Python

*An Introduction using the Raspberry Pi *

Frank Walsh

2026

Reference:

https://books.trinket.io/pfe/

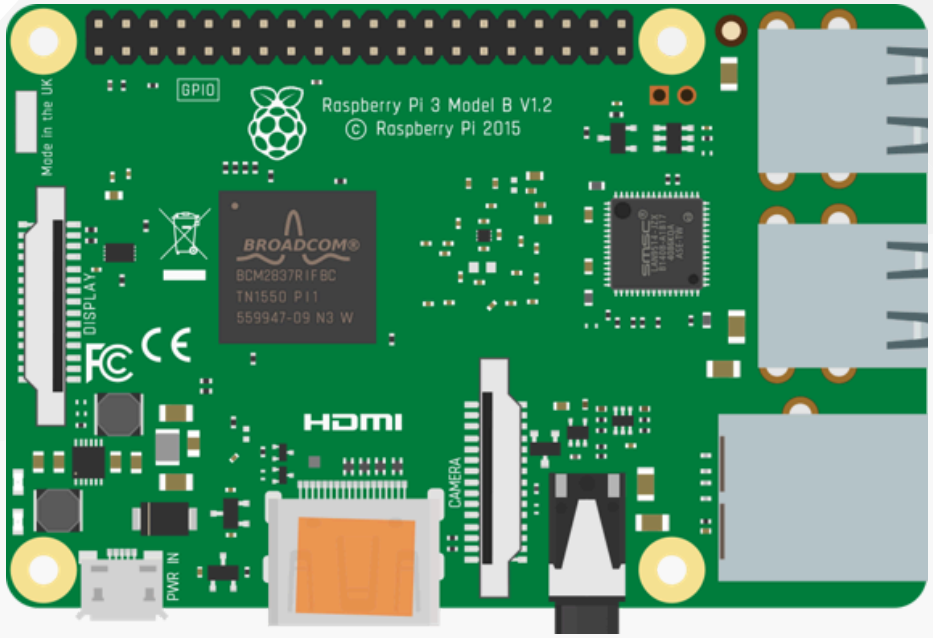
https://trinket.io/

Agenda (goals)

- Describe the Python programming language
- Understand Python fundamentals
- Identify similarities to lanuages and programming constructs you already know (Javascript/Java)
- Connect to and program a Single Board Computer (RPi)
- Explore Sense HAT and write programs with hardware interactions

Programming the RPi

- By default, the RPi supports Python as the educational language
- Supports a wide variety of programming languages:
  - Javascript(Node.js)
  - Java
  - C
  - Rust



Python Overview

- **What is Python?**
  - High-level, interpreted programming language created in 1991
  - Great for beginners and advanced users
  - Popular in data science, web development, and IoT
- **Key Features of Python**
  - **Readable and concise syntax:** Easy for beginners and experts alike.
  - **Versatile:** Used across web development, data science, automation, and more.
  - **Extensive libraries:** Libraries like SenseHat, NumPy, pandas, and Matplotlib provide powerful tools for various applications.

Interpreted vs. Compiled Languages

- Python is an interpreted language: Code is executed line-by-line by the Python interpreter.
  - Benefits: Easier to debug, allows for interactive programming.
  - Drawbacks: Generally slower than compiled languages (e.g., C, C++).

Interpreted vs. Compiled Code

- Interpreted Code (Python):
  - Run directly using an interpreter.
  - Great for rapid development and prototyping.
- Compiled Code (e.g., C/C++):
  - Converted into machine code, which the computer directly executes.
  - Typically faster, often used for performance-critical applications.

Python Applications

Mathematics & Statistics

- Libraries like NumPy and SciPy provide tools for mathematical operations.
- pandas: Essential for data manipulation and statistical analysis.
- Matplotlib and Seaborn: For data visualization, turning data into meaningful insights.

Python in Education

- Widely used as an introductory language in schools and universities.
- Readable syntax makes it ideal for beginners learning programming fundamentals.
- Tools: Trinket and other interactive coding platforms support Python learning.

Why are we using Python?

