동국대학교
dongguk university

Python Programming Basic

Machine Learning

Python Programming Basic

- Contents
  - Comments
  - Constants, Variables
  - Conditional Statements
  - Iteration Statements
  - Function
  - Container : List, Dictionary, Tuple, Set
  - File
  - Class
  - **Quiz**

Comments

- Some sentences to explain the codes
- Comments are not included in the codes

Comments

C, C++

```
// comment
/* multi-line comments */
```

Python

```
# comment
''' multi-line comments '''
```

```
# Type Comments
'''
This is a test
'''
print("hello") # print a string
```

```
hello
```

Constants

- Fixed values(unchangeable numbers, strings etc)
- Datatypes
  1. Numbers
     - Integer: -2, 0, 100
     - Real numbers: -2.5, 0.0, 99.9
  2. String
     - Single/double quotation marks(‘ or “)
     - ‘hello’, “hello”
  3. Logical
     - True, False
  4. Special
     - None

```
print(1 + 2) # integer
print(1.0 / 3.0) # real number(float)
```

```
3
0.3333333333333333
```

```
# print string "Machine", "Learning"
print("Machine", "Learning")
```

```
Machine Learning
```

```
print(True, False) # Boolean
```

```
True False
```

```
print(None) #Special
```

```
None
```

Constants

- Data Type Check : type()

```
type(100) # int
```

```
int
```

```
type("Machine Learning") # str
```

```
str
```

- Casting

```
int(10.4) # float to int
```

```
10
```

```
str(10.4) # float to string
```

```
10.0
```

- Dynamic Typing

```
# Operation
type(10 + 0.1) # float
```

```
float
```

```
# Comparison
10 == 10.0 # True
```

```
True
```

Variables

- Named memory to store values
  - Name consist of alpha-numeric characters and underscores (A-z, 0-9, and _)
  - Name cannot start with a number

```
x = 20
y = 30
z = x + y
print(z) # print z
type(z) # data type of z
```

```
50
int
```

```
s1 = "Hello "
s2 = "world!"
print(s1+s2) # print s1 + s2
type(s1+s2) # data type of z
```

```
Hello world!
```

Operators	Operation
+	Plus
-	Minus
*	Multiply
/	Division
**	Power 2
%	Remainder

Operator priority

Bracket > power 2 > multiply > plus, minus

Variables

- Basic operation of string
  - Multiplication(*)

```
# string multiplication
line = "=" * 20
print(line)
```

```
====================
```

  - Indexing

```
# Indexing
ml = "Machine Learning"
print(ml[0])
print(ml[-1])
```

```
M
g
```

  - len()

```
# len()
print(len(ml))
```

```
16
```

  - .split()

```
# split()
ml_list = ml.split(" ")
print(ml_list)
```

```
['Machine', 'Learning']
```

Conditional Statements

- If
  - If statement is control flow statements which helps us to run a particular code only when a certain condition is satisfied.

```
# Convert user input to integer
x = int(input())

# Conditional statements
if x < 5 :
    print('small')
elif x < 15 :
    print('medium')
else :
    print('large')
```

```
15
large
```

Comparison operator	Means
<	Smaller
<=	Smaller or equal
==	Same
>=	Larger or equal
>	Larger
!=	Not equal

Note.

In Python, blocks are marked as indentations in functions, iteration statements and conditional statements, etc. It is strongly recommended that you do not use the tab

Conditional Statements

- Boolean Operator
  1. and (&)

```
# and
(1 == 1) and (1 == 0)
```

```
False
```

  2. or (|)

```
# or
(1 == 1) or (1 == 0)
```

```
True
```

  3. not

```
# not
not (1 == 0)
```

```
True
```

Iteration Statements

- while
  - The while statement enables the execution of a body of statements multiple times in a loop while a certain condition is true.

```
# while statement
n = 5
while n > 0 :
    print(n)
    n = n - 1
```

```
5
4
3
2
1
```

```
# infinite Loop with break statement
n = 5
while True :
    if n <= 0:
        break
    print(n)
    n = n - 1
```

```
5
4
3
2
1
```

Iteration Statements

- for
  - for statement iterates over the items of any sequence (a list or a string), in the order that they appear in the sequence.
  - list : a compound data types like an array in C (will be addressed)

```
# sum of all numbers from 1 to 10
sum_all = 0
mylist = [1,2,3,4,5,6,7,8,9,10]

for i in mylist:
    sum_all = sum_all + i
print(sum_all)  # 55
```

```
55
```

