Python Basic 2022

Jongyeob Park

2022-02-15

Introduction

- Why python?
  - Readable code
  - Fast prototyping
  - Glue language
- Python + Scipy + Numpy + Matplotlib
- Anaconda: https://www.anaconda.com/
- IDEs: spyder, ipython + vim or emacs, PyCharm, vscode, eclipse-pydev
- Useful links:
  - https://docs.python.org/tutorial
  - https://docs.python.org/library
  - http://www.scipy-lectures.org/
- Book: The Python Standard Library by Example
  (예제로 배우는 파이썬 표준 라이브러리)

IDE: Integrated development environment

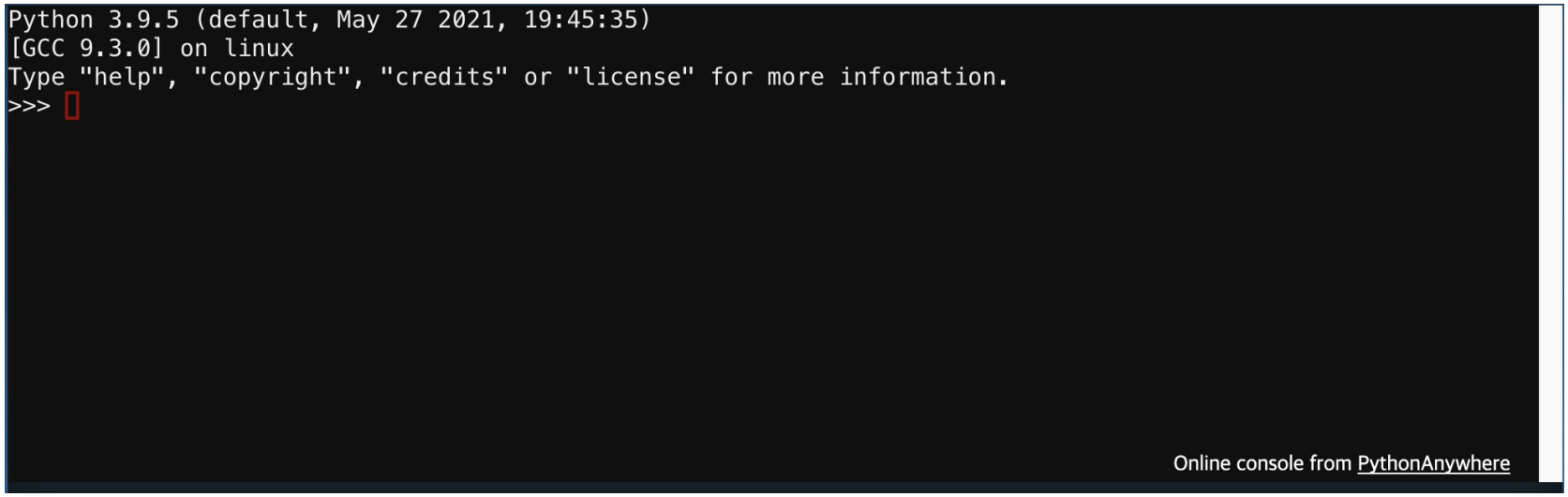
Python Basic

- Coding Rule
- Datatype
- Control Flow
- Function and Class
- Module and Packages
- Text Handling
- Time Handling
- File Handling
- File System Handling
- Internet
- Parallelism

Invoking Python

```
$ python
```

https://www.python.org/shell/



1) print("Hello World")

```
>>> exit()
or
>>> Ctrl + d (Linux)
>>> Ctrl + z (Windows)
```

Basic Syntax

```
1) print("Hello, \
2) World!")

3) print('Hello'); print("World!")

4) # Comments

5) if True:
6)     print("True")
7) else:
8)     pass

9) if True: print("True")
10)else: print("False")
```

They can be enclosed in single quotes ('...') or double quotes ("...") with the same result.

Pass can be used when a statement is required syntactically but the program requires no action.

Variables

```
1) 2 + 2
2) print(_)

3) 50 - 5*6
4) (50 - 5*6) / 4
5) 8 / 5
6) 17 / 3
7) 17 // 3
8) 17 % 3
9) 5 ** 2
10)2 ** 7

11)width = 20
12)height = 5
13)height *= 9
14)width * height
```

The last printed expression is assigned to the variable _.