Installing Python

- No need - it's already on your RPi if you installed Rasberry Pi OS
- Can install on your desktop too (if you want)
- On Rpi command line, type `python`
    - Python interpreter will start in interactive mode as follows:

```
frank@HDipRPi:~/week8/inclassdemo $ python
Python 3.9.2 (default, Mar 12 2021, 04:06:34)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello world!')
    Hello world!
```

Using Python in Interactive Mode

- Open a Terminal or Command Prompt:
- Type `python` (or `python3` on some systems) and press Enter.
- This command opens the Python interactive shell, indicated by the prompt `>>>`.

```
frank@HDipRPi:~/week8/inclassdemo $ python
Python 3.9.2 (default, Mar 12 2021, 04:06:34)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello world!')
    Hello world!
```

Variables

- A variable is a name that refers to a value
- Named storage for data
- An *assignment statement* creates new variables and gives them values
  - no need to define variable type
- Holds different types like integers, floats, strings

Example

```
temperature = 12
humidity = 55

print(f"Temperature: {temperature}°C")
print(f"Humidity: {humidity}%")
humidity = "66%"
print(f"Humidity: {humidity}")
```

f-Strings in Python

- An f-string (formatted string literal) is a way to embed expressions/variables directly within strings using {} brackets.
- New feature from Pyhton 3.6
- Prefix the string with `f` or `F` to indicate an f-string.
- Place expressions, variables, or function calls inside {} to evaluate and insert their values

```
name = "Frank"
print(f"Hello, {name}!")
# Output: Hello, Frank!
```

Conditionals

- **What are Conditionals?**
  - Code that runs only if certain conditions are met
  - `if` , `elif` , and `else` statements
- **Example**

```
if temperature > 25:
    print("It's hot!")
elif temperature > 15:
    print("Moderate temperature.")
else:
    print("It's cold!")
```

User Input(command line)

- Built In function (`input("the prompt")`)
- Returns a String
- Use `float( )` to convert string to float

```
temperature = input("Enter the temperature in °C: ")
print(f"You entered: {temperature}°C")
temperature = int(temperature)
if temperature > 25:
    print("It's hot!")
elif temperature > 15:
    print("Moderate temperature.")
else:
    print("It's cold!")
```

Functions

- **What are Functions?**
  - Reusable blocks of code
  - Defined using `def` keyword
- **Example**

```
def check_temperature(temp):
    if temp > 25:
        return "It's hot!"
    elif temp > 15:
        return "Moderate temperature."
    else:
        return "It's cold!"

temperature = input("Enter the temperature in °C: ")
temperature = int(temperature)
message = check_temperature(temperature)
print(message)
```

Built-in Functions

- Functions provided by Python that are always available
- Examples: print(), len(), type(), sum(), max(), min(), round()

Example

```
temperatures = [12, 18, 25, 30, 15]
max_temp = max(temperatures)
min_temp = min(temperatures)
avg_temp = sum(temperatures) / len(temperatures)
print(f"Max Temperature: {max_temp}°C")
print(f"Min Temperature: {min_temp}°C")
print(f"Average Temperature: {avg_temp}°C")
```

Function Declaration

- Unlike Javascript, Python does not have hoisting; functions and variables must be defined before use

```
# Trying to call `display_temp` before it's defined
display_temp()                    # This will raise an error: NameError

def display_temp():
    temperature = sense.get_temperature()
    print(f"Temp: {temperature}C")
```

Function Parameters and Arguments

- Parameters
  - Variables listed inside the function's parentheses in the definition

    Act as placeholders for values passed into the function

    Can have default values
- Arguments
  - Values passed to the function when calling it and assigned to the parameters

```
def display_temp_humidity(show_temp=True, show_humidity=True):
    # Parameters 'show_temp' and 'show_humidity' control what to display
    temperature = sense.get_temperature()
    humidity = sense.get_humidity()

    if show_temp:
        print(f"Temp: {temperature}C")
    if show_humidity:
        print(f"Humidity: {humidity}%")
```

Returning Values from Functions

- Use return keyword to send a value back

```
def check_temperature(temp):
    if temp > 25:
        return "It's hot!"
    elif temp > 15:
        return "Moderate temperature."
    else:
        return "It's cold!"

temperature = input("Enter the temperature in °C: ")
temperature = int(temperature)
message = check_temperature(temperature)
print(message)
```

Iteration - For Loop

- **What is Iteration?**
  - Repeating actions using `for` and `while` loops
  - Useful for working with collections or repeating Sense HAT messages
- **for Loop with range**

```
for i in range(5):
    print(f"Iteration {i + 1}")
```

- **for Loop over a List**

```
temperatures = [12, 18, 25, 30, 15]
for temp in temperatures:
    print(f"Temperature: {temp}°C")
```

Iteration - While Loop

```
count = 0
while count < 3:  # Continues until count reaches 3
    print(f"Count: {count}")
    count += 1
```

