Module 2 : Basic Applications of Programming in IoT

Presenter: Assc. Prof. Ts. Dr. Ahmad Shukri Mohd Noor

MODULE OUTLINE

2.1

MODULE 2.1

Introduction to Programming Languages

MODULE 2.2

Introduction to Python

2.2

2.3

MODULE 2.3

Variables and Data Types in Programming

MODULE 2.4

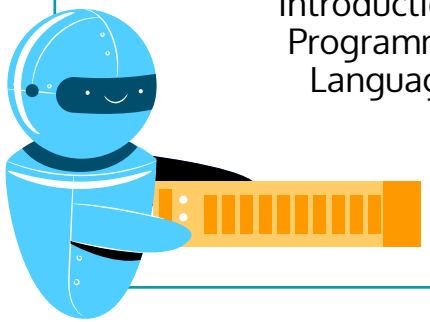
Basic Operators in Programming

2.4

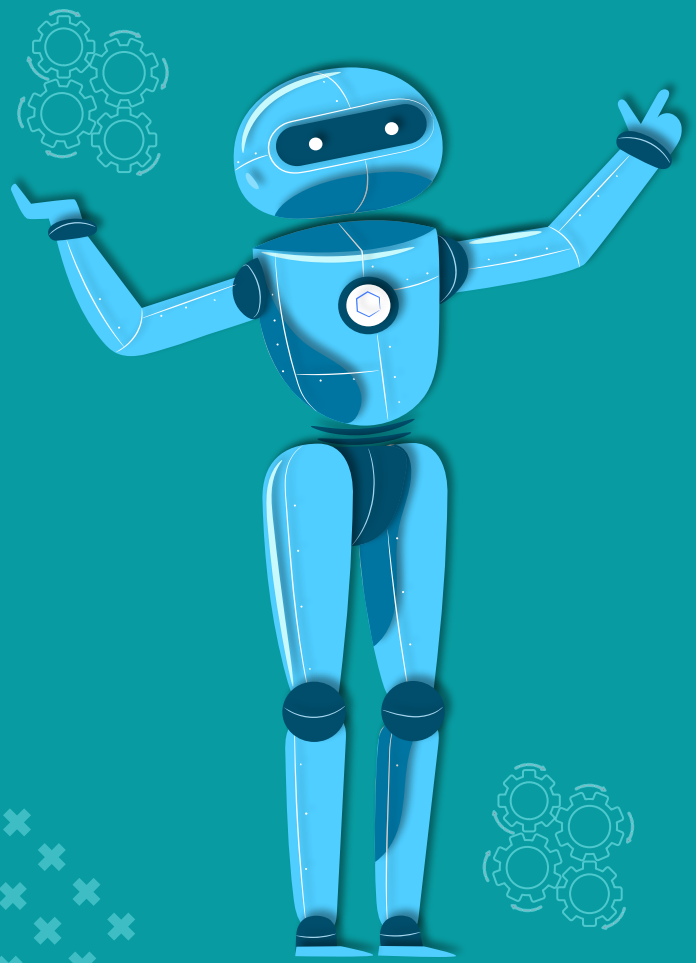
2.5

MODULE 2.5

Control Structures in Programming



2.1



Introduction to Programming Languages

- What is Programming?
- Types of Programming Languages
- The Difference Between Different Programming Languages
- The Natural Language of the Computer

2.1 Introduction to Programming Languages

2.1.1 What is Programming?

- A simple answer would be, “Programming is the act of instructing computers to carry out tasks.” It is often referred to as coding.
- So then, what is a computer program? A computer program is a sequence of instructions that the computer executes.
- Computer in the definition above is any device that is capable of processing code.
- This could be smartphones, ATMs, the Raspberry Pi, Servers to name a few.

Source: https://www.freecodecamp.org/news/a-gentler-introduction-to-programming-1f57383a1b2c/

2.1 Introduction to Programming Languages

2.1.1 What is Programming?

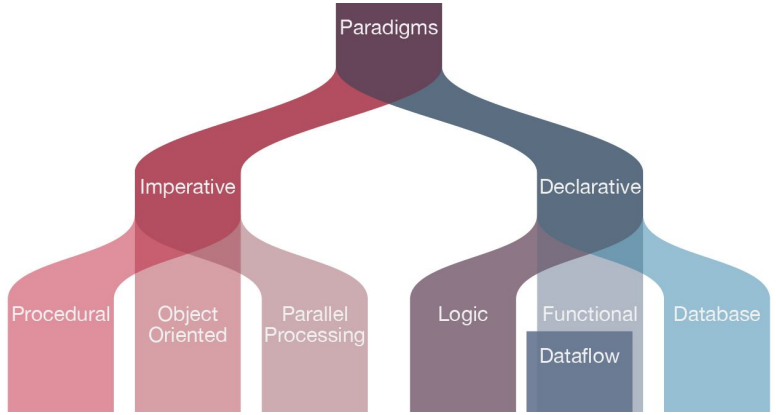
A Good Analogy for Programming

- Firstly, our daily lives are patterned. The world works in a predictable manner; for instance — day and night, seasons, sunrise, and sunset. People go through routines like rising early in the morning, going to school or going to work. We are getting orders from other people at work, including our supervisors. One can explain how we cook such recipes in finite steps.
- Second, there's some code running in the background each time we use smart devices. It may seem like a easy task to move a mouse pointer from one section of your computer screen to the other, but in fact so many lines of code just ran. An act as easy as typing letters into Google Docs leads to the execution of lines of code in the background. It's code all over the place.

2.1 Introduction to Programming Languages

2.1.2 Types of Programming Languages

- Procedural Programming Language
- Functional Programming Language
- Object-oriented Programming Language
- Logic Programming Language



2.1 Introduction to Programming Languages

2.1.2 Types of Programming Languages

Procedural Programming Language

- Used to execute a sequence of statements which lead to a result.
- This type uses multiple variables, heavy loops and other elements, which separates them from functional programming languages.
- Functions of procedural language may control variables, other than function's value returns.
- For example, printing out information.

2.1 Introduction to Programming Languages

2.1.2 Types of Programming Languages

Functional Programming Language

- Uses usually stored data, thus avoiding repeated task loops.
- Functional programming's primary emphasis is on return values of functions, and side effects and specific suggest that storing state is strongly discouraged.
- For example, in an extremely pure useful language, if a function is named, the function is supposed not to modify or perform any output. However, it can create algorithmic calls and alter these calls' parameters.
- Functional languages are generally simpler and easier to work out on abstract problems, but their programming model can also be "further from the machine," making it impossible to know exactly, but the code is decoded into the machine language (which is also troublesome for system programming).

2.1 Introduction to Programming Languages

2.1.2 Types of Programming Languages

Object-oriented Programming Language

- This programming language views the world as a group of objects that have internal data and external accessing parts of that data.
- The aim this programming language is to think about the fault by separating it into a collection of objects that offer services which can be used to solve a specific problem.
- One of the main principle of object oriented programming language is encapsulation that everything an object will need must be inside of the object.
- This language also emphasizes reusability through inheritance and the capacity to spread current implementations without having to change a great deal of code by using polymorphism.

