Python Programming [Introduction]

Siti Mariyah, M.T.

In House Training

Data Science

OJK, 21-22 Mei 2019

Outline

1. Installing Python
2. Interacting with Python
3. Python Syntax
4. Variables and Data Types
5. Basic Operator
6. Conditional Statements
7. Loops
8. User-defined Functions
9. Built-in Functions

Python

- Python is an open source, high-level programming language developed by Guido van Rossum in the late 1980s and presently administered by Python Software Foundation.
- It is a high-level language. Reading and writing codes in Python is much like reading and writing regular English statements.
- Python is an interpreted language.
- Python is an object-oriented language that allows users to manage and control data structures or objects to create and run programs

Advantages of Using Python

- Readability

  Python programs use clear, simple, and concise instructions that are easy to read even by those who have no substantial programming background

- Higher Productivity

  Codes used in Python are considerably shorter, simpler, and less verbose than other high-level programming languages such as Java and C++.

- Less Learning Time

- Runs Across Different Platforms

  Python works on Windows, Linux/UNIX, Mac OS X, other operating systems and smallform devices

Installing Python

- Installing Python in Windows
  - ✓https://www.python.org/downloads/
  - ✓Choose python version (Python 2 or Python 3)

- Installing Python in Mac
  - ✓https://www.python.org/downloads/mac-osx/
  - ✓Choose python version (Python 2 or Python 3)

Interacting with Python

- Python is a flexible and dynamic language that you can use in different ways.
- Command Line Interaction
- Starting Python
  - ✓If you're using Windows, you can start the Python command line by clicking on its icon or menu item on the Start menu.
  - ✓You may also go to the folder containing the shortcut or the installed files and click on the Python command line.
  - ✓If you're using GNU/Linux, UNIX, and Mac OS systems, you have to run the Terminal Tool and enter the Python command to start your session.

Python Syntax

- Python syntax refers to the set of rules that defines how human users and the system should write and interpret a Python program.
- Python keywords are reserved words in Python that should not be used as variable, constant, function name or identifier in your code.

```
and, break, continue, del, else, exec, for, global, import, is , not, pass, raise, try, with, assert, class, def, elif, except, finally, from, if, in, lambda, or, print, return, while, yield.
```

Python Identifiers

- A Python Identifier is a name given to a function, class, variable, module, or other objects that you'll be using in your Python program.
- An identifier can be a combination of uppercase letters, lowercase letters, underscores, and digits (0-9).
- Special characters such as %, @, and $ are not allowed within identifiers.
- An identifier should not begin with a number. Hence, 2variable is not valid, but variable2 is acceptable.
- Python is a case-sensitive language and this behaviour extends to identifiers. Thus, Labor and labor are two distinct identifiers in Python.
- You cannot use Python keywords as identifiers

Python Identifier

- Class identifiers begin with an uppercase letter, but the rest of the identifiers begin in lowercase
- You can use underscores to separate multiple words in your identifier

Python Statements

- my_variable = “cat”
- C = 3

Multi-line Statements

- A statement may over several lines.
- To break a long statement over multiple lines, using backslash.

Indentation

- A block of code can easily identified when you look at a Python program as they start on the same distance to the right.

```
def car_rental_cost(days):
    Cost = 35 * days
    if days >= 8:
        cost -= 70
    elif days >=3:
        cost -= 20
    return cost
```

Comments

- You can write comments within your program by starting the line with a hash (#) symbol.
- For multi-line comments, you can use a hash symbol at the beginning of each line.
- Alternatively, you can also wrap multi-line comment with triple quotes (‘’’ comments here ‘’’)

Variables

- A variable is like a container that stores values that you can access or change.
- It is a way of pointing to a memory location used by a program.
- Python is a lot more flexible when it comes to handling variables. If you need a variable, you'll just think of a name and declare it by assigning a value
- In Python, you declare a variable by giving it a value:

```
>>> my_variable = 10
>>> my_variable = my_variable + 3
>>> print(my_variable)
>>> 13
```

Data types

- Python handles several data types to facilitate the needs of programmers and application developers for workable data.
- These include strings, numbers, Booleans, lists, date, and time.

