AI Programming & Self-Driving Car

Acknowledgement

- Cytron
- RaspberryPi.org

Hello. I'm Daniel Vong.

Ts. Daniel Vong

BEng (Hons), BSc, CEng, MIET, MCP

Co-Founder & CTO, Wangi Lai PLT

Hello. I'm Daniel Vong.

Lim Tong En

AI Engineering Intern, Wangi Lai PLT

EEE Student in University of Southampton Malaysia

AI Programming & Self-Driving Car

Unit Outline

- Session 1: Introduction to Hardware and Python
- Session 2: Computer Vision & Object Recognition
- Session 3: Assembly & Integration of Software and Hardware
- Session 4: Challenge &QnA
- Session 5: Challenge &QnA

AI Programming & Self-Driving Car

Learning objectives

- In this module you will learn about:
  - General Raspberry Pi Ports and GPIO
  - Types of Sensor & How to read from sensor
  - Fundamental Python
  - Flash Python Script to control LED

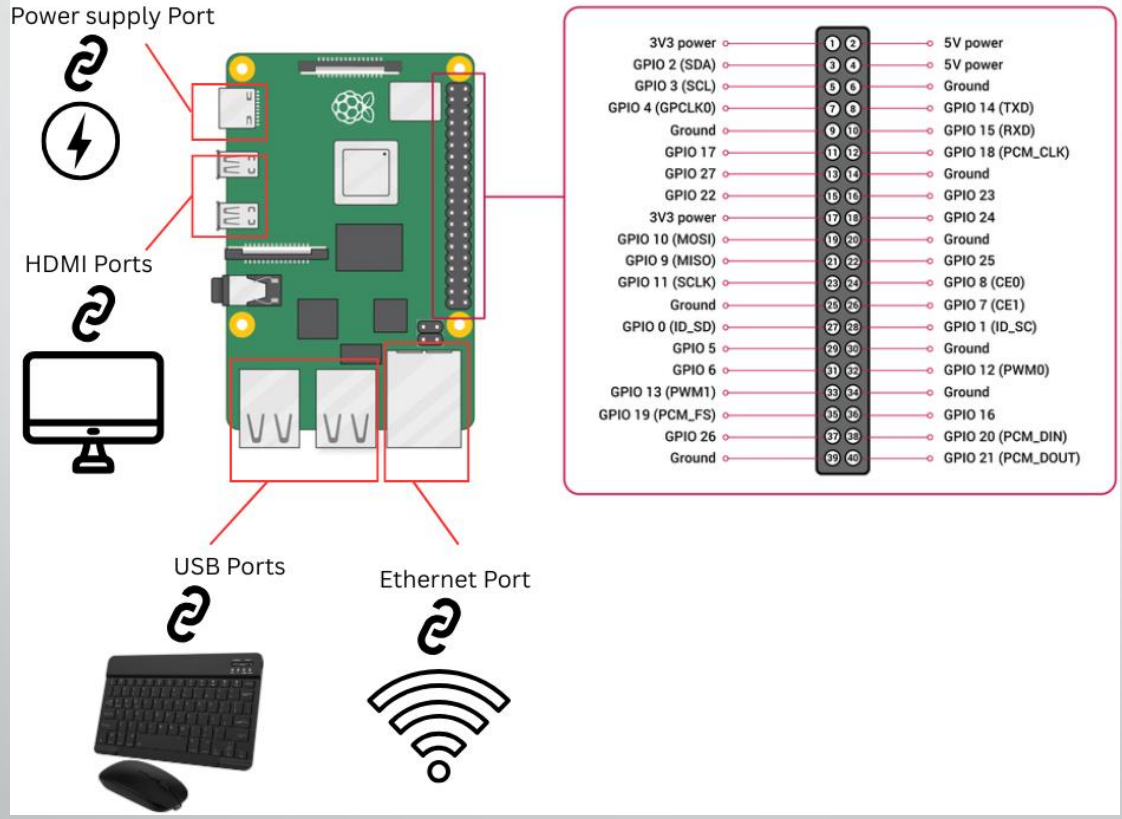
Session 1: General Raspberry Pi Ports and GPIO

