Al Hussein
Technical University
جامعة الحسين التقنية

Internet Of Things

Getting Started with Python

Prepared by:

Dr. Murad Yaghi
Eng. Malek Al-Louzi

School of Computing and Informatics – Al Hussein Technical University

Fall 2024/2025

Introduction

- The objective of this class focuses on controlling hardware connected to the Raspberry Pi
- To accomplish that, we will be using the Python programming language
- You should be familiar with some of the basics of Python, including literals, variables, operators, control flow, and scope

Introduction

- The simplest way to create Python programs is to write your code in a text editor (e.g. nano, vim, emacs, Midnight Commander, Leafpad, etc.), save it, and then run it from the terminal with the command

```
python <FILE>.py
```

Hello World Program

- Create a Python file from the terminal

```
nano hello.py
```

- This creates a file named *hello.py* and starts editing it with the *nano* editor
- In this file, enter the following on the first line

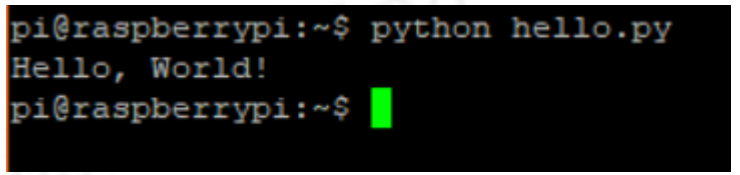
```
print("Hello, World!")
```

Hello World Program

- Save and exit (*ctrl+x* followed by *y* and then *enter*).
- Back in the Linux command prompt, enter the command

```
python hello.py
```

- This should run the code found in the file *hello.py*
- In our case, you should see the phrase printed out in the console



Python Basics

- **Comments**
  - A *comment* is any text to the right of the hash symbol #
  - Example:

```
# This is a comment and is not seen by the interpreter
print("Hello, World!")
```

- **Literals**
  - Literals, also known as *literal constants*, are fixed values and include integers (e.g. 42), floating-point numbers (e.g. 6.23), and strings (e.g. "Hello, World!").

Python Basics

- Note that strings need to be in between single quotation marks (' ') or in between double quotation marks (" ")
- Example:

```
print(42)
print("hi")
```

- **Variables**
  - Variables are containers whose values can change. We can store a number or string in a variable and then retrieve that value later
  - Example:

```
number = 42
print(number)
```

Python Basics

- **User Input**
  - You can ask a user to enter information into the terminal by using the **input()** function
  - This will prompt the user to type out some text (including numbers) and then press *enter* to submit the text
  - The **input()** function will read the text and then return it as a string, which can be stored in a variable

Python Basics

- **User Input**
  - Whatever is in between the parentheses (known as *arguments*) will be printed to the screen prior to accepting user input
  - Example:

```
message = input("Type a message to yourself: ")
print("You said:", message)
```

  - Note that you can use the **int()** function to turn a string into an integer
  - Example:

```
number = int(input("Type a number:"))
print("You entered:", number)
```

Python Basics

- **Exercise:**

Write a program that asks for the user's first name and last name (two separate **input()** calls) and then prints the user's first and last name on one line.

- **Answer**

```
first_name = input("Enter your first name: ")
last_name = input("Enter your last name: ")
print("Full name:", first_name, last_name)
```

Python Basics

- **Indentation**
  - White space (number of spaces) at the beginning of a line is important in Python
  - Statements that form a group together must have the same level of indentation
  - This will be important when we get into control flow statements (if, for) and functions

Python Basics

- **Indentation**
  - If you have written programs in other languages before, you might be familiar with curly braces { }
  - In other languages, code in between these curly braces would form a group (or block) of code
  - In Python, a group (or block) of code is designated by the level of indentation of the individual lines of code

Python Basics

- **Indentation**
  - Example:

```
answer = "yes"
guess = input("Is the sky blue? ")
if guess == answer:
    print("Correct!")
else:
    print("Try again")
```

Python Basics

- **Operators**
  - Relational and equality operators

Operator	Description	Example
`<`	`True` if the first number is less than the second, `False` otherwise	`5 < 3` returns `False`
`>`	`True` if the first number is greater than the second, `False` otherwise	`5 > 3` returns `True`
`<=`	`True` if the first number is equal to or less than the second, `False` otherwise	`2 <= 8` returns `True`
`>=`	`True` if the first number is equal to or greater than the second, `False` otherwise	`2 >=` returns `False`
`==`	`True` if the first number is equal to the second, `False` otherwise	`6 == 6` returns `True`
`!=`	`True` if the first number is not equal to the second, `False` otherwise (not equal)	`6 != 6` returns `False`

Python Basics

- **Operators**
  - An operator is a symbol that tells the interpreter to perform some mathematical, relational, or logical operation on one or more pieces of data and return the result