2.1 Introduction to Programming Languages

2.1.2 Types of Programming Languages

Scripting Programming Language

- These programming languages are often procedural and may comprise object-oriented language elements, but they fall into their own category as they are normally not full-fledged programming languages with support for development of large systems.
- For example, they may not have compile-time type checking.
- Usually, these languages require tiny syntax to get started.

2.1 Introduction to Programming Languages

2.1.2 Types of Programming Languages

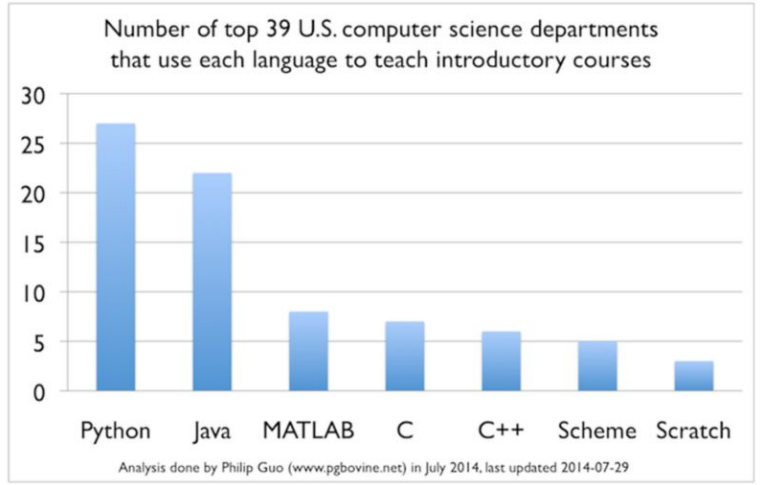
Logic Programming Language

- These types of languages let programmers make declarative statements and then allow the machine to reason about the consequences of those statements.
- In a sense, this language doesn't tell the computer how to do something, but employing restrictions on what it must consider doing.
- To call these groups " types of language " is really a bit confusing. It's easy to program in an object-oriented style in C language.
- In truth, most of the languages include ideas and features from various domains, which only helps to increase the usefulness of these types of languages.
- Nevertheless, most of the programming languages do not best in all styles of programming.

2.1 Introduction to Programming Languages

2.1.3 The Difference Between Different Programming Languages

- The programming languages are hundreds.
- They are rated by reputation, culture, support for the long term, pedagogy, business usage.
- They may also be classified by technicality, for example whether they are functional, imperative, static, heavy, or loosely typed.
- Some languages are more teachable than others. Some languages are intended for educational purposes and not for commercial use.
- For example, there are languages written so children can learn how to code.
- The languages are very strong and easy to set up and learn. One such programming language is Python.



2.1 Introduction to Programming Languages

2.1.3 The Difference Between Different Programming Languages

C++ Language

- The C++ language has an object oriented structure which is used in large projects.
- Programmers can collaborate one program into different parts or even one individual work on each part of the program.
- The structure of object oriented also permit code to be reused many times.
- This language is an efficient language. But, many programmers will disagree

2.1 Introduction to Programming Languages

2.1.3 The Difference Between Different Programming Languages

C Language

- The C language is a basic programming language and it is a very popular language, particularly used in game programming,
- Because C language includes the additional packing of the C++,
- Every programmer uses this language because it makes programs faster .
- However the value of this language gives the reusability of C++ to get the slight increase in performance with C language.

2.1 Introduction to Programming Languages

2.1.3 The Difference Between Different Programming Languages

JAVA Language

- The Java language is a multi platform language that's particularly helpful in networking.
- Of course, mostly this language is used on the web with Java applets.
- However, this language is used to design cross platform programs, Since it similar to C++ in structure and syntax.
- For C++ programmers, Java language is very easy to learn and it offers some advantages provided by object oriented programming.

2.1.3 The Difference Between Different Programming Languages

JAVA Language

- Like reusability and it can be difficult to write efficient code in Java.
- But, nowadays the speed of the Java language has increased and 1.5 version offers some good features for easy program making.

2.1 Introduction to Programming Languages

2.1.3 The Difference Between Different Programming Languages

PHP Language

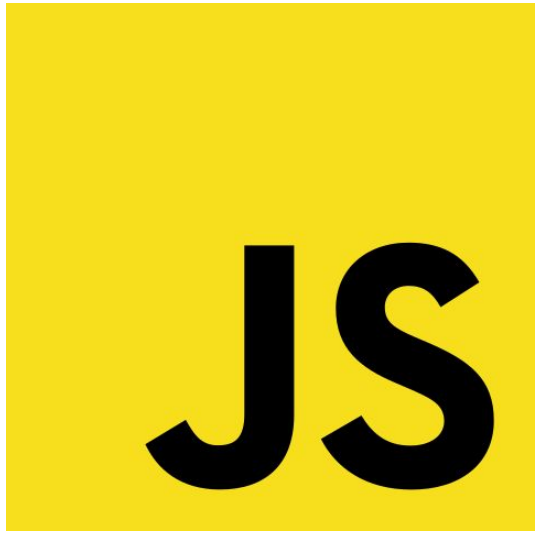
- The PHP language is used to design web pages and sometimes it is also used as scripting language.
- This language is designed to develop a rapid website, and as a result comprises features which make it easy generate HTTP headers and link to databases.
- As a scripting language, it includes a set of components permit the programmer to easily get up to speed.
- However, it has more sophisticated object oriented features.

2.1 Introduction to Programming Languages

2.1.3 The Difference Between Different Programming Languages

Javascript Language

- JavaScript is a very powerful client-side scripting language.
- JavaScript is used mainly for enhancing the interaction of a user with the webpage.
- In other words, you can make your webpage more lively and interactive, with the help of JavaScript.
- JavaScript is also being used widely in game development and Mobile application development.



2.1 Introduction to Programming Languages

2.1.3 The Difference Between Different Programming Languages

Javascript Language

- Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.
- Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together.
- Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance.

2.1 Introduction to Programming Languages

2.1.4 The Natural Language of the Computer

- Machines have their natural language like humans do. Computers do not understand the human language. The natural language of computers is the binary code — 1 and 0. These represent two states: on (1) and off (0).
- That is the natural language of electronic equipment. It would be hectic for us as humans to communicate with the computer in binary.

2.1 Introduction to Programming Languages

2.1.4 The Natural Language of the Computer

Enter Programming Languages

- To communicate with machines who speak binary, we do so in a language that's closer to our own natural language. Such as English, French, Swahili or Arabic. Programming languages are close to our natural languages. But they are more structured and must be thoroughly learned.

- They could be high level or low level languages. High level programming languages are farther away from the machine language than low level languages.