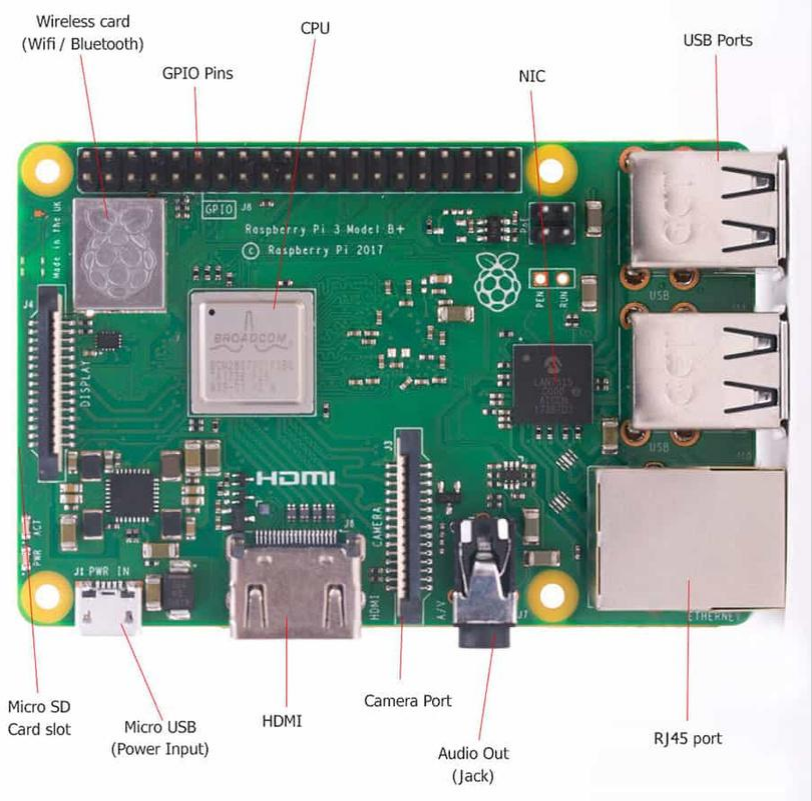
Day 1

What is Raspberry Pi?

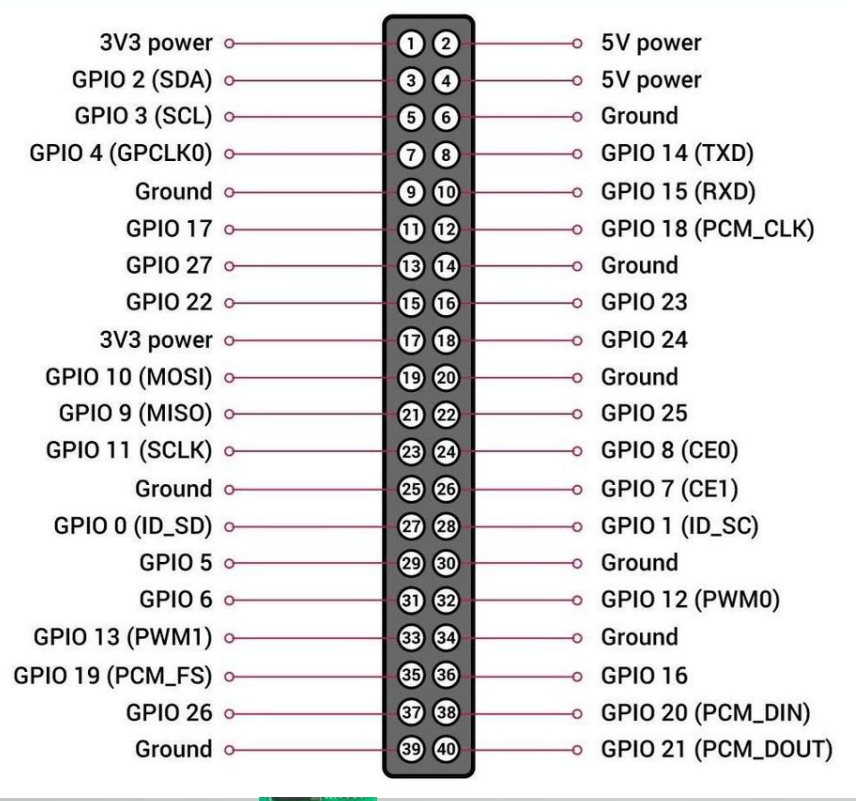
- A scaled down computer (single-board computer) that works and equipped with components of a standard computer but without in-built storage and hardware to interact with it.

Architecture of Raspberry Pi





Raspberry Pi 5 Pinout



Power & GND Pins:

Power pin directly supplies electricity of 3.3V or 5V respectively from Pi.

GND Pins complete the circuits by offering.