To do floor division and get an integer result (discarding any fractional result)

Variable Case Sensitive

function and variable:
    snake_code
    lowerCamelCode
Class:
    UpperCamelCode

```
height = height * 9
```

Strings (1)

```
1) 'spam eggs'
2) 'doesn\'t'
3) "doesn't"

4) '"Yes," they said.'
5) "\"Yes,\" they said."

6) s = 'First line.\nSecond line.'

7) print(s)

8) print('C:\some\name')
9) print(‘C:\\some\\name')

10)print(r'C:\some\name')
```

\n means newline

print() function produces a more readable output.

You can use raw strings by adding an r before the first quote.

Strings (2)

```
1) print("""\
2) One
3) Two""")


4) 'S' + 'u'* 3 + 'per'
5) 'Py' 'thon'


6) word = 'Python'
7) word[0]
8) word[5]


9) word[-1]
10)word[-6]
```

Strings can be indexed (subscripted), with the first character having index 0.

```
 +---+---+---+---+---+---+
 | P | y | t | h | o | n |
 +---+---+---+---+---+---+
 0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

Indexing

```
1) word[0:2]
2) word[2:5]

3) word[:2]
4) word[4:]
5) word[-2:]

6) word[:2] + word[2:]

7) word[::2]
8) word[::-1]

9) word[0] = 'J'
10)'J' + word[1:]

11)len(word)
```

An omitted first index defaults to zero, an omitted second index defaults to the size of the string being sliced.

String is immutable.

List (1)

```
1) squares = [1, 4, 9, 16, 25]
2) squares

3) squares[0]
4) squares[-1]
5) squares[-3:]
6) squares[::-1]

7) squares + [36, 49, 64, 81, 100]

8) cubes = [1, 8, 27, 65, 125]

9) cubes[3] = 64

10)del cubes[4]
```

the cube of 4 is 64, not 65!

List is mutable.

List (2)

```
1) letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']

2) letters[2:5] = ['C', 'D', 'E']
3) letters[2:5] = []
4) letters[:] = []

5) len(letters)

6) a = ['a', 'b', 'c']
7) n = [1, 2, 3]
8) x = [a, n]

9) y = x[:]
10)y[0][0] = 'z'
11)x
```

All slice operations return a new list containing the requested elements.

Tuple

```
1) numbers = (1, 2, 3)
2) numbers = 1, 2, 3

3) numbers = 1,

4) numbers[0]
5) numbers[0] = 2 # Fail

6) number_list = list(numbers)
```

Imutable

Iterable

Dictionary

```
1) numbers = {1:'One', 2:'Two'}
2) numbers = {'One':1, 'Two':2}

3) info = ( ('One', 1), ('Two', 2) )
4) dict( info )

5) dict( One= 1, Two= 2 )

6) numbers['One']

7) numbers['Three'] = 3

8) del numbers['Three']
```

Mutable

Iterable

Indexing

Control Flow (if)

```
1) num = input("Please enter an integer: ")
2) x = int(num)

3) if x <  0:
4)     print('Negative')
5) elif x == 0:
6)     print('Zero')
7) elif x == 1:
8)     print('Single')
9) else:
10)    print('More')

11)x = 1
12)if x: print("True")

13)x = 0
14)if not x : print("False")
```

True, number not 0, iterable not 0 length
ex.) -1, 1, "a", [1]

False, Zero, zero length, None
ex.) 0, "", []

Control Flow (for)

```
1) words = ['cat', 'window', 'defenestrate']
2) for w in words:
3)     print(w, len(w))

4) range(10)

5) list(range(5, 10))
6) list(range(0, 10, 3))
7) list(range(-10, -100, -30))

8) sum(range(4))

9) for i in range(len(words)):
10)    print(i, a[i])
```

Python's for statement iterates over the items of any sequence (a list or a string), in the order that they appear in the sequence.

Iterable object. It doesn't really make the list, thus saving space.

Control Flow (for) (2)

```
1) numbers = [1,2,2,3]
2) for i, n in enumerate(numbers):
3)     if n % 2 == 0: del numbers[i]
```

Code that modifies a collection while iterating over that same collection can be tricky to get right.