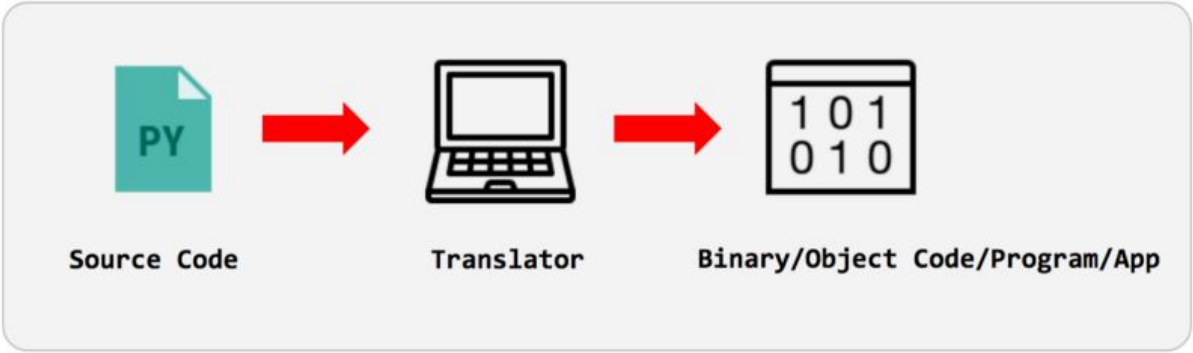
- The computer needs a way to understand our human language. To do this, we'll need a translator.

2.1 Introduction to Programming Languages

2.1.4 The Natural Language of the Computer

What are Translators

- Source code refers to code written in a particular programming language.
- Translators have the responsibility of converting your source code to the machine language. This is also known as binary. Remember ones and zeros. We may refer to the binaries as Object Code , the Program or a common word today: App.



2.1.4 The Natural Language of the Computer

- Translators can be any of:
  - Interpreters
  - Compilers
  - A hybrid of Interpreters and Compilers
  - Assemblers

2.1.4 The Natural Language of the Computer

Interpreters

- Some languages are interpreted. The translator processes the source code line by line and runs every line in the final program or app. This means that interpreted source code starts running until it encounters an error. Then the interpreter stops to report such errors.
- Python is a good example of an interpreted programming language.

2.1.4 The Natural Language of the Computer

Compilers

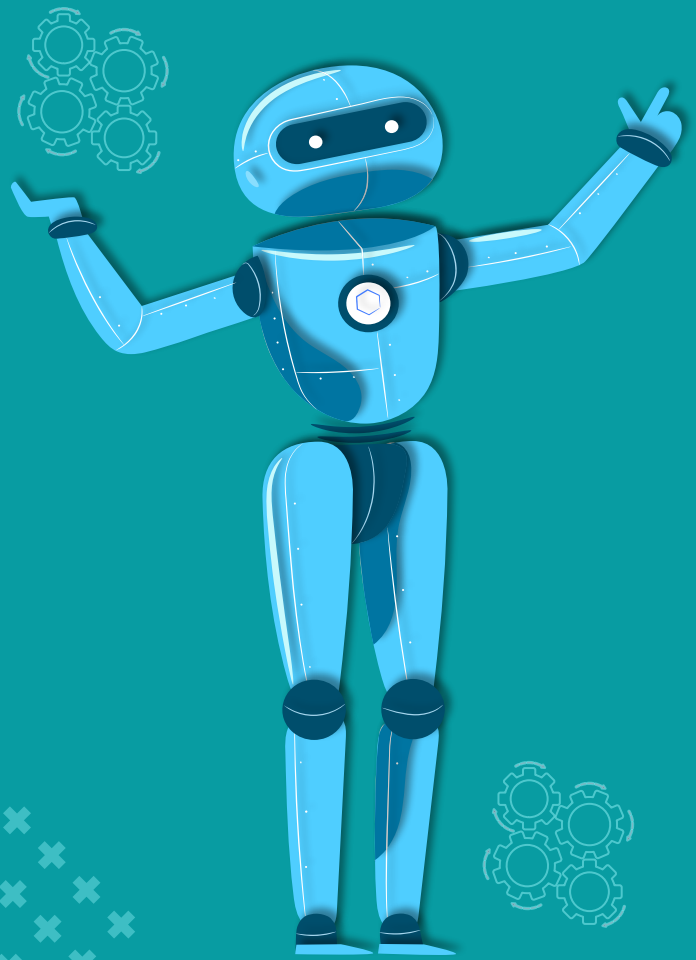
- Compilers function differently. They convert the source code in its entirety via a compilation process to binary. The binary is then executed. If there were errors in the source code, they are detected during the compilation time and flagged. This interrupts the compilation process, and no binary is generated.
- Interpreters translate line by line and execute the line before going on to the next line. Compilers translate all lines of a program to a file (binary) and execute the whole file.

2.1.4 The Natural Language of the Computer

Hybrid Translators

- A hybrid translator is a combination of the Interpreter and Compiler. A popular hybrid programming language is Java. Java first compiles your source code to an intermediate format known as the Bytecode.
- The Bytecode is then interpreted and executed by a runtime engine also known as a Virtual machine. This enables the hybrid translators to run the bytecode on various operating systems.

2.2



Introduction to Python

- Let's Python!
- Python Versions
- Python Development Environments
- Python Elements
- Python Comments

2.2 Introduction to Python

2.2.1 Let's Python!

Python

- Python is a general purpose scripting language that implements the imperative, object-oriented, and functional paradigms.
- Dynamic typing, automatic memory management, exceptions, large standard library, modular.
- Extensions can be written in C and C++
- Other language versions (Jython, IronPython) support extensions written in Java and .Net languages)
- Design philosophy: easy to read, easy to learn

2.2 Introduction to Python

2.2.1 Let's Python!

Why python?

- The world's fastest growing programing language
- Among software engineer, mathematicians, data analyst, scientist, accountants, network engineer and even KIDS.
- A multi-purpose language with a simple and beginner-friendly syntax.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.
- It is a high level language
- Cross-platform
- Huge community
- Large ecosystem

2.2 Introduction to Python

2.2.1 Let's Python!

What can python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

2.2 Introduction to Python

2.2.2 Python Versions

There are **TWO** versions right now

- Python 2: compatible with much existing software
- Python 3: a major redesign
  - Not backward compatible.
  - Most features are the same or similar, but a few will cause older programs to break.
  - Part of the Python philosophy –don't clutter up the language with outdated features

2.2 Introduction to Python

2.2.3 Python Development Environments

- PyDev with Eclipse
- PyCharm
- Komodo
- Emacs
- Vim
- TextMate
- Gedit
- Idle
- PIDA (Linux)(VIM Based)
- NotePad++ (Windows)
- BlueFish (Linux)

2.2 Introduction to Python

Lab Guide

1. Install python for windows

➔ https://docs.google.com/document/d/1Kr5uXCNE9vxY8bqJ1uZPwPPW9Ja1oiFMMZ_VNxG1zJA/edit

2. Use jupyter notebook to code python

➔ https://docs.google.com/document/d/1nGRgdZyAkEJMckepJANgArJqzCrtbEUIVWHBtoVWdoM/edit

3. Use uPyCraft ide for micropython

➔ https://docs.google.com/document/d/1Kr5uXCNE9vxY8bqJ1uZPwPPW9Ja1oiFMMZ_VNxG1zJA/edit

2.2 Introduction to Python

2.2.4 Python Elements

- Syntax
  - Python syntax can be executed by writing directly in the Command Line
- Identifiers
  - Must begin with letter or underscore, followed by any number of letters, digits, underscores

```
In [1]: print("Hello world")
        Hello world
```