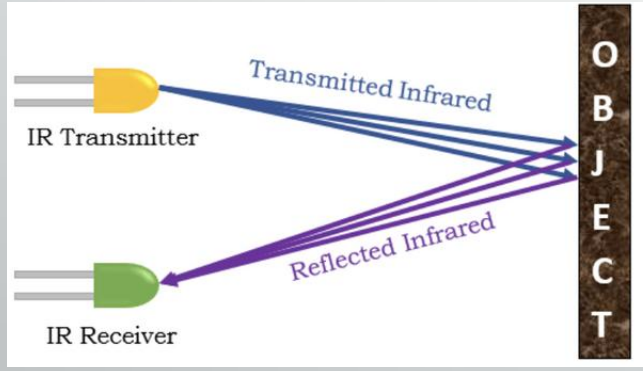
GPIO Pins:

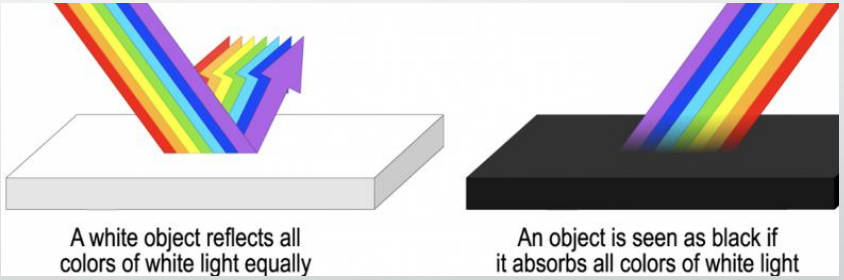
General-purpose pins; programmable pins for various functions.

Eg. Turning LED on/ off

Sensors (1)

- **Infrared sensor –** emits and detect the infrared radiation to determine object's property (e.g. colour and temperature) within a certain range.
- **Theory:** Line tracking

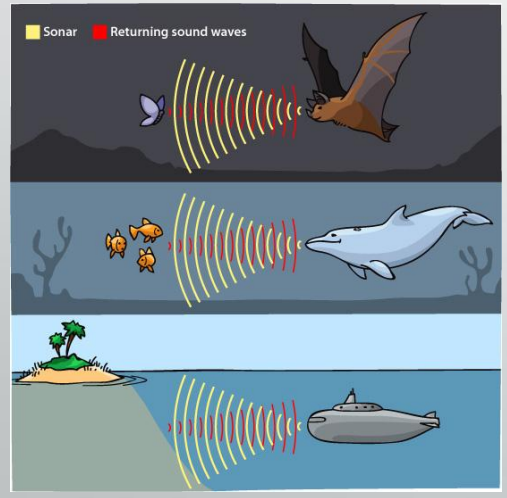




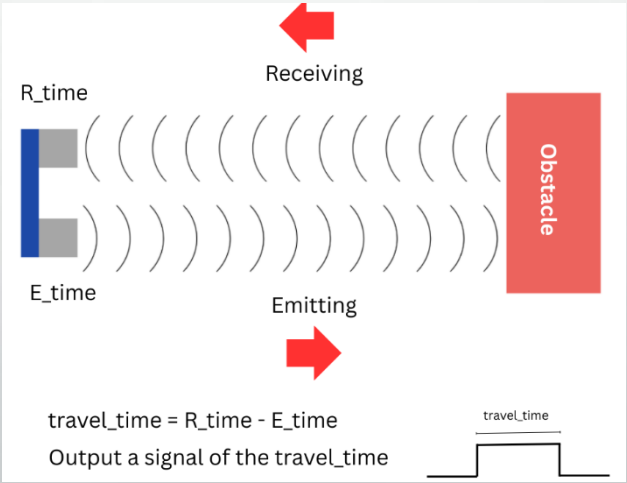
Receiver detects difference in received reflectance radiation.

Sensors (2)

- **Ultrasonic sensor – emits and detect the ultrasonic sound to determine presence obstacles and predict its distance.**
- **Theory:**



Line tracking

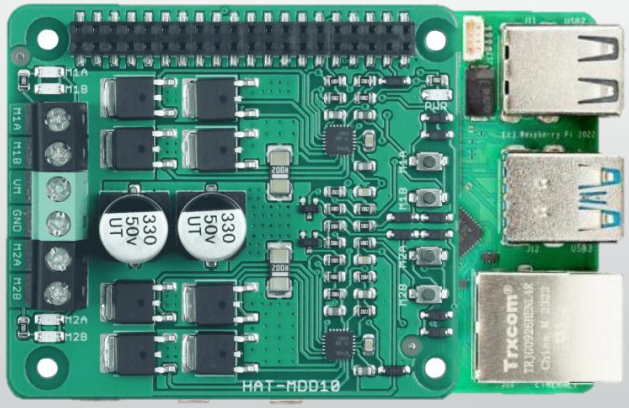


Safety Precaution with Motor (1)