Python Basics

- **Operators**
  - Mathematical operators

Operator	Description	Example
`+`	Adds two numbers	`2 + 3` returns `5`
`-`	Subtracts one number from another	`8 - 5` returns `3`
`*`	Multiplies two numbers together	`4 * 6` returns `24`
`**`	Raises the first number to the power of the second number	`2 ** 4` returns `16`
`/`	Divides the first number by the second number	`5 / 4` returns `1.25`
`//`	Divides the two numbers and rounds down to the nearest integer (divide and floor)	`5 / 4` returns `1`
`%`	Divides the first number by the second number and gives the remainder (modulo)	`19 % 8` returns `3`

Python Basics

- **Operators**
  - Logical operators

Operator	Description	Example
`not`	Gives the opposite (`True` becomes `False` and vice versa)	`x = False; not x` returns `True`
`and`	Returns `True` if both operands are `True` , `False` otherwise	`x = True; y = False; x and y` returns `False`
`or`	Returns `True` if either of the operands are `True` , `False` otherwise	`x = True; y = False; c or y` returns `True`

Python Basics

- **Exercise**
  - Ask the user for two integers, and print the addition, subtraction, multiplication, division, and modulo of those numbers.
- **Answer**

```
x = int(input("First number: "))
y = int(input("Second number: "))
print(x + y)
print(x - y)
print(x * y)
print(x / y)
print(x % y)
```

Python Basics

- **Control Flow**
  - The Python interpreter executes statements in your code from the top to the bottom of the file, in sequential order. That is unless, of course, we employ some time of *control flow* statements to break this normal sequential flow
  - We introduce the **range(x,y)** function in the examples in the next slide which generates a list of numbers between the first number, x (inclusive), and the second number, y (exclusive)

Python Basics

Statement	Description	Example

`if` `elif` `else`	If a condition is true, execute the block of code underneath the *if statement*. If not, see if the condition is true in one or more *else if* (`elif`) statements. If one of those is true, execute the code block under that. Otherwise, execute the code block underneath the *else statement*. `elif` and `else` statements are optional.	`number = 42`
		`guess = int(input("Guess a number between 1-100: "))`
		`if guess == number:`
		`  print("You win!")`
		`elif guess < number:`
		`  print("Nope")`
		`  print("Too low")`
		`else:`
		`  print("Nope")`
		`  print("Too high")`
		`print("Run the program to try again")`

`while`	A *while loop* executes the block of code underneath it repeatedly as long as the condition is true.	`counter = 15`
		`while counter >= 5:`
		`  print(counter)`
		`  counter = counter - 1`

`for..in`	Iterate over a sequence of numbers or objects. The variable declared in a *for loop* assumes the value of one of the numbers (or objects) during each iteration of the loop.	`for i in range(1, 11):` `  print(i)`

Python Basics

- **Exercise**
  - Write a program that prints integers counting up from 1 to 20, except that for every multiple of 3 (3, 6, 9, etc.), the word “IOT" is printed instead.
  - **Answer**

```
for i in range(1, 21):
    if i % 3 == 0:
        print(" IOT ")
    else:
        print(i)
```

Python Basics

- **Modules**
  - Modules are another way to reuse code and help you organize your program. They are simply files that are imported into your main program. After importing a module, you can use a module in much the same way you would an object: access constants and functions using the dot-notation
  - **Example**:

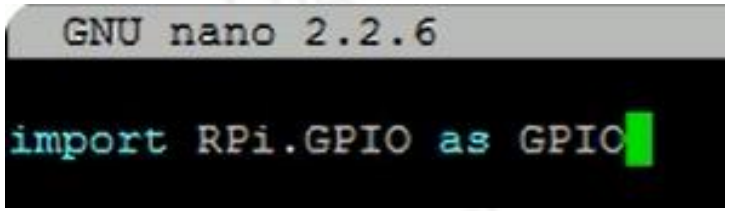
```
import stringmod

s = "Hello!"
print(stringmod.a)
print(stringmod.string_to_list(s))
```

Digital Output

- In the Hardware world, the first thing many developers like to do is blink an LED.
- In a sense, it is the "Hello, World!" of controlling Hardware.
- We'll start by controlling an LED.