```
4) corr = 0
5) for i in range(len(numbers)):
6)   if numbers[i - corr] % 2 == 0:
7)       del numbers[i - corr]
8)       corr += 1
```

Strategy: Iterate over a copy

```
9) odds = []
10)for n in numbers:
11)    if n % 2 == 1: odds = odds + [n]
```

Strategy: Create a new collection

Control Flow (for) (3)

```
1) for n in range(2, 10):
2)     for x in range(2, n):
3)         if n % x == 0:
4)             print(n, 'equals', x, '*', n//x)
5)             break
6)     else:
7)         print(n, 'is a prime number')

8) for num in range(2, 10):
9)     if num % 2 == 0:
10)        print("Found an even number", num)
11)        continue
12)    print("Found an odd number", num)
```

This is exemplified by the following loop, which searches for prime numbers

exhaustion of the iterable (with for) or when the condition becomes false (with while)

continues with the next iteration of the loop

Function (1)

```
1) def add(x,y):
2)     print(x + y)

3) add(1,2)

4) a = add
5) a(1,2)

6) def add(x,y):
7)     print(x + y)
8)     return x + y

9) result = add(1,2)

10)add = lambda x, y: x + y
11)add(1,2)
```

The function doesn't return a value called None.

Small anonymous functions can be created with the lambda keyword.

Function (2)

```
1) def add(x, y, z=1):
2)     return x + y + z

3) add(1,1)
4) add(1,1,0)
5) add(1,1,z=0)
```

x, y are positional argument; z is positional or keyword argument, optional

```
6) z_init = 1
7) def add(x, y, z=z_init):
8)     return x + y + z

9) z_init = 2
10)add(1,1)
```

The default values are evaluated at the point of function definition in the defining scope

Function (3)


```
1) def add(x, *args, y=0, **kwargs):
2)     return x, args, y, kwargs


3) add(1,2,3,z=5,a=6)
4) add(1,2,3,4,z=5,a=6)


5) args = (2,3)
6) kwargs = {'z':5, 'a':6}
7) add(1, *args, 4, **kwargs)


8) def add(x: int, y: int) -> int:
9)     return x + y
```

args and kwargs contain all positional and keyword arguments.
args is tuple; kwargs dict

Unpacking Argument Lists
4) and 7) show same results.

Function annotations are completely optional metadata information about the types used by user-defined functions.

Class


```
1) class Calc:
2)     def __init__(self, n1, n2):
3)         self.n1 = n1
4)         self.n2 = n2
5)     def add(self):
6)         return self.n1 + self.n2


7) calc = Calc(1, 2)
8) calc.n1, calc.n2


9) calc.add()
```

Class is a kind of data type which has functions and variables in its namespace.

More on Lists (1)

```
1) numbers = [1, 3, 2, 5, 4, 3]
2) len(numbers)

3) numbers.count(3)
4) numbers.index(3)
5) numbers.index(3,2)

6) numbers.append(0)
7) numbers.pop()
8) numbers.extend([10, 11])

9) numbers.insert(0, -1)
10)numbers.remove(-1)

11)numbers.reverse()
12)numbers.sort()
```

```
numbers[len(numbers):] = [0]
```

```
numbers[len(numbers):] = [10, 11]
```

More on Lists (2)

```
1) squares = []
2) for x in range(10):
3)     squares.append(x**2)

4) squares = [ x**2 for x in range(10) ]
5) squares = ( x**2 for x in range(10) )
6) list(squares)

7) [(x, y) for x in [1,2,3] for y in [1,2,3] if x != y]

8) combs = []
9) for x in [1,2,3]:
10)     for y in [1,2,3]:
11)         if x != y:
12)             combs.append((x, y))
```

Generator allows you to declare a function that behaves like an iterator.

Looping Technique

```
1) records = [ [ 0, 'Park'], [ 1, 'Kim'] ]
2) records.sort(reverse=True)

3) id = []; name = []
4) for _id, _name in records:
5)     id.append(_id)
6)     name.append(_name)

7) list(zip(*records))
8) id, name = zip(*records)

9) info = dict(records)
10)for k in info:
11)    print(k, info[k])