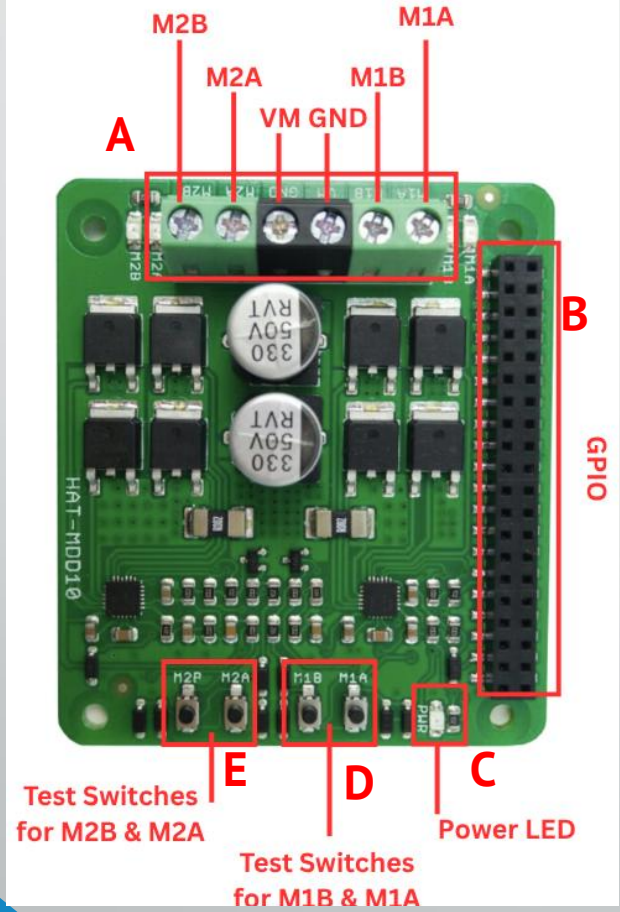
- **Why should we never connect motor directly to Raspberry Pi?**
  - Motor demands more power than the Pi could supply. Therefore, damaging the Pi.

Safety Precaution with Motor (2)

- **Purpose of Motor Driver?**
  - Acts like a bridge between motor and Pi to boost Pi's signal.

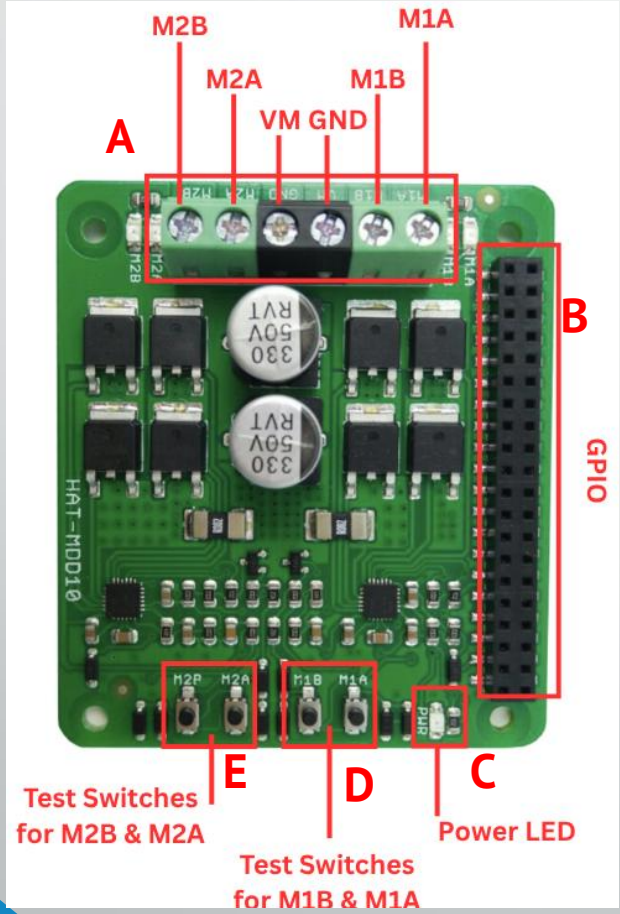


What is Motor Driver?