  - range(n) : it returns a list of numbers 0 to n-1

```
# sum of all numbers from 0 to 9
sum_all = 0

for i in range(10):
    sum_all = sum_all + i
print(sum_all)  # 45
```

```
45
```

Function

- A named section of a program that performs a specific task. It is reusable code
  - Bulit-in function(refer to the link below)
    - print(), input(), …
  - User-defined function
    - def function_name(arg_1, arg_2, …) :

      # algorithms

      return something

```
# Function to convert km to mile
def km_to_mile(km):
    mile = km/1.6 # 1 mile == 1.6 km
    return mile

print('Enter the distance in km scale.')
km = float(input("Input km : "))
print('It is', km_to_mile(km), 'mile')
```

```
Enter the distance in km scale.
Input km : 42
It is 26.25 mile
```

https://docs.python.org/3/library/functions.html

Container

- Container
  - Data structure for storing variables
- In python there are four data structures - lists, dict, tuple, and set
  - list [,]
  - dict { , }
  - tuple (,)
  - set { , }
- When programming in python, it is important to understand the characteristics of the four containers and write the appropriate data structures for your situation

https://docs.python.org/3/tutorial/datastructures.html

List

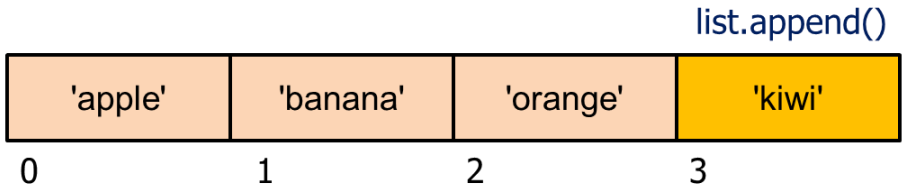
- A list is a data structure that stores ordered sequence of items
  - It can be initialized [item, item, …]
  - It can be initialized by the function ‘list()’ and then append item by the function ‘list.append(item)’

```
# make the list named'fruits'
fruits = [ 'apple', 'banana', 'orange' ]
fruits[0]
```

```
'apple'
```

```
# append 'kiwi' to the list named 'fruits'
fruits.append('kiwi')
fruits
```

```
['apple', 'banana', 'orange', 'kiwi']
```



https://docs.python.org/3/tutorial/datastructures.html

List

- Index of list
  - n:m - it means from ‘n’ to ‘m-1’
  - n:m:s : it means from 'n' to 'm-1' with step size 's'

```
# Indexing & Slicing (1)
mylist = [10, 20, 30, 40, 50 ]
print(mylist[1])   # 20
print(mylist[1:3]) # [20, 30]
print(mylist[:3])  # [10, 20, 30]
print(mylist[3:])  # [40, 50]
```

```
20
[20, 30]
[10, 20, 30]
[40, 50]
```

```
# Indexing & Slicing (2)
print(mylist[-1])   # 50
print(mylist[::2])  # [10, 30, 50]
print(mylist[::-1]) # [50, 40, 30, 20, 10]
```

```
50
[10, 30, 50]
[50, 40, 30, 20, 10]
```

- Functions with the list as a parameter and methods of the list

```
# functions & methods
numbers = [20, 50, 10, 40, 30]
print(len(numbers))
print(max(numbers))
print(min(numbers))
print(sum(numbers))
numbers.sort()
print(numbers)
```

```
5
50
10
150
[10, 20, 30, 40, 50]
```

https://docs.python.org/3/tutorial/datastructures.html

List

- List with for statement

```
# make a list with all even numbers from 0 to 9 (1)
even_numbers = []
for i in range(10):
    if i % 2 == 0:
        even_numbers.append(i)
print(even_numbers)
```

```
[0, 2, 4, 6, 8]
```

- List comprehension

```
# make a list with all even numbers from 0 to 9 (2)
even_numbers = [i for i in range(10) if i % 2 == 0]
print(even_numbers)
```

```
[0, 2, 4, 6, 8]
```

```
# List comprehension & function
def trans(x):
    y = 2*x + 1
    return y
trans_numbers = [ trans(i) for i in range(10)]
print(trans_numbers)
```

```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

https://docs.python.org/3/tutorial/datastructures.html

List

- Nested list