2.2 Introduction to Python

2.2.4 Python Elements

- Python Indentation
  - Indentation refers to the spaces at the start of a line of code.
  - Whereas indentation in code is for readability only in other programming languages, the indentation in Python is very significant.
  - Python uses the indent to indicate a block of code.

```
In [2]: if 5>3:
        print("wrong indentation")

  File "<ipython-input-2-ed14205c5890>", line 2
    print("wrong indentation")
    ^
IndentationError: expected an indented block
```

```
In [3]: if 5>3:
            print("right indentation")

right indentation
```

2.2 Introduction to Python

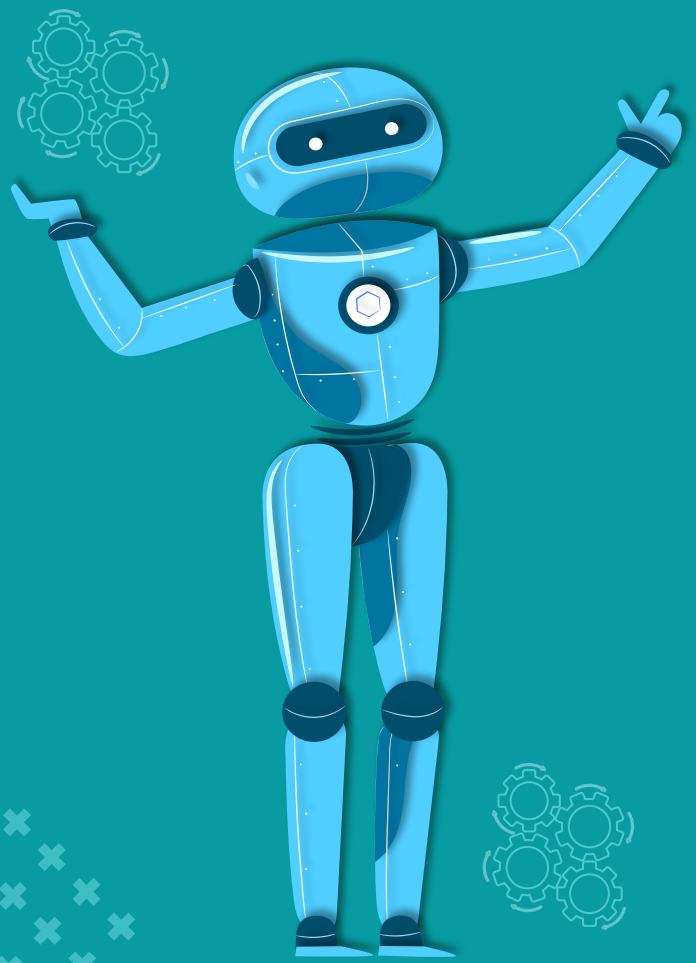
2.2.5 Python Comments

- Comments can be used to describe code in Python.
- Comments may be used to make the code readable.
- Comments may be used while checking code, to avoid execution.
- Comments start with a #, and will be ignored by Python.
- No, Python has no syntax for multi-line comments.

```
In [9]: # this is comment
        print("comment test")

        comment test
```

```
In [8]: # this is comment
        # python do not have multiline comment
        print("comment test")

        comment test
```

2.3



Variables and data types in programming

- Python Variables
- Python Built-in Data Types
- Python Input, Output and Import

2.3 Variables and data types in programming

2.3.1 Python Variables

- Variables are the containers where data values are stored.
- A variable is created the moment of first assign value to it.
- A name of a variable has to start with a letter or an underscore.
- A name variable can contain only alpha-numeric characters and underscores (A-z, 0-9, and).
- The names of the variables are case sensitive.

```
In [11]: a = 5
         b = 6.0098
         _car = 'BMW' # can use '' or " "

         print(_car)
         print(b)
         print(a)
```

```
BMW
6.0098
5
```

2.3 Variables and data types in programming

2.3.1 Python Variables

- Python allow us to assign value to multiple variables in single line.

```
In [13]: #value assigned to multiple variables

         car1,car2,car3 = "BMW",'Mercedes',"Volkswagen"
         print(car1)
         print(car2)
         print(car3)
```

```
BMW
Mercedes
Volkswagen
```

- Python helps one to assign several variables in single line with the same value.

```
In [16]: #same value assigned to multiple variables

         car1=car2=car3 = "Viva"
         print(car1)
         print(car2)
         print(car3)
```

```
Viva
Viva
Viva
```

2.3 Variables and data types in programming

2.3.1 Python Variables

- Often, the Python print statement is used to display variables.
- Python uses the + character to combine both text and a variable.
- The + character functions as a mathematical operator for numbers.
- Python will send you an error if you try to combine a string and a number.

```
In [18]: nama = 'Dan'
         print('My name is ' + nama)

My name is Dan
```

```
In [19]: nama = 'Dan'
         ayat = 'My name is '
         print( ayat + nama)

My name is Dan
```

```
In [20]: number1 = 20
         number2 = 2020
         print(number1 + number2)

2040
```

2.3 Variables and data types in programming

2.3.1 Python Variables

- Variables created outside of a function are called global variables.
- Everybody may use global variables, both within and outside of the functions.
- If there is the same variable name as the global and local variable, the local variable may only be used within the function while the original global variable remains

```
In [22]: location = 'Sungai Petani'

         def function1():
             print(location)

         function1()
```

```
Sungai Petani
```

```
In [23]: #global variable
         location = 'Sungai Petani'

         def function1():
             #local variable
             location = 'Jitra'
             print(location)

         function1()
         print(location)
```

```
Jitra
Sungai Petani
```

2.3 Variables and data types in programming

2.3.1 Python Variables

- You may use the Global Keyword to create a global variable within a function.
- Also, if you want to modify a global variable within a function, use the global keyword

```
In [25]: #global variable
location = 'Sungai Petani'

def function1():
    #global variable created
    #inside function
    global location

    location = 'Jitra'
    print(location)

function1()
print(location)
```

```
Jitra
Jitra
```

2.3 Variables and data types in programming

2.3.2 Python Built-in Data Types

- Data type is an important concept in programming.
- Variables may store different types of data, and different types of data can do different things.
- Python has built-in data forms in these categories, by default:

Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`

2.3 Variables and data types in programming

2.3.2 Python Built-in Data Types

- Lists and tuples store, in a particular order, one or more items or values.
- The objects stored in a list or tuple may be of any form including the None Keyword specified form of nothing.
- In most cases the lists and tuples are identical but there are some differences

```
In [39]: e = ["wij","dan","mohamad"]
         print(e)
         print(type(e))

['wij', 'dan', 'mohamad']
<class 'list'>
```

```
In [40]: f = ("wij","dan","mohamad")
         print(f)
         print(type(f))

('wij', 'dan', 'mohamad')
<class 'tuple'>
```