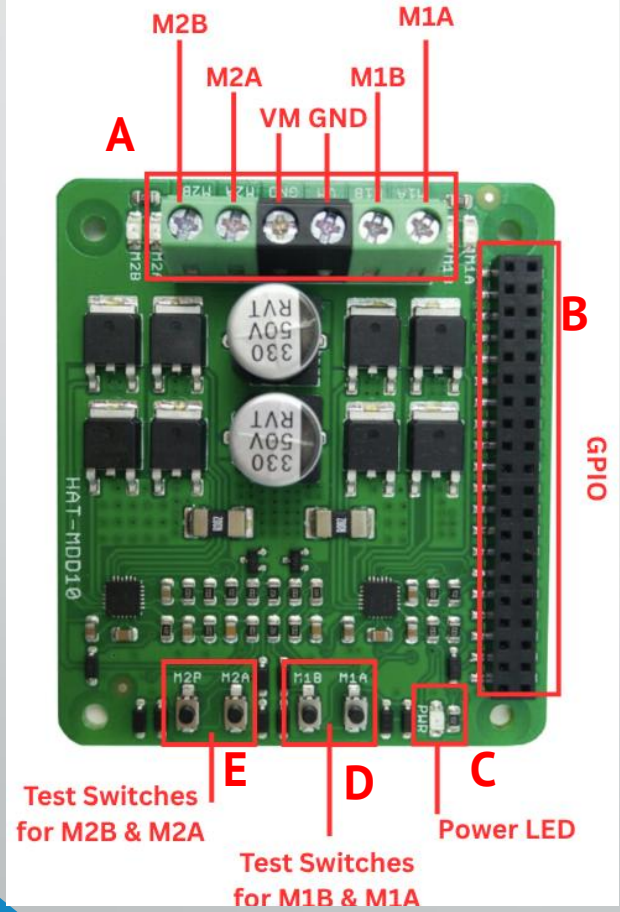
Box, Function	Pin Name	Pin Description
A, Terminal Block	M1A	Connect to Motor1 terminal A
	M1B	Connect to Motor1 terminal B
	VM	Battery positive supply (6V to 24V)
	GND	Battery negative supply
	M2A	Connect to Motor2 terminal A
	M2B	Connect to Motor2 terminal B

What is Motor Driver?



Box, Function	Pin Name	Pin Description
B, Raspberry Pi GPIO Connector	-	Provide signal to control motor speed & direction
C, Power LED		Indicate ON/OFF status of motor driver

What is Motor Driver?



Box, Function	Pin Name	Pin Description
D, Motor 1 Test Switches	M1A	To test Motor 1 in direction A (clockwise)
	M1B	To test Motor 1 in direction B (anti-clockwise)
E, Motor 2 Test Switches	M2A	To test Motor 2 in direction A (clockwise)
	M2B	To test Motor 2 in direction B (anti-clockwise)

Access Raspberry Pi Remotely (1)

- On Raspi:
  - Open up terminal, type in command "`sudo raspi-config`".
  - Interfacing Options -> Enable VNC and SSH options -> Click Finish
  - Type "`sudo reboot`" in terminal
  - Find your Raspi IP Address by hovering your cursor onto the WiFi icon. Copy down.
  - Username: raspi ; Password: raspi123

Access Raspberry Pi Remotely (2)

- On desktop:
  - Download VNC Viewer here.
  - Open the app and type in Raspi IP address.
  - Sign in by typing in your Raspi username and password.

Knowledge check

1. Why do we use a motor driver with a Raspberry Pi?
   a) To protect the motors from overvoltage
   b) Because the Raspberry Pi cannot directly provide enough power or control to motors
   c) To increase the speed of the Raspberry Pi

2. Which of the following allows you to control the Raspberry Pi's desktop remotely over Wi-Fi?
   a) SSH (Secure Shell)
   b) VNC (Virtual Network Computing)
   c) GPIO

Exercise

Access Raspberry Pi 5

Summary

- In this session you've learnt about:
  - General Raspberry Pi Ports and GPIO
  - Types of Sensor & How to read from sensor
  - Fundamental Python
  - Flash Python Script to control LED

AI Programming & Self-Driving Car

Learning objectives

- In this module you will learn how to:
  - Navigate and use **Visual Studio Code** and the **Terminal** on a Raspberry Pi for Python development
  - Understand and apply basic **Python data types**, loops (`for`, `while`), and functions
  - Write and run simple Python scripts to perform tasks like printing output, using variables, and creating loops

Session 2: Introduction to Python

Day 1

What is Python?

- Python is a simplified programming language, that allows one to do:

Web Development | Machine Learning | Data Analysis | Game Development

Python

```
print("Welcome to Malaysia")
```

C

```
Console.WriteLine("Welcome to Malaysia");
```

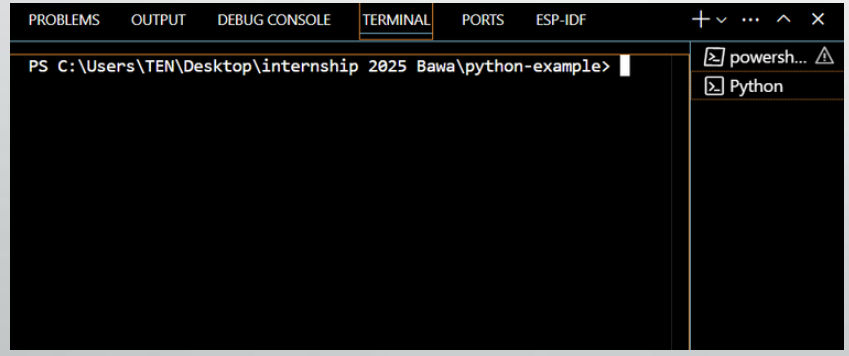
Coding Tools

Visual Studio Code

This is a code editor that support various programming languages.

