Singapore Polytechnic

DevOps for AIoT

INTRODUCTION TO PYTHON PROGRAMMING

Overview

- Python is very popular high level programming language
- It is an “Interpreted” language and needs to be executed via the Python Interpreter
- There are 2 major versions of Python, Python 2.x and 3.x
- For this module we will use only Python 3.x
- Python is also platform independent and can be run on OS such as Windows, Mac OS, Raspberry Pi OS, etc

Learning Objectives

- Basics of Python as an Interpreted Programming Language
- Introduction to Python coding conventions and data types
- Implement Python code using conditional and logical operators and iterative loops
- Installation and usage of additional 3rd party packages for additional utilities

Singapore Polytechnic

Python Basics – Main Entry Point

- Similar to other programming languages, for Python we need to first define an entry point which the Python Interpreter will first execute
- In Python, we have to define a single entry point similar to the code example below,

```
def main():
      print(“This is the main entry point”)

if __name__ == '__main__':
    main()
```

Python Basics – Functions

- User defined functions allow the Python code to be decomposed into several logical functions
- Breaking down the Python code into individual functions is also a prerequisite for Software Unit Testing which we will cover in the next chapter
- In Python, the access scope of a function is limited to the current Python file the function is located
- However, functions can also be accessed outside of the current Python file by using "imports" which we will also cover in the next sections of this lesson

Python Basics – Functions

- Functions defined in Python always begin with the “def” keyword and end with a semicolon as show in the example below.
- Input parameters are defined without explicit data types similar to other variables declared in Python within a function body

```
def process_temperature(temp):
    if temp >= 80:
        ret_val = "Hot"
    elif temp >= 40 and temp < 80:
        ret_val = "Warm"
    else:
        ret_val = "Cold"

    return ret_val
```

Python Basics – Functions

➢ The example below shows how a function can be defined and invoked in Python

```
def process_temperature(temp):
    if temp >= 80:
        ret_val = "Hot"

    elif temp >= 40 and temp < 80:
        ret_val = "Warm"

    else:
        ret_val = "Cold"

    return ret_val

def main():
    temp_val = process_temperature(60)

    print(temp_val)

if __name__ == '__main__':
    main()
```

Python Data Types and Structures

- Python internally supports several common data structures summarized in the table below,

Data Type	Description
str	String data type
int	Integer numeric data type
float	Float numeric data type
list	List data type
tuple	Tuple data type
Bool	Boolean data type
bytes	Byte data type

- We will discuss in more detail some examples on how to use these data types in different scenarios

Python Data Types and Structures

- Python does not require explicit type declaration and variable types are implicitly inferred by the Python Interpreter based on the initialization values and usage

- To determine exactly which data type is assigned by the Python Interpreter, use the function type(…) as shown in the example below

```
temperature = 38.4
print("Data Type of \"temperature\" = " + str(type(temperature)))

score = 80
print("Data Type of \"score\" = " + str(type(score)))

title = "DevOps for AIoT"
print("Data Type of \"title\" = " + str(type(title)))

names = ["John", "Paul", "Ringgo", "George"]
print("Data Type of \"names\" = " + str(type(names)))
```

Console Output

```
Data Type of "temperature" = <class 'float'>

Data Type of "score" = <class 'int'>

Data Type of "title" = <class 'str'>

Data Type of "names" = <class 'list'>
```

Python Data Types and Structures

- The Python library function type(variable) can returns the data type of the variable assigned by the Python Interpreter
- Notice that we also used another Python library function str(…) to print the data type of the variable
- The str(…) function is another useful Python function that converts a non-String variable into a String data type which can be used for display on the console, etc.

Python Data Types and Structures - Lists

- The “list” data type in Python is very used for processing list of data in an array format
- Using the “list” data type, we can store arrays of data read from a CSV file, data base, or from the console, etc
- In the example below, a list of numbers can be defined below and accessed using a for-loop below

```
number_list = [5, 67, 78, 80, -1]

for number in number_list:
      total = total + number

print(“total = ” + str(total))
```

Python Conditional Statements

- Conditional Statements form the basis of any programming language to evaluate logical conditions
- Python supports the basic “if-elsif-else” conditional statements as shown in the example below,

```
if temp >= 80:
    print("Hot")

elif temp >= 40 and temp < 80:
    print("Warm")

else:
    print("Cold")
```

Python Logical Operators

- Logical Operators are used in several scenarios to evaluate logical conditions together with “if-elif-else”, loops, etc

Logical Operator	Python Logical Operator
AND	and
OR	or
NOT	not

Python Bit-Wise Operators

- Bit-Wise operators are very useful for manipulating binary data
- This is very useful when developing Python code that require low level communication, controlling IO ports, etc on an embedded device such as the Raspberry Pi