2.3 Variables and data types in programming

2.3.2 Python Built-in Data Types

- The literal syntax of tuples is shown by parentheses () whereas the literal syntax of lists is shown by square brackets [] .
- Lists has variable length, tuple has fixed length.
- List has mutable nature, tuple has immutable nature.
- List has more functionality than the tuple.

```
In [41]: e = ["wij","dan","mohamad"]
         print(e)

         e[2] = "ariff"
         print(e)
```

```
['wij', 'dan', 'mohamad']
['wij', 'dan', 'ariff']
```

```
In [42]: f = ("wij","dan","mohamad")
         print(f)
         f[2] = ariff
         print(f)
```

```
('wij', 'dan', 'mohamad')

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-42-07d2fee7e8f1> in <module>
      1 f = ("wij","dan","mohamad")
      2 print(f)
----> 3 f[2] = ariff
      4 print(f)

NameError: name 'ariff' is not defined
```

2.3 Variables and data types in programming

2.3.2 Python Built-in Data Types

- Lists has more builtin function than that of tuple. We can use dir([object]) inbuilt function to get all the associated functions for list and tuple.

```
In [43]: dir(e)
Out[43]: ['__add__',
          '__class__',
          '__contains__',
          '__delattr__',
          '__delitem__',
          '__dir__',
          '__doc__',
          '__eq__',
          '__format__',
          '__ge__',
          '__getattribute__',
          '__getitem__',
          '__gt__',
          '__hash__',
          '__iadd__',
          '__imul__',
          '__init__',
          '__init_subclass__',
          '__iter__',
          '__le__',
          '__len__',
          '__lt__',
          '__mul__',
          '__ne__',
          '__new__',
          '__reduce__',
          '__reduce_ex__',
          '__repr__',
          '__reversed__',
          '__rmul__',
          '__setattr__',
          '__setitem__',
          '__sizeof__',
          '__str__',
          '__subclasshook__',
          'append',
          'clear',
          'copy',
```

```
In [44]: dir(f)
Out[44]: ['__add__',
          '__class__',
          '__contains__',
          '__delattr__',
          '__dir__',
          '__doc__',
          '__eq__',
          '__format__',
          '__ge__',
          '__getattribute__',
          '__getitem__',
          '__getnewargs__',
          '__gt__',
          '__hash__',
          '__init__',
          '__init_subclass__',
          '__iter__',
          '__le__',
          '__len__',
          '__lt__',
          '__mul__',
          '__ne__',
          '__new__',
          '__reduce__',
          '__reduce_ex__',
          '__repr__',
          '__rmul__',
          '__setattr__',
          '__sizeof__',
          '__str__',
          '__subclasshook__',
          'count',
          'index']
```

2.3 Variables and data types in programming

2.3.2 Python Built-in Data Types

- Tuples operation has smaller size than that of list, which makes it a bit faster.

```
In [45]: e = ["wij","dan","mohamad"]
         f = ("wij","dan","mohamad")

         print(e.__sizeof__())
         print(f.__sizeof__())
```

```
64
48
```

2.3 Variables and data types in programming

2.3.2 Python Built-in Data Types

- Dictionary is an unordered collection of key-value pairs.
- It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value.
- Each item being a pair in the form key:value.
- Key and value can be of any type.

```
In [51]: h = {'name': 'wijdan', 'age': 20}

         print("his name is",h['name'])

         print("his age is",h['age'])
```

```
his name is wijdan
his age is 20
```

2.3 Variables and data types in programming

2.3.2 Python Built-in Data Types

- Set is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }. Items in a set are not ordered.
- We can perform set operations like union, intersection on two sets. Set have unique values. They eliminate duplicates.
- Since, set are unordered collection, indexing has no meaning. Hence the slicing operator [] does not work.

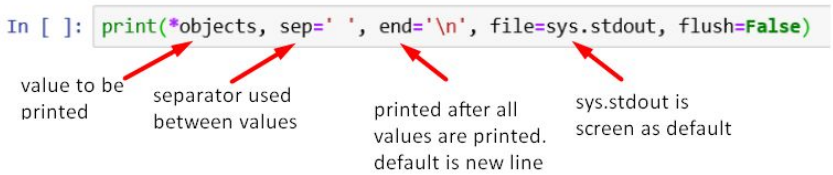
```
In [53]: i = {'dan', 'dan', 'dan', 'wij', 'mohamad'}
         print(i)
         print(i[1])

         {'dan', 'wij', 'mohamad'}
         ---------------------------------------------------------------------------
         TypeError                                 Traceback (most recent call last)
         <ipython-input-53-1bf0bce0a741> in <module>
               1 i = {'dan', 'dan', 'dan', 'wij', 'mohamad'}
               2 print(i)
         ----> 3 print(i[1])

         TypeError: 'set' object does not support indexing
```

2.3 Variables and data types in programming

2.3.3 Python Input, Output and Import

- We use the print() function to output data to the standard output device (screen).
- We can also output data to a file.
- The actual syntax of the print() function is

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```




```
In [1]: print(1,2,3,4)
        print(1,2,3,4, sep='#', end='.')

        1 2 3 4
        1#2#3#4.
```

2.3 Variables and data types in programming

2.3.3 Python Input, Output and Import

- Sometimes we would like to format our output to make it look attractive.
- This can be done by using the str.format() method. This method is visible to any string object.

```
In [2]: x = 10
        y = 2020

        print("I am {} years old in {}".format(x,y))
```

```
I am 10 years old in 2020
```

```
In [3]: print("i love {0} and {1}".format("roti canai","teh tarik"))
        print("i love {1} and {0}".format("roti canai","teh tarik"))
```

```
i love roti canai and teh tarik
i love teh tarik and roti canai
```

2.3 Variables and data types in programming

2.3.3 Python Input, Output and Import

- To allow flexibility we might want to take the input from the user. In Python, we have the input() function to allow this. The syntax for input() is

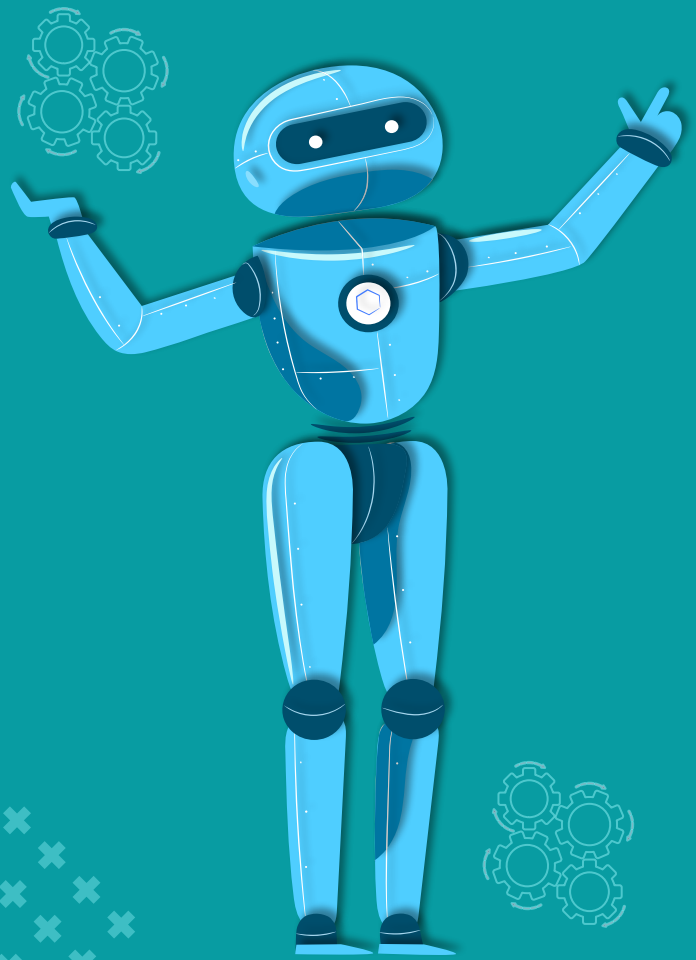
```
input([prompt])
```

- It is save in string data type
- Use a cast to take numeric data

```
In [5]: z = input('Enter a number :')
        z
Enter a number :200
Out[5]: '200'
```

```
In [20]: name,age = input('enter your name:'),int(input('enter your age:'))
enter your name:wijdan
enter your age:20
```

