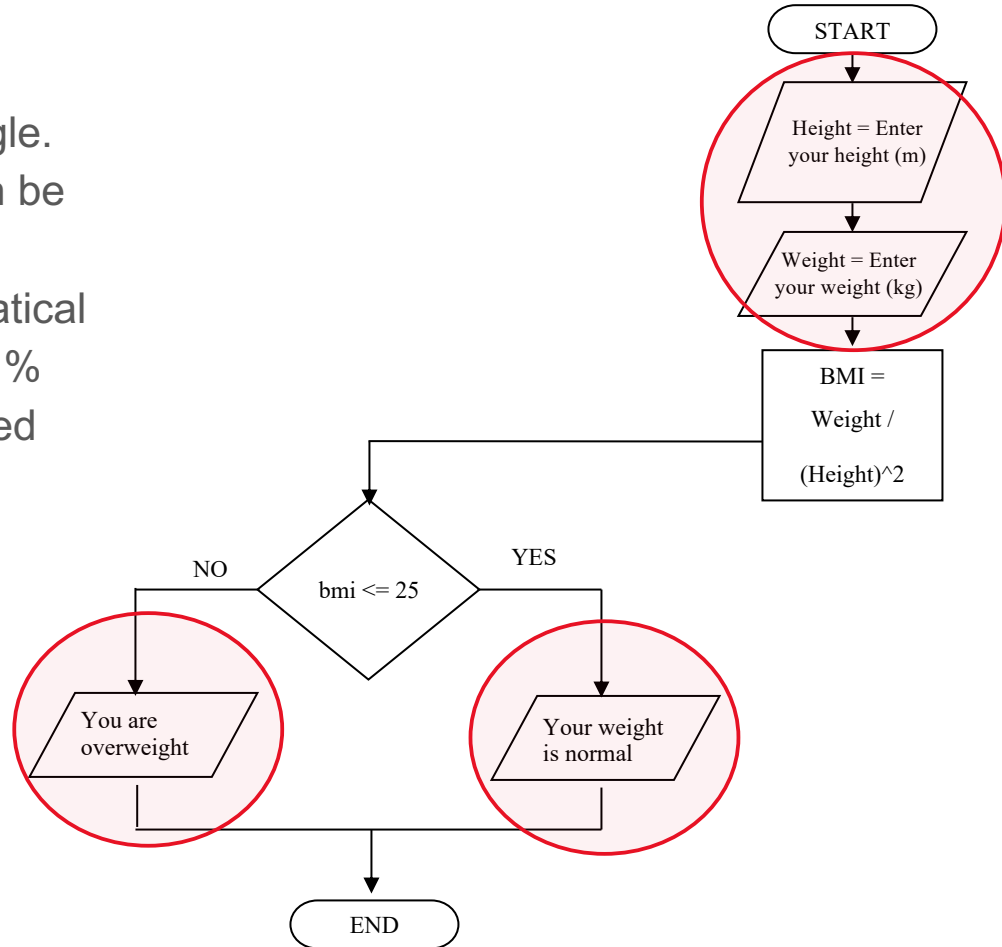
Input and Output

- The **input** is used to receive the information the flowchart needs to generate the answer.
- The **output** is used to provide some information to the user.
- The input/output is represented using a parallelogram.