Terminal

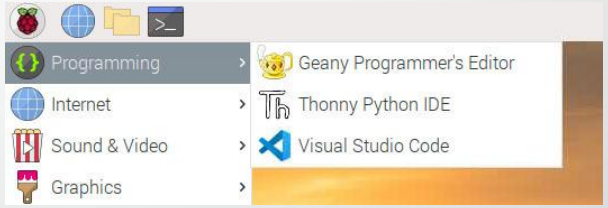
The display area for the output of your codes.



VS Code Setup Guide (1)

- Download Visual Studio Code by running:

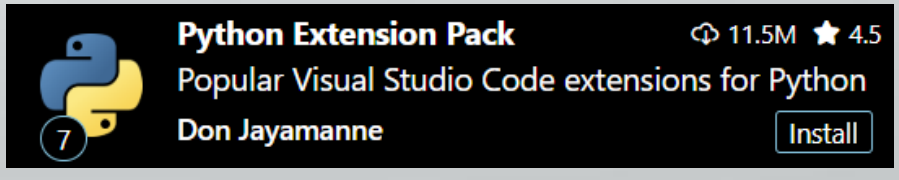
```
sudo apt update
sudo apt install code
```



- Click on Pi icon -> Programming -> Visual Studio Code
- Launch Visual Studio Code

VS Code Setup Guide (2)

- Navigate to sidebar and click on this icon 
- Type “Python” in search bar
- Install the following extensions and package:
- Create new .py file and start programming.



Library

- **Definition:**
- Reusable code written by others to take over the boring and lengthy work.
- **Example:**

OpenCV

RPi.GPIO

Ultralytics

picamera2

Numpy

How to use a Library?

- Install the library via terminal
- Type in "`pip install <library_name>`".
  - E.g: `pip install numpy`
- Upon successful installation, include library at the top of python script.
  - E.g: `import numpy`
  - `# The rest of your codes are down here`

Bash File (1)

- Definition:
  - File with series of commands in it. File format ends with ".sh".
- Why use bash file?
  - To increase efficiency of setup and avoid repetitive tasks.

Bash File (2)

- How to setup a bash command?
  - Navigate to terminal
  - Type in "`wsl --install`"
  - Reboot laptop
- How to run a bash file?
  - Navigate to terminal
  - Navigate to the file's directory
  - Type "`bash <name_file>.sh`" and enter.

Virtual Environment

- **Definition:**
  - Isolation of different Python projects from each other.
- **Why use virtual environment?**
  - To avoid conflicts between different project requirements.

How to setup virtual environment?

- Navigate to terminal
- Create virtual environment by typing in “python -m venv <venv_name>”
- Activate virtual environment by changing directory to “<venv_name>\Scripts\activate.bat” for Windows

  “source <venv_name>/bin/activate” for Raspi

```
C:\Users\TEN\Desktop\internship 2025 Bawa\python-example>test_env\Scripts\activate.bat

(test_env) C:\Users\TEN\Desktop\internship 2025 Bawa\python-example>
```

- Deactivate after using with typing in “deactivate”

Python Basic Syntax

```
#This is how you comment

country = "Malaysia"

print("Welcome to", country)
```

Putting a "#" before any a statement could document your code

This is a **VARIABLE** called "country" and store "Malaysia" inside it

Displaying "Welcome to Malaysia" with print() **FUNCTION** in terminal

Python & Programming concept

Variable - a container to store your data.

Function - take in variable (parameter) and produces output after processing.
Example, `print()`.

Python & Programming concept - Hands On

Question: Solve a simple math question, where a = 1, b=5 and print out the final answer.