Strings

```
>>> string1 = ‘I am enclosed in single quotes’
>>> string2 = “I am enclosed in double quotes”
```

- If a literal string enclosed in single quotes contains a single quote, you’ll have to place a backslash (\) before the single quote within the string to escape the character

```
>>> txt = “He said: \” I got a gift \”
>>> print(txt)
>>> He said: “I got a gift”
```

Index in Python

-12	-11	-10	-9	-8	-6	-6	-5	-4	-3	-2	-1
H	e	l	l	o		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

```
>>> s = “Hello Python”
>>> s[0]
>>> H
>>> s[-2]
>>> o
```

Concatenation Strings

- Strings can be added together with plus (+) operator. To concatenate the string “Hello Python”:

```
>>> “Hello” + “Python”
HelloPython
```

Repeating Strings

- You can easily repeat strings or its concatenation with the * operator.

```
>>> s = “**^**”
>>> s * 5
**^**^**^**^**^**^**^**^**^**
```

Getting the Size of Strings

- You can get the size of a string with the len() function

```
>>> len(“World”)
5
```

Slicing Strings

- You can create substrings with the slicing notation. You can do this by placing two indices (separated by a colon) within square brackets.

```
>>> “Program”[3:5]
gr
>>> “Program”[3:6]
gra
```

The lower(), upper(), and str() function

- You can use the `lower()` function to print the string in lower case and `upper()` to print in upper case.

```
>>> c = “Grand River”
>>> print(c.lower())
grand river
>>> print(c.upper())
GRAND RIVER
>>> pi = 3.1416
>>> str(pi)
“3.1416”
```

Numbers (Numeric Data Types)

- One of the many conveniences of using Python is that you don’t really have to declare a numeric value to distinguish its type.
- Python can readily tell one data type from another when you write and run your statement.
- Python 3 supports three types: integer, floating-point numbers, and complex numbers.

Integer (int)

Integers are whole numbers without decimal point. They can be positive or negative.

E.g. 793, -254, 5

Numbers (Numeric Data Types)

Octo literals (base 8)

To indicate an octal number, you will use the prefix 0o or 0O (zero followed by either a lowercase or uppercase letter ‘o’)

```
>>> a = 0O7
>>> print(a)
7
```

Hexadecimal literals (base 16)

To indicate hexadecimal literals, you will use the prefix ‘0X’ or ‘0x” (zero and uppercase or lowercase letter ‘x’).

```
>>> hex_lit = 0xA0C
>>> print(hex_lit)
2572
```

Converting Integers to their String Representation

- To convert an integer into its string representation, you can use the functions hex(), bin(), and oct().

```
>>> oct(7)
‘0o7’
>>> hex(2572)
‘0xa0c’
>>> bin(12)
‘0b1100’
```

Floating-point numbers

- Floats are written with a decimal point that segregates the integer from the fractional numbers.
- They may also be written in scientific notation where the uppercase or lowercase letter ‘e’ signifies the 10^{th}

```
>>> 6.2e3
6200.0
>>> 6.2e2
620
```

Date and Time

- In Python, you can use the function `datetime.now()` to retrieve the current date and time.
- The command `datetime.now()` calls on a built-in Python code which gives the current date and time.

```
>>> from datetime import datetime
>>> datetime.now()
datetime.datetime(2016,3,10,2,16,19,962429)
>>> from time import strftime
>>> strftime(“%Y-%m-%d %H:%M:%S”)
‘2016-03-10 02:20:03’
```

Boolean Data Types

- Comparisons in Python can only generate one of two possible responses: True or False. These data types are called booleans.

```
>>> bool_1 = 4 == 2*3
>>> bool_2 = 10 < 2 * 2**3
>>> bool_3 = 8 > 2 * 4 + 1
>>> print(bool_1)
>>> print(bool_2)
>>> print(bool_3)
```

Lists

- A list is a data type that can be used to store any type and number of variables and information.
- You can define and assign items to a list with the expression:

```
my_list = [item_1, item_2, item_3]
```

- Python also allows creation of an empty list:

```
my_list = []
```

```
>>> colors = [“red”, “orange”, “yellow”, “green”, “indigo”, “white”]
>>> print(colors[0])
red
>>> len(colors)
6
```