2.4



Basic operators in programming

- Python Operators - Arithmetic
- Python Operators - Comparison
- Python Operators - Logical
- Python Operators - Bitwise
- Python Operators - Assignment
- Python Namespace

2.4 Basic Operators in Programming

2.4.1 Python Operators - Arithmetic

- Assume variable x holds 2 and variable y holds 4, then −

```
In [8]: x = 2
        y = 4
        print("x + y = ", x+y)
        print("x - y = ", x-y)
        print("x * y = ", x*y)
        print("x / y = ", x/y)
        print("x // y = ", x//y)
        print("x ** y = ", x**y)
```

```
x + y =  6
x - y =  -2
x * y =  8
x / y =  0.5
x // y =  0
x ** y =  16
```

2.4 Basic Operators in Programming

2.4.2 Python Operators - Comparison

- These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.
- Assume variable x holds 2 and variable y holds 4, then –

```
In [10]: x = 2
         y = 4
         print("x > y = ", x>y)
         print("x < y = ", x<y)
         print("x == y = ", x==y)
         print("x != y = ", x!=y)
         print("x >= y = ", x>=y)
         print("x <= y = ", x<=y)
```

```
x > y =  False
x < y =  True
x == y =  False
x != y =  True
x >= y =  False
x <= y =  True
```

2.4 Basic Operators in Programming

2.4.3 Python Operators - Logical

- There are following logical operators supported by Python language.
- Assume variable x holds 2 and variable y holds 4, then −

```
In [11]: x = True
         y = False
         print("x and y = ", x>y)
         print("x or y = ", x<y)
         print("x not y = ", x==y)
```

```
x and y =  True
x or y =  False
x not y =  False
```

2.4 Basic Operators in Programming

2.4.4 Python Operators - Bitwise

- Bitwise operator works on bits and performs bit by bit operation.
- Assume if x = 8; and y = 4; Now in the binary format their values will be 1000 and 0100 respectively.

```
In [13]: x = 8
         y = 4

         print(x&y) #and
         print(x|y) #or
         print(~x) #not
         print(x^y) #exclusive or
         print(x>>2) #bitwise right shift
         print(x<<2) #bitwise left shift
```

```
0
12
-9
12
2
32
```

2.4 Basic Operators in Programming

2.4.5 Python Operators - Assignment

- Assigns values from right side operands to left side operand

Assignment operators in Python		
Operator	Example	Equivalent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

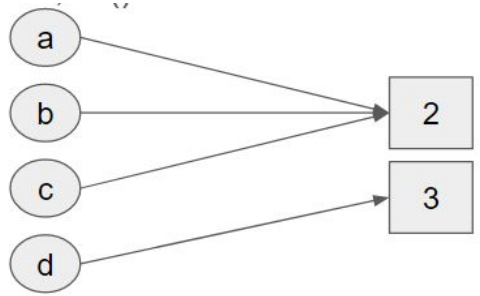
2.4 Basic Operators in Programming

2.4.6 Python Namespace

- Name (also called identifier) is simply a name given to objects.
- Everything in Python is an object.
- Name is a way to access the underlying object.
- We can get the address (in RAM) of some object through the built-in function, id()

```
In [14]: a = 2
         print("address for name 2", id(a))
         print("address for object a", id(2))

address for name 2 140704690135920
address for object a 140704690135920
```

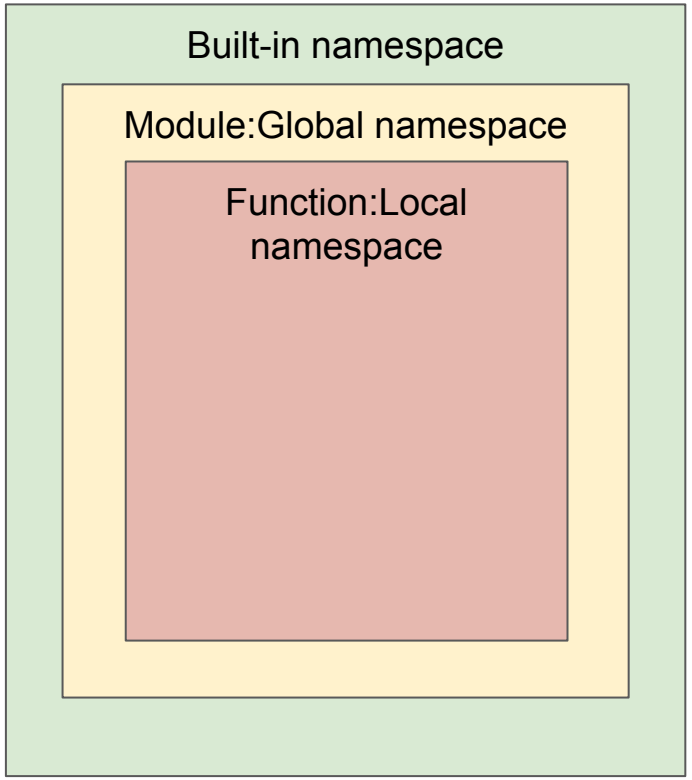
```
In [17]: a = 2
         b = a
         c = 2
         d = 3
         print("address for a", id(a))
         print("address for b", id(b))
         print("address for c", id(c))
         print("address for d", id(d))

address for a 140704690135920
address for b 140704690135920
address for c 140704690135920
address for d 140704690135952
```