Data Types in Python – Strings (1)

- Text based data, where it has series of characters and quoted within “”.
- For example:
  - “123”
  - “abc”
  - “$%^”
- String Type Conversion: `str()` - convert any data format to string.

```
int_to_str = str(100)
print(type(int_to_str))# output: <class 'str'>
```

Data Types in Python – Strings (2)

- Can choose to use **"abc"** or **'abc'.**
- However, string must start and enclosed with either double quotes ("") or single quotes ('), never mix them together.

```
# With double & single quotes
country = "'Malaysia'"
print(country)
# output: 'Malaysia'

# With double & single quotes
country = "Malaysia'
print(country)
# output: ERROR
```

```
name1 = "Tianjin"
name2 = "University"

# method 1
print(name1, " ",name2)
# output: Tianjin University

# method 2
fullname = name1+" "+name2
print(fullname)
# output: Tianjin University
```

Data Types in Python - Numbers & Maths (1)

- Number types:
  - integer (e.g. 3, 100, -200)
  - float (e.g. 2.5, 50.0023, -200.503)
- Number type Conversion:
  - int() - convert any number format to integer
  - float() - convert any number format to float

```
#int() function
float_number = 3.051
int_number = int(float_number)
print(int_number)
# output: 3

#float() function
int_number = -100
float_number =
float(int_number)
print(float_number)
# output: -100.0
```

Data Types in Python - Numbers & Maths (2)

- **Basic Maths Operators**
  - Addition +
  - Subtraction -
  - Division /
  - Multiplication *
  - Power **

```
x = 2.0
y = 5.6

#no brackets, so multiplication prioritize
ans = x+y*3
print(ans) #output: 18.8

#brackets used, so addition prioritize
ans2 = (x+y)*3
print(ans2) #output: 22.8

#normal division
print(7/2) #output: 3.5
```

Data Types in Python - Numbers & Maths (3)

- **Special Maths Operators**
  - Floor Division //
  - Modulo Operation %

```
x = 2.0
y = 5.6

#ground division returns rounded integer
print(7//2) #output: 3

#modulo operation returns remainder of division
print(7%2) #output: 1
```

Data Types Strings & Maths - Hands On

- Question 1: What is the output of the following code?

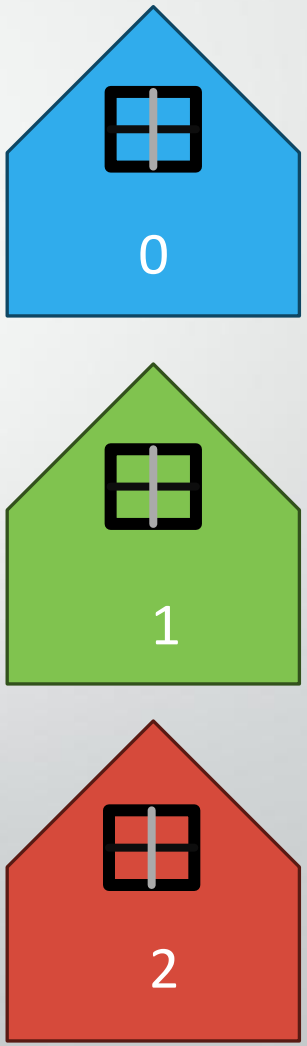
```
sum = "2" + "2"
print (sum)
```

- Question 2: What is the remainder of (5200 + 35) ÷ 33?

Data Types in Python – List

- **Application**
  - List stores multiple elements of same or different data type together, within square brackets, “[” and “]”.
- **Example**
  - `number_list = [100, 10, 22, 30.0 ]`
  - `customer_1_info = [“John”, 20, [“Burger”, “Fries”, “Cola”] ]`
  - `fruits_list = [“apple”, “orange”, “banana”]`

Data Types in Python – List



- **How to extract element?**
  - Use index: address for each element in the list.
  - `fruits_list = ["apple", "orange", "banana"]`
- **Example**
  - `number_list = [12, 33, 66, 77.0]`
  - `print(number_list[2]) #output: 66`

Data Types List – Hands on

customer_1_info = [“John”, 20, [“Burger”, “Fries”, “Cola”]]

Question 1

What will the below code print out?

```
customer1_info[2]
```

A. [“Burger”, “Fries”, “Cola”]

B. Burger

C. Error

Question 2

What will the below code print out?

```
customer1_info[2][2]
```

A. [“Burger”, “Fries”, “Cola”]

B. Cola

C. Fries

Data Types in Python - Boolean

Definition:

A data type with only 2 possible outcome, true or false.

Application:

- Evaluate if a statement is true or false
- if-else statement

Comparison Operator:

Operator	Name
==	Equal
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

```
print(2 == 2) #output: true
print(10>7) #output: true
print(5!=5) #output: false
print(7>=7) #output: true
```

**Read more about Boolean here.

Conditional Statements: if, elif and else (1)

Application:

Only execute the specific code block if a particular condition is met.

Structure:

```
if condition1:
  #code block 1

elif condition2:
  #code block 2

else:
  #code block 3
```

Prioritises and evaluate first, run code block 1 if true.