Digital Output
Controlling LED

- Create new Python file using nano editor
- To interact with GPIO pins, you need to import a module “RPi.GPIO”, we can make a reference to it by typing “GPIO”

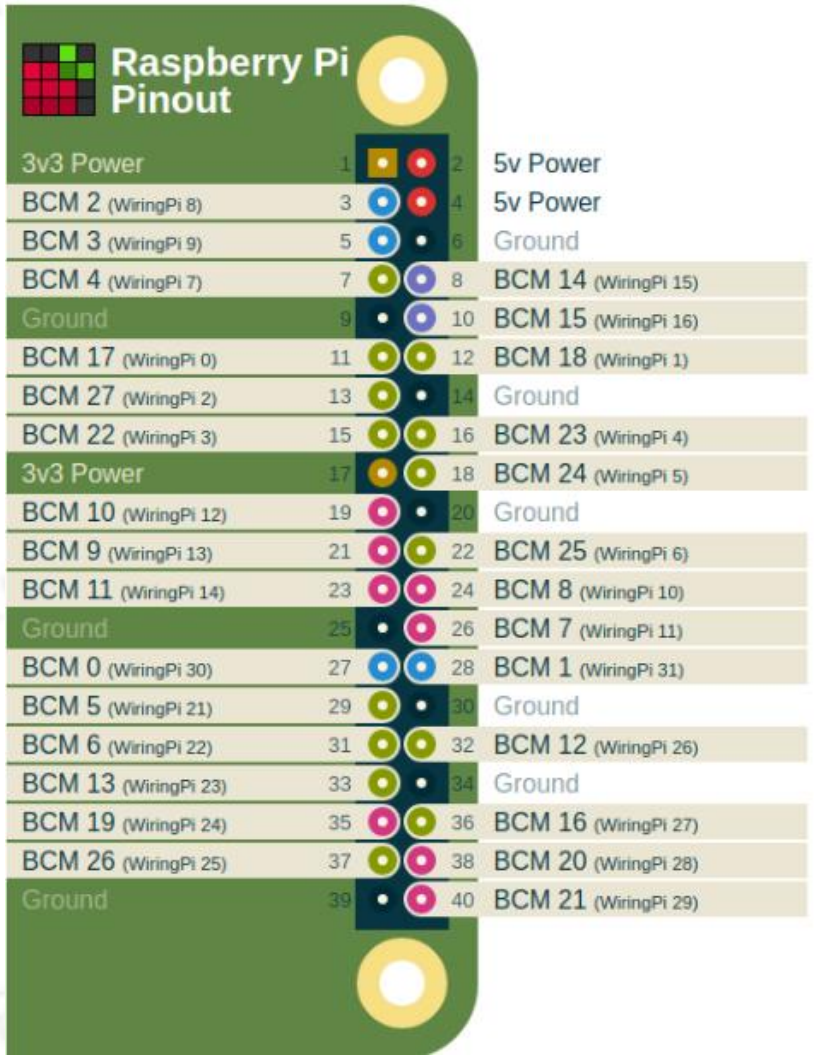


- We could change the name “GPIO” to any word you like, but the “GPIO” word is a relevant name
- Next, we need to say how to reference a GPIO pin, there are two options to do that:

```
GPIO.setmode(GPIO.BOARD)
```
OR
```
GPIO.setmode(GPIO.BCM)
```

Digital Output Controlling LED

- We will use GPIO.BOARD, it will allow us to refer to the pins by their numbers



Digital Output Controlling LED

- In our example, our LED connected to pin number 12

```
 GNU nano 2.2.6

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BOARD)

GPIO.setup(12,GPIO.OUT)
```

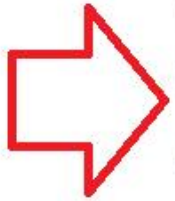
- The third line is to setup pin number 12 as an output pin
- After setting up pin 12, we can start telling it what to do
- Actually, we only have two choices
  - Set it to LOW, setting the pin to 0 V (LED turning OFF)
  - Set it to HIGH, setting the pin to 3.3 V (LED turning ON)

Digital Output Controlling LED

```
GNU nano 2.2.6

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BOARD)

GPIO.setup(12,GPIO.OUT)

GPIO.output(12,GPIO.LOW)
```
= 0 Volts

- There is a simpler way of writing HIGH and LOW, we can use 1 and 0

```
GNU nano 2.2.6

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BOARD)

GPIO.setup(12,GPIO.OUT)

GPIO.output(12,GPIO.LOW)

GPIO.output(12,GPIO.HIGH)
```



```
GNU nano 2.2.6

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BOARD)

GPIO.setup(12,GPIO.OUT)

GPIO.output(12,0)

GPIO.output(12,1)
```

Digital Output Controlling LED

- The complete program

```
 GNU nano 2.2.6

import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD)

GPIO.setup(12,GPIO.OUT)

GPIO.output(12,1)

time.sleep(3)

GPIO.output(12,0)
```

- To run the program, “sudo python filename.py”, whenever we are using a program that uses the GPIO pins, we need to include “sudo”

Digital Output Controlling LED

- **Exercise: write a program that continuously blink LED ON and OFF with a time interval of 1 second between the two states**

Any Questions???