

Python

*An Introduction using the
Raspberry Pi and SenseHAT*

Frank Walsh

2024

Reference:

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

<https://trinket.io/>

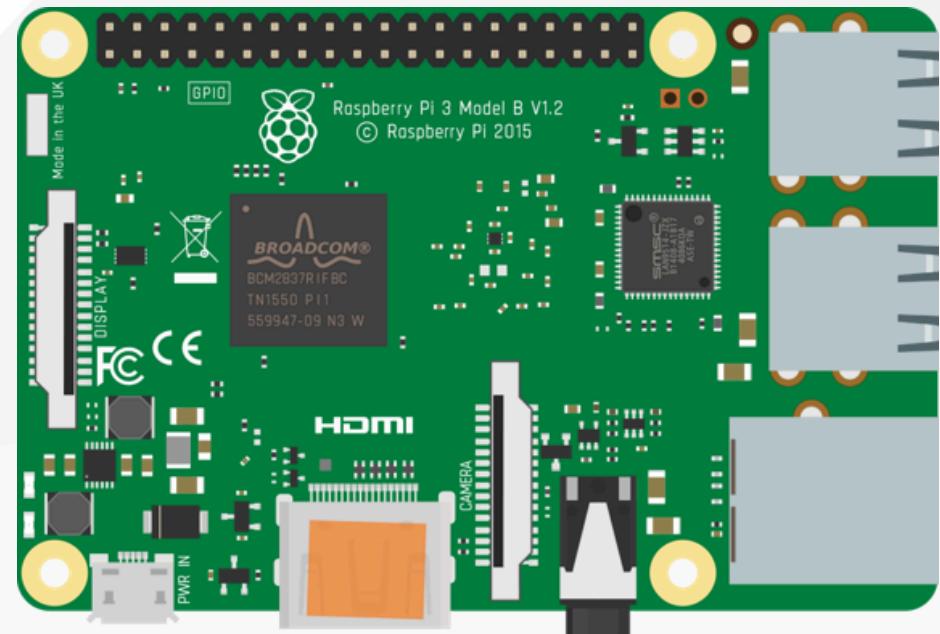


Agenda (goals)

- 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?

- Accessible syntax, large community, powerful libraries, versatility

Installing Python

- No need - it's already on your RPi if you installed Raspberry 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

```
from sense_hat import SenseHat
sense = SenseHat()

temperature = sense.get_temperature()
humidity = sense.get_humidity()

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

```
temperature = sense.get_temperature()

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

```
inp = input('Enter Fahrenheit Temperature: ')
fahr = float(inp)

cel = (fahr - 32.0) * 5.0 / 9.0
print(cel)
```

Functions

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

```
def display_temp_humidity():
    temperature = sense.get_temperature()
    humidity = sense.get_humidity()
    print(f"Temp: {temperature}C, Hum: {humidity}%")

display_temp_humidity()
```

Built-in Functions

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

Example

```
# Using built-in functions with SenseHAT
temperature = sense.get_temperature()
rounded_temp = round(temperature, 1) # Round to 1 decimal place
print(f"Temp: {rounded_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}%")

display_temp_humidity(show_temp=True, show_humidity=False) # Only shows temperature
```

Returning Values from Functions

- Use return keyword to send a value back

```
def get_temp_in_fahrenheit():
    temperature_c = sense.get_temperature()
    temperature_f = (temperature_c * 9/5) + 32
    return temperature_f

temp_f = get_temp_in_fahrenheit()
print(f"Temp: {temp_f}F")
```

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

```
colors = ["Red", "Green", "Blue"]
for color in colors:
    print(f"Color: {color}")
```

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, Sense HAT!"  
first_char=message[0]  
length= len(message)  
reversed_message = message[::-1]  
print(f"""")  
print(reversed_message)
```

Files

- Working with Files
 - Reading and writing data for storage
 - Useful for logging data from Sense HAT
- Example
- Writing to a file (replaces content)

```
with open("sensor_data.txt", "w") as file:  
    temperature = sense.get_temperature()  
    file.write(f"Temperature: {temperature}°C\n")
```

- Writing to a file (appending content)

```
with open("sensor_data.txt", "a") as file:  
    temperature = sense.get_temperature()  
    file.write(f"Temperature: {temperature}°C\n")
```

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

```
RED = (255, 0, 0) # RGB for red  
sense.clear(RED)
```

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=56f1951baca7d4d5030b38c28fc962d4")
data = response.json()
temperature = data["main"]["temp"] - 273.15 # Kelvin to Celsius
print(f"Tramore Temp: {temperature}C")
print(f"Tramore Temp: {temperature}C")
```

Regular Expressions 1

- What are Regular Expressions?
 - Special sequences to match patterns in text
 - Useful for text search, validation, and extraction
- Basic Syntax
 - `\d` matches any digit, `\w` matches any word character, `+` matches one or more, etc.

Regular Expressions 2

- Example: Validating Temperature Format

```
import re
from sense_hat import SenseHat

sense = SenseHat()
temperature = sense.get_temperature()
temp_str = f"{temperature}C" # Format temperature with one decimal place

# Pattern to match temperature format: digits, optional decimal, "C"
pattern = r"^\d+(\.\d)?C$"

if re.match(pattern, temp_str):
    print(f"Valid format: {temp_str}")
else:
    print("Invalid format")
```

Python Modules

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

```
import math # Imports the math module  
  
print(math.sqrt(4)) # Output: 2.0
```

- 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: greeting.py

```
def greet(name):  
    return f"Hello, {name}!"
```

Some Other Program:

```
from greeting import greet  
print(greet("Alice")) # Output: Hello, Alice!
```

`__main__` in Python

- `__main__` is a special built-in variable that represents the name of the top-level script being run.
- `__name__` is set to `"__main__"`, when the main program.
- If the script is imported as a module in another script, `name` is set to the module's name instead.
- The conditional `if __name__ == "__main__":` is used to distinguish between executing a script directly and importing it as a module

__main__ in Python

or Example: greeting.py

```
def greet(name):
    return f"Hello, {name}!"

if __name__ == "__main__":
    print(greet("Frank")) # Only runs if greeting.py is run directly
```

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

```
import greeting
```

Slide 12: Summary and Questions

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