Second in line to be evaluated, if condition1 not met, run code block 2.

If none of the condition met, run code block 3

Conditional Statements: if, elif and else (2)

- **Example:**

```
number = -100
if number > 0:
    print('Positive number’)

elif number < 0:
    print('Negative number’)
else:
    print('Zero’)

print('This statement is always executed’)

#output:
#Negative Number
#This statement is always executed
```

Conditional Statement - Exercise

- Question:
  - You're creating a simple health monitoring app that checks if someone has a fever.
- Requirements:
  - Check the patient's temperature:
  - Normal (below 37.5°C): "Temperature normal - you're healthy!"
  - Low fever (37.5°C to 38.4°C): "Mild fever - rest and drink fluids"
  - High fever (38.5°C and above): "High fever - see a doctor immediately!"
  - If they have a headache: add "Take pain relief if needed"
  - If they have a cough: add "Consider wearing a mask around others"

Given Variables:

temperature = 38.2

headache = True

cough = False

For Loops (1)

Application:

Iterating over a sequence of elements, such as list.

Structure:

Method 1

```
for element in your_list:
    #your code here
```

element: a newly defined variable to represent each element in your pre-defined list.

your_list: an already defined list.

Method 2

```
for count in range(times):
    #your code here
```

count: a newly defined variable to represent each number of the given range.

range(times): range() returns sequence of numbers, if times = 3, then output: 0, 1 ,2.

For Loops (2)

Example:

```
fruits = ["apple", "banana", "cherry"]

for fruit in fruits:
  print(fruit)
#output:
#apple
#banana
#cherry

for fruit in fruits:
  print(fruit)
  if fruit == "banana":
    break
#output:
#apple
#banana
```

? Why in 2nd for loop, it did not iterate through all 3 fruits?

- **break()** : used to exit or "break" out of a loop before it finishes iterating through the list

For Loop - Exercise

- Question:
  - A patient wore a heart rate monitor for 5 hours. Check if their heart rate was normal during each hour.
- Requirement:
  - If 60-100 bpm: print "Normal heart rate: [X] bpm"
  - If below 60: print "Low heart rate: [X] bpm"
  - If above 100: print "High heart rate: [X] bpm"
- Hint: [X] use if-else statement inside

Given Data:

```
heart_rates = [72, 85, 90, 78, 82]
```

While Loops (1)

- **Application:**
  - Kept on looping as long as the given condition is true, it will break out the loop when condition becomes false
- **Structure:**

```
while bool_condition:
    #your code will execute if bool_condition is true
```

- **bool_condition:** This conditional input must be in Boolean type, either true or false.

While Loops (2)

- **Example**

```
i = 1
while i < 6:
  print(i)
  i = i+1
else:
  print("i is no longer less than 6")
#output:
#1
#2
#3
#4
#5
#i is no longer less than 6
```

? How to make for forever looping while loop?

Try and Except (1)

- **Application**
  - Execute code block while checking for possible errors.
- **Structure**

```
try:
   #your code to run
except:
   #what you want to do if there's an error
```

- **try -** runs main code block and check for errors
- **except** - if an error is raised, it will stop the main code block and run its code instead.

Try and Except (2)

- **Example**

```
try:
  print(x)
except:
  print("An Error occurred")

#output: An Error occurred
```

- More Specific Version:

```
try:
  print(x)
except NameError:
  print("Variable x is not defined")
except:
  print("Something else went wrong")

# output: Variable x is not defined
```

Create your own Function

Application:

A function is a block of code which only runs when it is called.

You can pass inputs into a function. A function can return data as an output.

Structure:

```
def my_function():
   #your code block

my_function()
```

def uses to intialize and define your own function

my_function can be renamed to whatever you wish

( ) can take in any intended input as parameter

Student must call your function to execute your code.

Create your own Function

- **Example:**

```
def fullname_func(firstname, lastname):
  fullname = firstname + " " + lastname
  return fullname

person_name = fullname_func("Lim","Tong En")

print(person_name) # output: Lim Tong En
```

Create your own Function-Exercise

- Write a function to print all the even numbers given in a list.
- **Requirements:**
  - Use a function
  - Use a for loop to go through each number
  - Use an if statement to check if a number is even
  - Print out the even numbers.
- Hint: A number is even if **number % 2 == 0**

```
num_list= [15, 14, 28, 19, 30]
```

Summary

- In this session you've learnt how to:
  - 🧠 We set up **Visual Studio Code** and used the **Terminal** to run Python programs
  - 🔤 We learned about **basic data types** like strings, integers, and Booleans
  - 🔁 We practiced using **loops** to repeat actions and created **functions** to organize our code