Strings

- A string is a sequence of characters
- Can be manipulated with various methods

Example

```
message = "Hello, World!"
print(message.lower())           # hello, world!
print(message.upper())           # HELLO, WORLD!
print(message.replace("World", "Python"))  # Hello, Python!
print(message.split(", "))      # ['Hello', 'World!']
```

Files

- **Uses**
  - Reading and writing data for storage
  - Useful for logging data from Sensors
- **Example**
- Writing to a file (replaces content)

```
with open("temperature_data.txt", "w") as file:
    file.write("Temperature Data\n")
    file.write("12°C\n")
    file.write("18°C\n")
    file.write("25°C\n")
```

Files

- Writing to a file (appending content)

```
with open("temperature_data.txt", "r") as file:
    content = file.read()
    print(content)
```

Lists

- **What are Lists?**
  - Ordered, mutable collections
  - Stores multiple values
- **Examples**

```
temperatures = [12.0,12.5,13.0,15,12]
print(f"Average Temp: {sum(temperatures) / len(temperatures)}C")
print(f"Average Temp: {sum(temperatures) / len(temperatures)}C")
```

```
temperatures = [sense.get_temperature() for _ in range(5)]
print(f"Average Temp: {sum(temperatures) / len(temperatures)}C")
print(f"Average Temp: {sum(temperatures) / len(temperatures)}C")
```

Tuples

- **What are Tuples?**
  - Ordered, immutable collections
  - Useful for storing fixed data
- **Example**

```
coordinates = (10.0, 20.0)
print(f"X: {coordinates[0]}, Y: {coordinates[1]}")
# Trying to modify a tuple will raise an error
 coordinates[0] = 15.0  # This will raise a TypeError
```

Dictionaries

- A data structure used to store data in **key–value pairs**
- a bit like a real dictionary: you look up a word (key) and you get a definition(value)

```
weather = { "temperature": 12, "humidity": 55 }
print(f"Temperature: {weather['temperature']}°C")
print(f"Humidity: {weather['humidity']}%")
weather["pressure"] = 1013  # Adding a new key-value pair
print(f"Pressure: {weather['pressure']} hPa")
```

Networked Programs

- **Networking with Python**
  - Use `requests` or `socket` to fetch data
  - Display fetched data on Sense HAT
- **Example (Using HTTP)**

```
import requests
response = requests.get("http://api.openweathermap.org/data/2.5/weather?q=Tramore&appid=56f0b38c28fc962d4")
data = response.json()
temperature = data["main"]["temp"] - 273.15  # Kelvin to Celsius Conversion
print(f"Tramore Temp: {temperature}C")
print(f"Tramore Temp: {temperature}C")
```

Python Modules

- A module is a file containing Python definitions and code, which can be functions, variables, etc.
- Modules help organise code and premote reuseable functionality across different programs.
- Use the `import` keyword

```
import math

radius = 5
area = math.pi * (radius ** 2)
print(f"Area of circle with radius {radius}: {area}")
```

- Can import specific functions too:

```
from math import sqrt  # Imports only the sqrt function
print(sqrt(16))  # Output: 4.0
```

Pyhton Modules - Custom Modules

- Create your own module by saving a .py file and import it.

  For Example: utils.py

```
#A funtion that converts celcius to fehrenheit
def celsius_to_fahrenheit(cel):
    return (cel * 9/5) + 32
```

Some Other Program:

```
from utils import celsius_to_fahrenheit
temp_c = 25
temp_f = celsius_to_fahrenheit(temp_c)
print(f"{temp_c}°C is {temp_f}°F")  # Output: Hello, Alice!
```

`__main__` in Python

- `__main__` is a special built-in name that refers to the **top-level script** being executed.
- Every Python file has a built-in variable called `__name__`.
- When a file is run directly, `__name__` is set to `"__main__"`. When the file is imported as a module, `__name__` is set to the **module's filename** (without `.py`).
- The conditional statement

```
if __name__ == "__main__":
```

is used to distinguish between:

- Code that should run when the file is executed directly
- Code that should *not* run when the file is imported

`__main__` in Python

For Example: utils.py

```
def celsius_to_fahrenheit(cel):
    return (cel * 9/5) + 32

if __name__ == "__main__":
    # Test the function
    print(celsius_to_fahrenheit(0))    # 32.0
    print(celsius_to_fahrenheit(100))  # 212.0
```

- When imported, `greet(...)` does not run automatically

```
import utils
```

Slide 12: Summary and Questions

- **Summary**
  - Covered Python basics
  - Variables, Conditionals, Functions, Loops, Strings, Files, Lists, Tuples, Networking
- **Questions?**