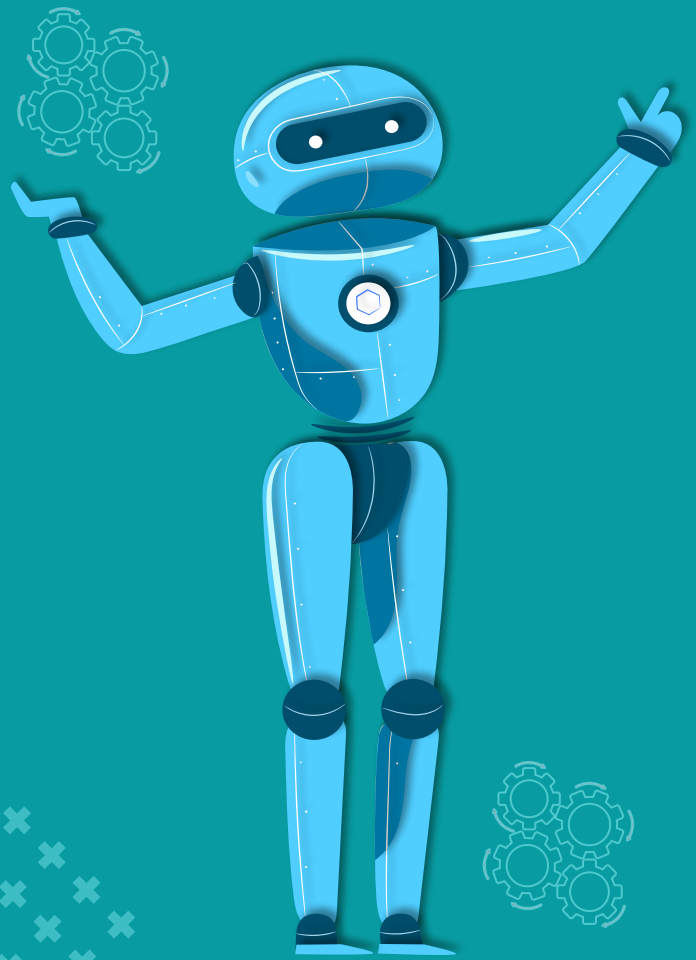
2.4 Basic Operators in Programming

2.4.6 Python Namespace

- Namespace is collection of name
- Built-in names is created when Python started.
- Each module creates its own global namespace
- Local namespace is created when a function is called



2.5



Control Structures in Programming

- Python Control Structure - If..Else
- Python Control Structure - For loop
- Python Control Structure - while loop
- Python Function
- Python Exercise

2.5 Control Structures in Programming

2.5.1 Python Control Structure - If..Else

- Decision-making is needed when we only want a code to be executed if a certain requirement is met.
- The program evaluates the condition and will execute statements if the condition result is True
- Python take non-zero values as True, None and 0 as False.

```
if condition:
        statement(s)
elif condition:
        statement(s)
else:
        statement(s)
```

```
In [21]: value = int(input('enter a number:'))

if value > 0:
    print('positive number')
elif value == 0:
    print('zero')
else:
    print('negative number')
```

```
enter a number:20
positive number
```

2.5 Control Structures in Programming

2.5.2 Python Control Structure - For loop

- The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects.
- Here, val is the variable that takes the value of the item inside the sequence on each iteration.
- Loop continues until we reach the last item in the sequence.

```
for val in sequence:
        Body of for
```

```
In [22]: car = ['BMW','Merc','Proton']
         for x in car:
             print(x)

BMW
Merc
Proton
```

```
In [24]: for x in 'Mercedes':
             print(x)

M
e
r
c
e
d
e
s
```

2.5 Control Structures in Programming

2.5.3 Python Control Structure - while loop

- The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is True.
- We generally use this loop when we don't know beforehand, the number of times to iterate.

```
while condition:
        Body of while
```

```
In [39]: a = 1
         b = 10

         while a < b:
             print('a lower than b')
             a = a+1
```

```
a lower than b
a lower than b
a lower than b
a lower than b
a lower than b
a lower than b
a lower than b
a lower than b
a lower than b
```

2.5 Control Structures in Programming

2.5.4 Python Function

- In Python, function is a collection of associated statements that perform a specific task.
- Functions help break into smaller and more flexible parts of our program. As our system grows bigger and bigger, it's more structured and manageable by functions.
- It also prevents repetition, and makes code reusable.

```
def function_name(parameters):
        """This function……"""
        statement(s)
```

```
In [43]: def my_function():
             """This function to
             print hello"""

             print('Hello')

         my_function()
```

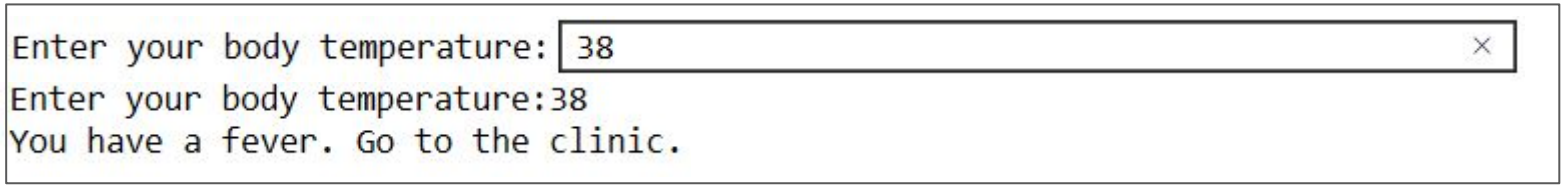
```
Hello
```

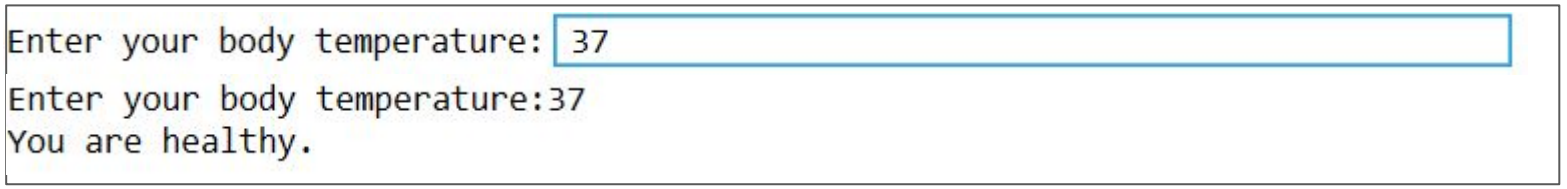
```
In [42]: def my_function():
             """This function to make
             addition between a and b"""
             a = int(input('a:'))
             b = int(input('b:'))
             print(a+b)

         my_function()
```

```
a:20
b:30
50
```