12)for k, v in info.items():
13)    print(k, v)
```

The comparison
uses lexicographical ordering: first
the first two items are compared,
and if they differ this determines
the outcome of the comparison;
(1, 2, 3) < (1, 2, 4)
(1, 2, 3, 4) < (1, 2, 4)

Modules and Packages (1)

```
1) import os
2) dir(os)

3) os.name
4) os.listdir()

5) os.mkdir('kasi')
6) os.listdir()

7) import os.path
8) os.path.exists('kasi')

9) import os.path as os_path
10)os_path.exists('kasi')

11)from os.path import exists, isdir as path_isdir
12)path_isdir('kasi')
```

A module is a file containing Python definitions and statements. The file name is the module name with the suffix .py appended.

Packages are a way of structuring Python's module namespace by using "dotted module names".

Modules and Packages (2)

package/__init__.py

1) from package.sub_package import module
2) from .sub_package import module

package/sub_package/__init__.py

1) from . import module
2) from ..sub_package2 \
3) import module as module2

4) from package import module
5) from package.sub_package \
6) import module, module2

```
package/
  __init__.py
  sub_package/
    __init__.py
    module.py
  sub_package2/
    __init__.py
    module.py
```

The __init__.py files are required to make Python treat directories containing the file as packages.

Relative imports use leading dots to indicate the current and parent packages involved in the relative import.

Text Handling (1)

```
[[fill]align][sign][#][0][width][grouping_option][.precision][type]
https://docs.python.org/3/library/string.html#formatstrings
```

```
1) bin = b'Park'
2) txt = bin.decode()
3) txt.encode()
```

bytes stores raw data as 8 bits sequence.
str stores encoded raw data to represent text in system encoding scheme.

```
4) score = 88.334
5) name = 'Park'

6) f'{name}\'s score is {score}'
7) f'{name:10}\'s score is {score:+08.4}'

8) for x in range(1, 11):
9)     r = x, x*x ,x*x*x
10)    print(f'{r[0]:2d} {r[1]:3d} {r[2]:4d}')
```

Text Handling (2)

```
1) '{}, {}'.format('Park', 'Jongyeob')

2) fullname = ('Park', 'Jongyeob')
3) '{}, {}'.format(*fullname)
4) '{1} {0}'.format(*fullname)
5) '{1[0]}. {0}'.format(*fullname)

6) '{last} {first}'.format(first='Jongyeob', last='Park')

7) fullname = {'first':'Jongyeob', 'last':'Park'}
8) '{last} {first}'.format(**fullname)

9) '{0[first]} {0[last]}'.format(fullname)
10)'{0[first][0]} {0[last]}'.format(fullname)
11)'{0[first][0]:5} {0[last]}'.format(fullname)
```

Text Handling (3)

```
1) text = 'Spam-Egg-Sandwitch'
2) text.upper()
3) text.lower()

4) text.startswith('Spam')
5) text.endswith('Sandwitch')
6) text.replace('Egg', 'Onion')

7) i = text.find('a')
8) i2 = text.find('a', i+1)
9) text[i+1:i2]

10)text.split('-')

11)"42".zfill(5)
12)"-42".zfill(5)
```

Imutable

https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str

Dates and Times (1)

```
1) import datetime
2) d = datetime.date(2022,2,11)
3) t = datetime.time(12,0,0)

4) dt = datetime.datetime.combine(d,t)
5) print("Datetime is {:%y%m%d_%H%M%S}".format(dt) )

6) dt_str = "220211_120000"
7) fmt = "%y%m%d_%H%M%S"

8) dt = datetime.datetime.strptime(
9)     dt_str,
10)    fmt)

11)dt.strftime(fmt)
```

strftime() and strptime() Format Codes:
https://docs.python.org/3/library/datetime.html

Dates and Times (2)

```
1) td = datetime.timedelta(days=1)
2) td.total_seconds()

3) td2 = datetime.timedelta(days=1, seconds=10)
4) td2 - td

5) t1 = datetime.datetime(2020,2,28,0,0,0)
6) td = datetime.timedelta(days=1)
7) t1 + td

8) t2 = datetime.datetime(2020,2,29,0,0,0)
9) t2 - t1

10)datetime.datetime.now()
11)datetime.datetime.utcnow()
```

File Handling (1)