```
# Nested list
odd_numbers = [ i for i in range(10) if i % 2 == 1]
even_numbers = [ i for i in range(10) if i % 2 == 0]
nested_list = [odd_numbers, even_numbers]
print(nested_list)
```

```
[[1, 3, 5, 7, 9], [0, 2, 4, 6, 8]]
```

```
# Index of nested list
print(nested_list[0]) # first row
print(nested_list[1]) # second row
print(nested_list[0][0])# 1st element of first row
print(nested_list[1][2])# 3rd element of second row
```

```
[1, 3, 5, 7, 9]
[0, 2, 4, 6, 8]
1
4
```

https://docs.python.org/3/tutorial/datastructures.html

Dictionary

- A dictionary is a data structure that stores unordered <key : value> pairs
  - It can be initialized as { key : value , key : value , … } or
  - Make empty dictionary using 'dict()', and then insert the new (key, value) pair
  - List : items are indexed by location(integer)
  - Dictionary : values are indexed by keys

```
# List
num_list = [ 30 , 10 , 20 ]
print(num_list[2])
```

```
20
```

```
# Dictionary
word_count = { 'red' : 30 , 'blue' : 10 , 'yellow' : 20 }
print(word_count['red'])

word_count['green'] = 5
print(word_count)
```

```
30
30
{'red': 30, 'blue': 10, 'yellow': 20, 'green': 5}
```

Dictionary

- methods of the dictionary

1. .items()

```
word_count.items()
```

```
dict_items([('red', 30), ('blue', 10), ('yellow', 20), ('green', 5)])
```

2. .keys()

```
word_count.keys()
```

```
dict_keys(['red', 'blue', 'yellow', 'green'])
```

3. .values()

```
word_count.values()
```

```
dict_values([30, 10, 20, 5])
```

Dictionary

- list vs. dictionary

```
# List
lst = list()
lst.append(185)
lst.append(78)
print(lst)
print(lst[0])
```

```
[185, 78]
185
```

List

index	Item
[0]	185
[1]	78

```
# Dictionary
my_dict = dict()
my_dict['height'] = 185
my_dict['weight'] = 78
print(my_dict)
print(my_dict['height'])
```

```
{'height': 185, 'weight': 78}
185
```

Dictionary

Key	Value
['height']	185
['weight']	78

Dictionary

- Dictionary with for statement

```
# make 'vocab' dictionary from a sentence (1)
vocab_list = "Life is like a box of chocolates".split(" ")
vocab = {}
for i, v in enumerate(vocab_list):
    vocab[i] = v
print(vocab)
```

```
{0: 'Life', 1: 'is', 2: 'like', 3: 'a', 4: 'box', 5: 'of', 6: 'chocolates'}
```

- Dictionary Comprehension

```
# make 'vocab' dictionary from a sentence (2)
vocab = { k : v for k, v in enumerate(vocab_list)}
print(vocab)
```

```
{0: 'Life', 1: 'is', 2: 'like', 3: 'a', 4: 'box', 5: 'of', 6: 'chocolates'}
```

Tuple

- A tuple is a collection of items which is ordered and **unchangeable**.
  - It is more simple and efficient because the tuple is saved with the condition not to be changed in the future
  - 'items()' method of dictionary returns list of (key, value) tuples

```
# tuple
x = (20, 30)
y = (20,)

print(x)
print(y)
```

```
(20, 30)
(20, )
```

```
# tuple and dictionary
ids = dict()
ids['tom'] = 1234
ids['john'] = 5678

for (key, val) in ids.items():
    print(key, val)
print(ids.items())
```

```
tom 1234
john 5678
dict_items([('tom', 1234), ('john', 5678)])
```

Set

- A set is an unordered collection of items with no duplicates
  - It can be initialized as { item1, item2, … } or
  - It can be defined by the function 'set()'

```
# Set
s = {1, 2, 3}
print(s)

# Ignore duplicates
lst = [1, 2, 2, 3, 3, 4]
set1 = set(lst)
print(set1)

# Ignore order
strings = "Hello"
set2 = set(strings)
print(set2)
```

```
{1, 2, 3}
{1, 2, 3, 4}
{'e', 'l', 'H', 'o'}
```

Set

- Set operations :
  - .intersection() (&)
  - .union() (|)
  - .difference() (-)