Lists

```
>>> print(colors)
['red', 'orange', 'yellow', 'green', 'indigo', 'white']
>>> colors.remove("white")
>>> print(colors)
['red', 'orange', 'yellow', 'green', 'indigo']
>>> colors.append("violet")
>>> print(colors)
['red', 'orange', 'yellow', 'green', 'indigo', 'violet']
>>> colors.insert(4, "blue")
>>> print(colors)
['red', 'orange', 'yellow', 'green', 'blue', 'indigo',
'violet']
```

Slicing lists

- You can also slice lists in the same way that you slice strings.

```
>>> colors[3:6]
['green', 'blue', 'indigo']
```

Dictionary

- A dictionary is like a list but instead of looking up an index to access values, you'll be using a unique key, which can be a number, string, or tuple.
- Dictionary values can be anything but the keys must be an immutable data type.
- A colon separates a key from its value and all are enclosed in curly braces. Here is the dictionary structure:

```
d = {key_1 : a, key_2 : 2, key_3 : ab}
```

- An empty dictionary will have this format:

```
d = {}
```

Dictionary

```
>>> menu = {“spam”:12.50, “carbona”:20, “salad”:15}
>>> len(menu)
3
>>> print(menu)
{‘spam’:12.50, ‘carbona’:20, ‘salad’:15}
```

To add another entry in the menu dictionary, you can use this format:

`d[key_4 : b]`

```
>>>menu[“cake”] = 6
>>> print(menu)
{‘spam’:12.50, ‘carbona’:20, ‘salad’:15, ‘cake’:6}
```

Dictionary

- Assuming you no longer want to include spam in your menu, you can easily do so with the **del** command:

```
>>> del menu[“spam”]
{‘cake’: 6, ‘carbonara’: 20, ‘salad’: 15}
```

- You might want to change the values in any of the keys at one point. For instance, you need to change the price of carbonara from 20 to 22.

```
>>> menu[“carbonara”] = 22
>>> print(menu)
{‘cake’: 6, ‘carbonara’: 22, ‘salad’: 15}
```

Dictionary

- ✓If you want to remove all entries in the dictionary, you can use the function **dict.clear()**

```
>>> dict.clear(menu)
>>> print(menu)
{}
```

- ✓If you want to delete the dictionary, you can use the function **del**

```
>>> del menu
>>> print(menu)
Traceback (most recent call last):
File “<pyshell#19>”, line 1, in <module>
print(menu)
NameError: name ‘menu’ is not defined
```

Conditional Statements

- If-then-else statements or conditional expressions are essential features of programming languages and they make programs more useful to users.
- The if-then-else statement in Python has the following basic structure:

```
if condition1:
    block1_statement
elif condition2:
    block2_statament
else:
    block3_statement
```

Conditional Statements

```
>>>
def your_choice(answer):
  if answer > 5:
     print(“You are overaged.”)
  elif answer <= 5 and answer >1:
     print(“Welcome to the
Toddler’s Club!”)
  else:
     print(“You are too young for
Toddler’s Club.”)
>>> print(your_choice(6))
>>> print(your_choice(3))
>>> print(your_choice(1))
>>> print(your_choice(0))
```

- Output:
- You are overaged.
- None
- Welcome to the Toddler’s Club!
- None
- You are too young for Toddler’s Club.
- None
- You are too young for Toddler’s Club.
- None

Loop

- A loop is a programming construct that enables repetitive processing of a sequence of statements.
- Python provides two types of loops to its users: the ‘for loop’ and the ‘while loop’.
- The ‘for’ and ‘while’ loops are iteration statements that allow a block of code (the body of the loop) to be repeated a number of times.