```
1) text = \
2) '''Apple
3) Orange
4) Watermelon
5) '''

6) f = open('workfile.txt', 'w')
7) f.write(text)
8) f.close()

9) f = open('workfile.txt', 'r')
10)f.read()
11)f.close()

12)import os
13)os.listdir()
```

```
'r' : File read
'w' : File write
'r+': File read and write
'a' : File append
```

File Handling (2)

```
1) add_text = \
2) '''Carrot
3) '''


4) with open('workfile.txt', 'a') as f:
5)     f.write(add_text)


6) f = open('workfile.txt', 'r')
7) f.readline()
8) for line in f:
9)     print(line)
```

The advantage using with keyword is that the file is properly closed after its suite finishes.

f.readline() reads a single line from the file; a newline character (\n) is left at the end of the string, and is only omitted on the last line of the file if the file doesn’t end in a newline.

File System Handling (1)

```
1) import os

2) cwd = os.getcwd()
3) os.listdir(cwd)

4) filepath = os.path.join(cwd, 'workfile.txt')
5) os.path.exists(filepath)
6) os.path.isfile(filepath)
7) os.path.isdir(filepath)

8) relpath = os.path.relpath(filepath)
9) abspath = os.path.abspath(relpath)

10)os.path.split(abspath)
11)os.path.splitext(abspath)
```

```
UNC: c:\windows\path
POSIX: /linux/path
```

File System Handling (2)

```
1) for i in range(10):
2)     d = f'sub/sub{i}'
3)     os.makedirs(d)
4)     for j in range(10):
5)         p = f'{d}/test{j}.txt'
6)         with open(p,'w') as f:
7)             pass

8) import glob
9) files = glob.glob('sub/**/*.txt')
10)len(files)

11)glob.glob('sub/sub[0-1]/test[0-1].txt')
```

```
sub/sub0/test0.txt
         ...
         test9.txt
    ...
    sub9/test9.txt
```

The glob module provides a function for making file lists from directory wildcard searches

Internet Access

```
1)  from urllib.parse import urlparse

2)  url = 'http://worldtimeapi.org/api/timezone/etc/UTC.txt'
3)  o = urlparse(url)

4)  o.hostname
5)  o.path

6)  import os
7)  dirpath, filepath = os.path.split(o.path)

8)  from urllib.request import urlopen

9)  with urlopen(url) as response:
10)     with open(filepath,'w') as f:
11)         text = response.read().decode()
12)         f.write(text)

13) os.listdir()
```

```
abbreviation: UTC
client_ip: 175.121.103.248
datetime: 2022-02-11T05:34:08.457849+00:00
day_of_week: 5
day_of_year: 42
dst: false
dst_from:
dst_offset: 0
dst_until:
raw_offset: 0
timezone: Etc/UTC
unixtime: 1644557648
utc_datetime: 2022-02-11T05:34:08.457849+00:00
utc_offset: +00:00
week_number: 6
```

Convert bytes to a str

Parallelism (1)

```
1) import time

2) num = 0
3) def pop(id):
4)     global num
5)     time.sleep(1)
6)     for i in range(10):
7)         print(f"{id} [{i}] Num= {num}")
8)         num += 1
9)         time.sleep(0.1)

10) import threading

11) threads = [ threading.Thread(target=pop, args=(i,)) for i in range(2) ]
12) for th in threads: th.start()

13) for th in threads: th.join()
```

```
1 [8] Num= 16
0 [8] Num= 16
1 [9] Num= 18
0 [9] Num= 18
```

Parallelism (2)

```
1)  lock = threading.Lock()

2)  num = 0
3)  def pop(id):
4)      global num
5)      time.sleep(1)
6)      for i in range(10):
7)          lock.acquire()
8)          print(f"{id} [{i}] Num= {num}")
9)          num += 1
10)         lock.release()
11)         time.sleep(0.1)

12) threads = [ threading.Thread(target=pop, args=(i,)) for i in range(2) ]
13) for th in threads: th.start()

14) for th in threads: th.join()
```

Critical section

```
0 [8] Num= 16
1 [8] Num= 17
0 [9] Num= 18
1 [9] Num= 19
```