```
# intersection, union, difference
set1 = set([1, 2, 3, 4])
set2 = set([2, 4, 6, 8])

# intersection
print(set1 & set2)              # using &
print(set1.intersection(set2)) # using intersection function

# union
print(set1 | set2)              # using |
print(set1.union(set2))         # using union function

# difference
print(set1 - set2)              # using using -
print(set1.difference(set2))    # using difference function
```

```
{2, 4}
{2, 4}
{1, 2, 3, 4, 6, 8}
{1, 2, 3, 4, 6, 8}
{1, 3}
{1, 3}
```

File processing

- Opening the file
  - open() – it returns the handle used to manipulate the file
  - The file handle is a sequence of each line of the file

```
# Open 'sample.txt' file and count lines
fhand = open("sample.txt")
count = 0
for line in fhand:
    count = count + 1
fhand.close()
print(count)
```

```
27
```

- Read all contents of the file
  - .read() – it reads all contents in the file and returns it as a sequence of a string

```
# Open & close
fhand = open("sample.txt")
mfile = fhand.read()
print(len(mfile))
fhand.close()
```

```
9483
```

```
# Open & with
with open('sample.txt') as f:
    mfile = f.read()
print(len(mfile))
```

```
9483
```

Class

- Object = Attribute + Method
  - Classes provide a means of bundling data and functionality together
  - Creating a new class creates a new *type* of object, allowing new *instances* of that type to be made
- Creating a class

```
# Make 'Car' class
class Car:
    # Constructor
    def __init__(self, brand, color, year, current_speed):
        self.brand = brand
        self.year = year
        self.current_speed = current_speed
    # Method
    def accelerate(self):
        self.current_speed += 10
        return self.current_speed
```

"self" represents the instance of the class. By using the "self" keyword we can access the attributes and methods of the class

brand, year, current_speed are attributes

accelerate is a method

Class

- Creating objects

```
# Make 'car1', 'car2' objects
car1 = Car('toyota', 'red', 1995, 100)
car2 = Car('hyundai', 'green', 2000, 120)
```

- Attributes & methods

```
# Print attributes, execute the method
print(car1.brand)
print(car2.current_speed)
car2.accelerate()
print(car2.current_speed)
```

```
toyota
120
130
```

Submit

- To make sure if you have completed this practice,
  Submit the your practice file(Wee02_givencode_python_tutorial.ipynb) to e-class.
- Due tomorrow 11:59pm
- Modify your ipynb file name as “Week02_StudentNum_Name.ipynb”
  Ex) Week02_2020123456_홍길동.ipynb
- You may upload the file without the quiz codes, but it is recommended to solve the quiz as well to prepare for the homeworks and the exam.

Quiz 1

- input, while, if
  - Write a program that continuously inputs numbers(positive integers), and outputs the total counts, sum, and average of the integers.
  - The program ends when the input is 0.

```
Enter a number : 2
Enter a number : 3
Enter a number : 9
Enter a number : 4
Enter a number : 0
Total = 4
Sum = 18
Average = 4.5
```

Quiz 2

- function, for, list
  - The temperature (temp) and iced c sales (sale) have the following formula.
    - $sale = 30.08 * temp - 159.47$
  - Define a function to compute above equation
  - For following list of temps predicted for the next 10 days, make the list of sales(integers) for the next 10 days

```
temps = [11.9, 14.2, 15.2, 16.4, 17.2, 18.1, 18.5, 19.4, 22.1, 22.6]
```

  - Print the sales and average of them

```
Sales = [198, 267, 297, 332, 357, 384, 396, 423, 504, 519]
Average = 367.7
```

Quiz 3

- Dictionary
  - Change the scores in following dictionary into 'P' (if score >= 70) or 'F' and print the result

```
scores = {
    'Tom':60,
    'Jane':95,
    'Bill':80,
    'Mary':55
}
```

```
{'Tom': 'F', 'Jane': 'P', 'Bill': 'P', 'Mary': 'F'}
```

Quiz 4

- Class
  - Create a 'Circle' class that has an attribute 'radius', and a method 'area()' that computes the area of the circle
  - Create a 'Rectangle' class that has attributes 'width' and 'height', and a method 'area()' that computes the area of the rectangle
  - Create an instance for each class, and print out the area of each instance.
  - Set π to 3.14.

```
circle.area()
```

```
314.0
```

```
rectangle.area()
```

```
600
```