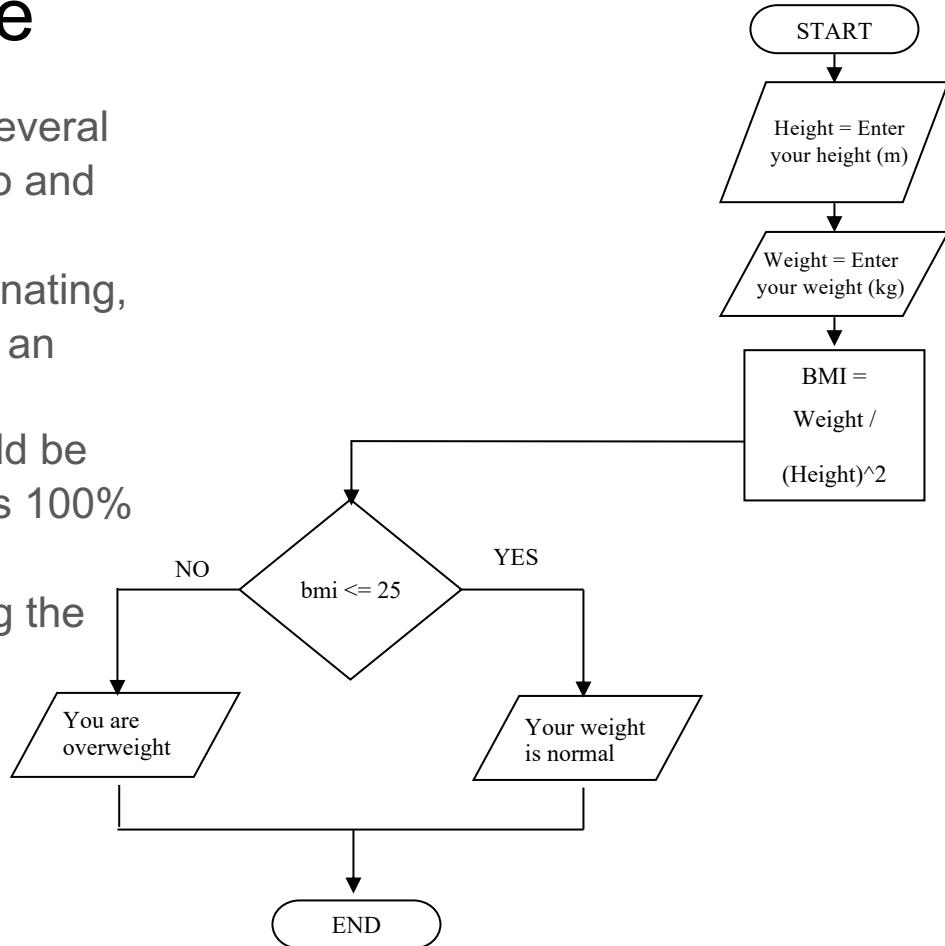
Process

- Represented using a Rectangle.
- Simple operation/process can be done in one step
- Process is usually a mathematical operation such as $+$, $-$, $*$, $/$ and $\%$
- Process cannot be complicated



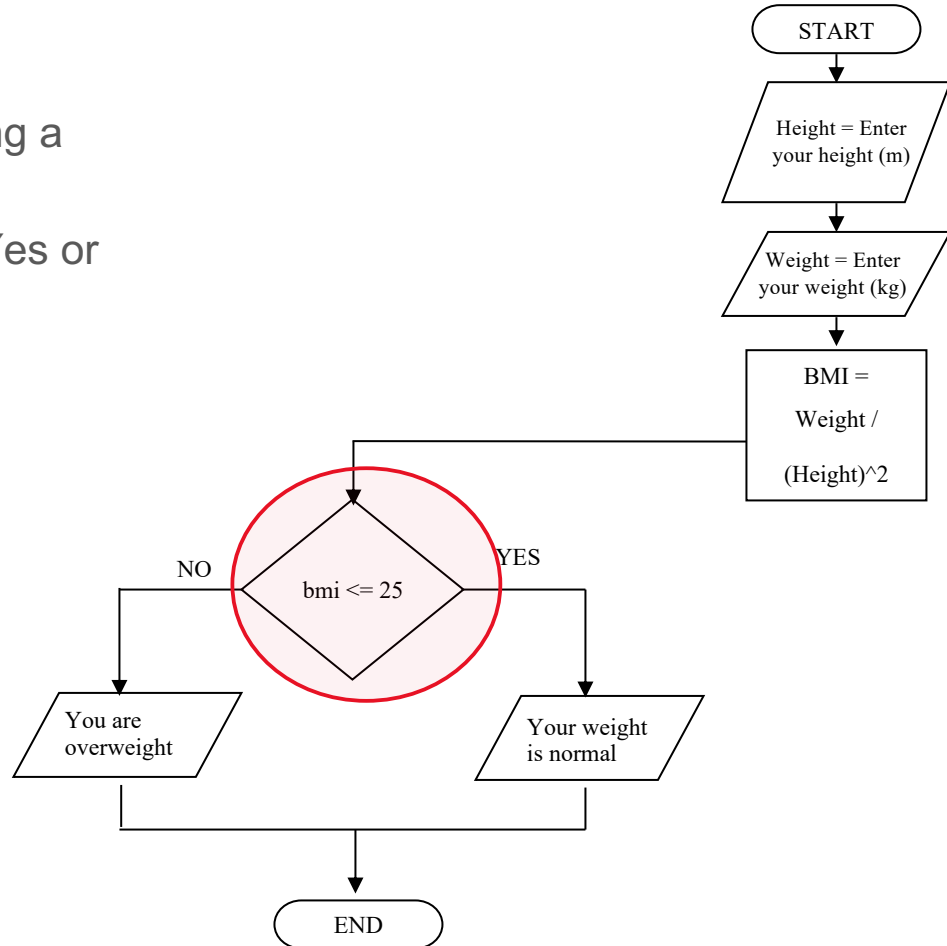
Sequence Structure

- A flowchart is a sequence of several steps starting from a START to and END.
- The sequence should be terminating, meaning that it has to result in an End state.
- The steps in a sequence should be deterministic, meaning that it is 100% clear what the next step is.
- Sequence is represented using the arrows.



Decision Structure

- A Decision is represented using a Diamond.
- The Decision is evaluated to Yes or No



Repetition Structure

- A repetition structure creates a loop in your program.
- There is a Decision structure always involved in a Repetition structure.
- The Repetition / Iteration is carried out until the condition is true.
- A wrong Repetition structure could result in a situation called: “Infinite loop”.