Bit-Wise Operator	Python Logical Operator
AND	&
OR	|
Left Shift	<<
Right Shift	>>
NOT	~

Python Bit-Wise Operators - Example

- The example below shows some examples of using Python Bit-Wise operators

```
#Python bit-wise example to set only bit 7 to 1

port_val = 0x00

port_val = port_val | 0x80

print("port_val = " + str(bin(port_val)))
```

Console Output

```
port_val = 0b10000000
```

Python Iterative “for” loop

- Iterative loops form the basis for executing repeated operations in any programming language
- Python provides standard iterative loops such as the “for …” loop in the example below to display the value of an integer variable “i” using a for-loop below
- In this example, we also use the Python range(…) which returns the integer values 0,1,2,3,4 which are then displayed on the console using the print(…) function

```
for i in range(0,5):
      print(i)
```

Console Output

```
0
1
2
3
4
```

Python Iterative “while” loop

- Python also supports the while(..) loop which can be used as an alternative to the for(..) loop

- In the example below, we again iteratively display the integer values 0,1,2,3,4 but using a while(..) loop instead

```
i = 0

while i<5:
      print(i)
      i = i + 1
```

Console Output

```
0
1
2
3
4
```

Importing Packages in Python

- We discussed earlier that access scope of functions in Python is restricted to the Python file in which a function is defined
- However, it’s good practice to always try to decompose our software project into several Python projects and not implement everything into a single huge Python file
- Fortunately Python provides the “import” mechanism which allows functions from other Python files to be imported into another Python file
- After a user defined Python library of file has been imported using the “imported” keyword, all functions in the imported Python file are then accessible

Importing Packages in Python

- ➢ The example below shows how a package can be imported

temperature.py

```
def process_temperature(temp):
    if temp >= 80:
        ret_val = "Hot"
    elif temp >= 40 and temp < 80:
        ret_val = "Warm"
    else:
        ret_val = "Cold"

    return ret_val
```

main.py

```
import temperature as temp

def main():
    temp_val = temp.process_temperature(60)

    print(temp_val)

if __name__ == '__main__':
    main()
```

- ➢ In the file main.py, the temperature.py is imported as the "temp" object and the function "process_temperature()" can then be accessed from main.py

Importing Packages in Python

➢ The example below shows how the Python “import” feature can be used to import other Python files and use the available functions in a different Python file

```
c:\pip3 install pandas
```

Installation of Python 3rd Party Libraries and Packages

- There are several 3rd Party "Package Managers" that support the installation of Python libraries and packages

- We will focus for now on the usage of the "pip" command line tool which is a popular Python package manager that can be used to install additional libraries and packages

- The command line syntax for "pip" is shown in the example below to install the commonly used "pandas" data processing package

```
c:\pip3 install pandas
```

- In this example above, we use the command "pip3" using the Python 3 Interpreter

- Note that an active Internet connection is required for PIP packages on remotely hosted repositories

Useful Python 3rd Party Libraries and Packages

- To supplement the standard Python libraries and packages, there are several open source packages that are useful for data processing, number crunching, etc.
- Some packages that are commonly used are the following,
  - Pandas
    - Provides additional utilities for the processing, analysis and manipulation of data sets
    - Also provides data structures such as the “DataFrame” data structure
    - DataFrames can be used to store data read in from databases, CSV files, etc

CSV ("Comma Separated Value) file

- CSV files or "Comma Separated Values" files are text based files which store datasets in rows and columns
- Similar to an Excel spreadsheet, CSV files represent each dataset as line or row and the fields in each dataset are represented by the columns which are separated by the comma character ','
- An example of a CSV file "weather_data.csv" is show below,

```
date, time, temperature, rel_humidity
18.10.2021, 1530, 33.2, 80
18.10.2021, 1535, 33.2, 82
18.10.2021, 1540, 33.2, 85
18.10.2021, 1545, 33.2, 90
19.10.2021, 1030, 33.2, 78
19.10.2021, 1040, 33.2, 80
19.10.2021, 1059, 33.2, 82
```

Pandas CSV File Reader example

- The Python code below is an example of how the Pandas library can be used to read and process a CSV file,

```
import pandas as pd

df = pd.read_csv('weather_data.csv')

print(df)
```

Console Output

```
date    time    temperature    rel_humidity
0  18.10.2021   1530      33.2       80
1  18.10.2021   1535      33.2       82
2  18.10.2021   1540      33.2       85
3  18.10.2021   1545      33.2       90
4  19.10.2021   1030      33.2       78
5  19.10.2021   1040      33.2       80
6  19.10.2021   1059      33.2       82
```