For Loop

- Python implements an iterator-based ‘for loop’. It is a type of ‘for loop’ that iterates over a list of items through an explicit or implicit iterator.
- The loop is introduced by the keyword ‘for’ which is followed by a random variable name which will contain the values supplied by the object.

```
for variable in list:
   statements
else:
   statements
```

For Loop

```
>>> pizza = [“New York Style Pizza”, “Pan Pizza”, “Thin n Crispy Pizza”, “Stuffed Crust Pizza”]
>>>for choice in pizza:
    if choice == “Pan Pizza”:
        print(“Please pay $16. Thank you!”)
        print(“Delicious, cheesy ” + choice)
    else:
        print(“Cheesy pan pizza is my all-time favorite!”)
    print(“Finally, I’m full!”)
```

Using a break statement

- A Python break statement ends the present loop and instructs the interpreter to starts executing the next statement after the loop. It can be used in both ‘for’ and ‘while’ loops.
- Besides leading the program to the statement after the loop, a break statement also prevents the execution of the ‘else’ statement.

Using a break statement

```
>>> pizza = [“New York Style Pizza”, “Pan Pizza”, “Thin n Crispy Pizza”, “Stuffed Crust Pizza”]
>>>for choice in pizza:
     if choice == “Pan Pizza”:
          print(“Please pay $16. Thank you!”)
          break
          print(“Delicious, cheesy ” + choice)
     else:
          print(“Cheesy pan pizza is my all-time favorite!”)
     print(“Finally, I’m full!”)
```

Using continue statement

- The continue statement brings back program control to the start of the loop. You can use it for both ‘for’ and ‘while’ loops.

```
>>> pizza = [“New York Style Pizza”, “Pan Pizza”, “Thin n Crispy Pizza”, “Stuffed Crust Pizza”]
>>>for choice in pizza:
     if choice == “Pan Pizza”:
          print(“Please pay $16. Thank you!”)
          continue
          print(“Delicious, cheesy ” + choice)
     else:
          print(“Cheesy pan pizza is my all-time favorite!”)
     print(“Finally, I’m full!”)
```

Using the range() with the for loop

- The `range()` function can be combined with the ‘for loop’ to supply the numbers required by the loop.
- In the following example, the range(1, x+1) provided the numbers 1 to 50 needed by the ‘for loop’ to add the sum of 1 until 50

```
>>> x = 50
>>> total = 0
>>> for number in range(1, x+1):
        total = total + number
>>> print(“Sum of 1 until %d: %d” % (x, total))
Sum of 1 until 50: 1275
```

The While Loop

- A Python ‘while loop’ repeatedly carries out a target statement while the condition is true.
- The loop iterates as long as the defined condition is true. When it ceases to be true and becomes false, control passes to the first line after the loop.
- The ‘while loop’ has the following syntax:

```
while condition:
    statement
```

The While Loop

```
counter = 0
while (counter < 10):
    print('The count is:' , counter)
    counter = counter + 1
Print("Done")
```

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
The count is: 9
Done
```

User-defined Functions

- A function is a set of statements that perform a specific task, a common structuring element that allows you to use a piece of code repeatedly in different parts of a program.
- The use of functions improve a program's clarity and comprehensibility and makes programming more efficient by reducing code duplication and breaking down complex tasks into more manageable pieces.
- Functions are also known as routines, subroutines, methods, procedures, or subprograms.
- They can be passed as arguments, assigned to variables, or stored in collections.

A user-defined Python Function

```
def function_name(parameter list):
    function body/statements
```

- Function bodies can have more than one return statement which may be placed anywhere within the function block.
- Return statements end the function call and return the value of the expression after the return keyword.
- A return statement with no expression returns the special value ‘None’.
- In the absence of a return statement within the function body, the end of the function is indicated by the return of the value ‘None’.

A user-defined Python Function

```
>>> def love_pizza():
        print(“I love Pizza!”)
>>> love_pizza()
‘I love Pizza’
>>> def absolute_value(number):
      if number >= 0:
           return number
      else:
           return –number
>>> print(absolute_value(3))
>>> print(absolute_value(-5))
```

A user-defined function with an if-then-else

```
def shutdown(yn):
  if yn.lower() == “y”:
     return(“Closing files and shutting down”)
  elif yn.lower() == (“n”):
     return(“Shutdown cancelled”)
  else:
     return(“Please check your response.”)
print(shutdown(“y”))
print(shutdown(“n”))
print(shutdown(“x”))
```

Functions can call other functions

- Functions can perform different types of actions such as do simple calculations and print text. They can also call another function.