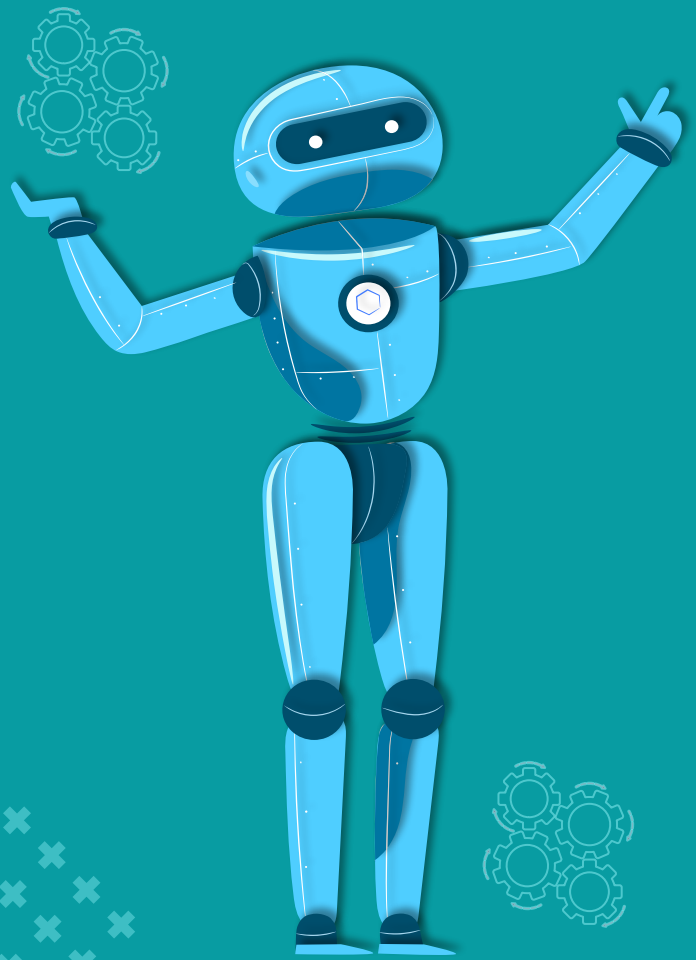
2.5.5 Python Exercise

- Create a function to determine fever
- When the function is called
  - Ask to enter body temperature
  - Answer whether or not you have a fever
    - 38 and above - fever
    - Below than 38 - healthy





2.6



Object Oriented Programming

- Basic Object Oriented Programming Principle
- Python Object Oriented Programming - class
- Python Object Oriented Programming - class object
- Python Object Oriented Programming - Method
- Python Object Oriented Programming - Inheritance
- Python Object Oriented Programming - Encapsulation
- Python Object Oriented Programming - Polymorphism

2.6 Object Oriented Programming

2.6.1 Basic Object Oriented Programming Principle

- One of the popular approach to solve a programming problem is by creating objects.
- An object has two characteristics:
- Attributes
- behavior
- Human is an object
- Name,age,color are attributes
- Singing, dancing are behavior
- The concept of OOP in Python focuses on creating reusable code. This concept is also known as DRY (Don't Repeat Yourself).

Basic OOP Principle

Inheritance
A process of using details from a new class without modifying existing class.

Encapsulation
Hiding the private details of a class from other objects.

Polymorphism
A concept of using common operation in different ways for different data input.

2.6 Object Oriented Programming

2.6.2 Basic Object Oriented Programming - Class

- A class is a blueprint for the object.
- We can think of class as a sketch of a parrot with labels.
- It contains all the details about the name, colors, size etc.
- Based on these descriptions, we can study about the parrot.
- Here, parrot is an object.

In [1]:
```
class Car:

    def __init__(self, name, cc, manufacturer, auto_transmission):
        self.name = name
        self.cc = cc
        self.manufacturer = manufacturer
        self.auto_transmission = auto_transmission

car1 = Car("Axia",1.0,"Produa",True)
car2 = Car("Wira",1.6,"Proton",False)
print(car2.name)
```

```
Wira
```

2.6 Object Oriented Programming

2.6.3 Basic Object Oriented Programming - Class Object

- An object (instance) is an instantiation of a class.
- When class is defined, only the description for the object is defined.
- Therefore, no memory or storage is allocated.

In [2]:
```
class Car:

    #class attributes
    category = 'transportation'

    #instance attributes
    def __init__(self, name, cc, manufacturer, auto_transmission):
        self.name = name
        self.cc = cc
        self.manufacturer = manufacturer
        self.auto_transmission = auto_transmission

car1 = Car("Axia",1.0,"Produa",True)
car2 = Car("Wira",1.6,"Proton",False)
print(car2.name)
print(car2.__class__.category)
```

```
Wira
transportation
```

2.6 Object Oriented Programming

2.6.4 Basic Object Oriented Programming - Method

- Methods are functions defined inside the body of a class.
- They are used to define the behaviors of an object.

```
In [5]: class Car:

            #class attributes
            category = 'transportation'

            #instance attributes
            def __init__(self, name, cc, manufacturer, auto_transmission):
                self.name = name
                self.cc = cc
                self.manufacturer = manufacturer
                self.auto_transmission = auto_transmission

            #instance method
            def roadtax(self):
                if self.cc <= 1.0:
                    return 'RM20'
                elif self.cc >= 1.001 and self.cc <= 1.2:
                    return 'RM55'
                elif self.cc >= 1.201 and self.cc <= 1.4:
                    return 'RM70'
                elif self.cc >= 1.401 and self.cc <= 1.6:
                    return 'RM90'
                elif self.cc >= 1.601 and self.cc <= 1.8:
                    return 'RM200'
                else:
                    return 'not define'


        car1 = Car("Axia",1.0,"Produa",True)
        car2 = Car("Wira",1.6,"Proton",False)
        print(car2.name)
        print(car2.__class__.category)
        print(car1.roadtax())
        print(car2.roadtax())
```

```
Wira
transportation
RM20
RM90
```

2.6 Object Oriented Programming

2.6.5 Basic Object Oriented Programming - Inheritance

- Inheritance is a way of creating new class for using details of existing class without modifying it.
- The newly formed class is a derived class (or child class).
- Similarly, the existing class is a base class (or parent class).

In [9]:
```
class Car:

    #class attributes
    category = 'transportation'

    #instance attributes
    def __init__(self, name, cc, manufacturer, auto_transmission):
        self.name = name
        self.cc = cc
        self.manufacturer = manufacturer
        self.auto_transmission = auto_transmission

class car_type(Car):
    def __init__(self, name, cc, manufacturer, auto_transmission, types):
        super().__init__(name, cc, manufacturer, auto_transmission)
        self.types = types
car3 = car_type("Waja",1.6,"Proton",False,"Sedan")
print(car3.name)
```

```
Waja
```

2.6 Object Oriented Programming

2.6.6 Basic Object Oriented Programming - Encapsulation

- Using OOP in Python, we can restrict access to methods and variables.
- This prevent data from direct modification which is called encapsulation.
- In Python, we denote private attribute using underscore as prefix i.e double " __ ".

In [22]:
```
class teh_tarik:

    def __init__(self):
        self.__price = 1.40

    def sell(self):
        print("Selling price: {}".format(self.__price))

    def setPrice(self, price):
        self.__price = price

#original price
t = teh_tarik()
t.sell()

#change price
t.__price = 1.6
t.sell()

#change using function
#inside class
t.setPrice(1.60)
t.sell()
```

```
Selling price: 1.4
Selling price: 1.4
Selling price: 1.6
```

2.6 Object Oriented Programming

2.6.7 Basic Object Oriented Programming Principle - Polymorphism

- Polymorphism is an ability (in OOP) to use common interface for multiple form (data types).
- Suppose, we need to color a shape, there are multiple shape option (rectangle, square, circle).
- However we could use same method to color any shape.
- This concept is called Polymorphism.

In [23]:
```
class Parrot:

    def fly(self):
        print("Parrot can fly")

    def swim(self):
        print("Parrot can't swim")

class Penguin:

    def fly(self):
        print("Penguin can't fly")

    def swim(self):
        print("Penguin can swim")

# common interface
def flying_test(bird):
    bird.fly()

#instantiate objects
blu = Parrot()
peggy = Penguin()

# passing the object
flying_test(blu)
flying_test(peggy)
```

```
Parrot can fly
Penguin can't fly
```