```
>>> def members_total(n):
        return n * 3
>>>  def org_total(m):
        return members_total(m) + 5
>>> print(org_total(2))
>>> print(org_total(5))
>>> print(org_total(10))
```

Class and Object-Oriented

- Python is an object-oriented programming language, which means that it manipulates and works with data structures called objects.
- Objects can be anything that could be named in Python – integers, functions, floats, strings, classes, methods, etc.
- They can be used anywhere an object is required. You can assign them to variables, lists, or dictionaries.
- They can also be passed as arguments.
- Every Python object is a class.
- A class is simply a way of organizing, managing, and creating objects with the same attributes and methods.

Class Syntax

- To define a class, you can use ‘class’, a reserved keyword, followed by the class name and a colon.
- By convention, all classes start in uppercase.

```
class Students:
    pass
```

To create a class that takes an object:

```
class Students(object)
```

The __init__() method

- Immediately after creating an instance of the class, you have to call the __init__() function.
- This function initializes the objects it creates. It takes at least the argument ‘self’.
- a Python convention, which gives identity to the object being created.

```
class Students:
    def __init__(self):

class Employees(object):
    def __init__(self, name, rate, hours)
```

The __init__() method

- A function used in a class is called a method. Hence, the __init__() function is a method when it is used to initialize classes.

Instance Variables

- When you add more arguments to the def_init_() besides the self, you'll need to add instance variables so that any instance object of the class will be associated with the instance you create.

```
class Employees(object):
    def __init__(self, name, rate, hours) :
    name.self = name
    rate.self = rate
    hours.self =hours
```

- In the above example, name.self, rate.self, and hours.self are the instance variables.

Instance Variables

- When you create instances of the class Employees, each member will have access to thevariables which were initialized through the __init__ method.
- To illustrate, you can create or ‘instantiate’ new members of the class Employees:

```
>>> staff = Employees(“Wayne”, 20, 8)
>>> supervisor = Employees(“Dwight”, 35, 8)
>>> manager = Employees(“Melinda”, 100, 8)
>>> print(staff.name, staff.rate, staff.hours)
>>> print(supervisor.name, supervisor.rate, supervisor.hours)
>>> print(manager.name, manager.rate, manager.hours)
```

Python’s Built-in Function

- Functions provide efficiency and structure to a programming language.
- Python has many useful built-in functions to make programming easier, faster, and more powerful.

The input() function

- Programs usually require input that can come from different sources: keyboard, mouse clicks, database, another computer's storage, or the internet.
- Since the keyboard is the most common way to gather input, Python provided its users the input() function.
- This function has an optional parameter called the prompt string

The input() function

```
>>>name = input(“May I know your name?”)
>>>print(“It’s pleasure to meet you” + name)
>>>age = input(“Your age, please?”)
>>>print(“So, you’re ” + age + ” years old, ” + name
+ “!”)
```

The range() function

- Python has a more efficient way to handle a series of numbers and arithmetic progressions and this is by using one its built-in functions: **`range()`**

```
>>> range(5)
range(0,5)
>>> list(range(5))
[0,1,2,3,4]
>>> range(5,9)
>>> list(range(5,9))
[5,6,7,8]
>>> list(range(10,71,5))
[10,15,20,25,30,35,40,45,50,55,60,65,70]
```

The abs() function

- The **abs()** function returns the absolute value of a single number.
- It takes an integer or float number as argument and always returns a positive value.

```
>>> abs(-10)
10
```

The max() function

- The **max()** function takes two or more arguments and returns the largest one.

```
>>> max(9,12,6,15)
15
>>> max(-2,-7,-35,-4)
-2
```

Type() function

- The type() function returns the data type of the given argument.

```
>>> type(“This is string”)
<class ‘str’>
>>> type(12)
<class ‘int’>
```

Len() function

- The **len()** function returns the length of an object or the number of items in a list given as argument.

```
>>> len(“pneumonoultramicroscopicsilicovolcanoconiosi”)
44
>>> s = (“winter”, “spring”, “summer”, “fall”)
>>> len(s)
4
```