



데이터 분석을 위한 PYTHON

PART

FUNCTIONS

PYTHON FUNCTIONS

◆ 함수(Function)란

반복되는 기능 코드를 하나로 묶어 이름을 붙인 것을 **함수**라 함
해당 기능 필요 시 **함수 이름을 불러서 사용** 가능

함수 호출

함수 호출이 되지 않으면
함수 코드는 실행(동작)하지 않음

◆ 함수(Function) 선언

■ 선언 형식

```
def 함수명(매개변수1, 매개변수2,..., 매개변수N):  
    실행문  
    실행문  
    return 반환값
```

함 수 명 : 코드의 기능을 알 수 있도록 작성

매개 변수 : 기능 구현에 필요한 재료로 파라미터라 함

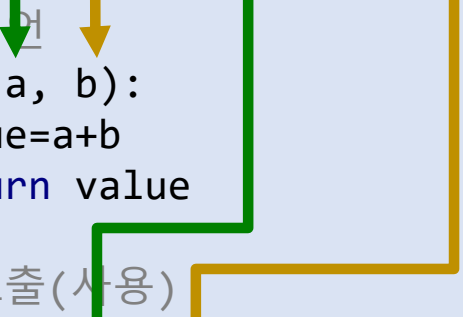
반 환 값 : 기능 구현 후 결과물, 존재하지 않을 수도 있음

◆ 함수(Function) 호출

■ 선언된 함수 사용 방법

→ 형식 : 함수명 (데이터) : 매개변수 있는 함수 호출

함수명 () : 매개변수 없는 함수 호출



```
# 함수 선언
def add(a, b):
    value=a+b
    return value

# 함수 호출(사용)
result=add(10, 20)
print('add(10, 20) =', result)
```

The diagram illustrates the execution flow of a Python function. A green arrow originates from the `def` keyword in the function definition and points to the `add` function name in the function call `add(10, 20)`. A yellow arrow originates from the `return` statement in the function definition and points to the `result` variable in the function call, indicating the return value being passed back.

◆ 함수(Function) 반환

■ 결과 반환 return

- 호출한 곳으로 돌아감
- 어떤 종류의 객체도 반환 가능
- return 사용하지 않거나 retur만 적어도 함수 종료
- 함수 중간에 빠져나가기
- 여러 개 값 반환 가능

◆ 함수(Function) 다양한 형태

▪ 1) 인자와 반환값 존재하는 경우

```
def add( a, b ):
    result = a + b
    return result

value = add( 5, 9 )
```


◆ 함수(Function) 다양한 형태

▪ 2) 인자만 존재하는 경우

```
def print_address( name ):
    print("서울 특별시 종로구 1번지")
    print("파이썬 빌딩 7층")
    print( name )

print_address("홍길동")
```

◆ 함수(Function) 다양한 형태

▪ 3) 인자, 반환값 없는 경우

```
def print_address( ):
    print("서울 특별시 종로구 1번지")
    print("파이썬 빌딩 7층")
    print("홍길동")
```

```
print_address()    # 함수 호출
```

◆ 함수(Function) 다양한 형태

▪ 4) 반환값만 존재 하는 경우

```
def calculate_area ( ):
    area = 3.14 * 10 **2
    return area

print( calculate_area ( ) )
```

◆ 함수(Function) 다양한 형태

▪ 5) 인자 개수 미정인 경우

```
def 함수명( *매개변수 ):
    실행문
    실행문
    return 반환값
```

가변인자

◆ 함수(Function) 다양한 형태

▪ 5) 인자 개수 미정인 경우

```
def getSum ( *args ):  
    total = 0  
    for i in args:  
        total += i  
    return total  
  
print("result %d" %getSum())  
  
result = getSum(1,2,3,4,5)  
print("result %d" %result)  
  
result = getSum(11,22,33,44,55,66,77,88,99)  
print("result %d" %result)
```

◆ 함수(Function) 다양한 형태

▪ 6) 키워드 파라미터 형태

def 함수명(**매개변수):

실행문

실행문

return 반환값

key=value 형태
매개변수

딕셔너리
형태 반환

◆ 함수(Function) 다양한 형태

▪ 6) 키워드 파라미터 형태

```
def print_kwargs( **kwargs ):
    print( kwargs )

# 함수 호출
print_kwargs(name='kim', age=30, gender='F')
```

◆ 함수(Function) 다양한 형태

▪ 7) 매개변수 초기값 설정

```
def 함수명( 매개변수1, 매개변수2, 매개변수3=초기값):
```

```
    실행문
```

```
    실행문
```

```
    return 반환값
```

맨마직 위치!!

◆ 함수(Function) 다양한 형태

▪ 7) 매개변수 초기값 설정

```
def set_info(name, old, man=True):  
    print(f"Name is {name}")  
    print(f"Old is {old}")  
    if man:  
        print(f"Man")  
    else:  
        print(f"Woman")  
  
set_info("Kim", 10)  
set_info("Tom Lee", 15, False)
```

◆ 함수(Function) 다양한 형태

▪ 8) Lambda(람다) 표현식/함수

- 이름 없는 한 줄 함수 즉, **익명함수**
- **문법적 필요성에 의해** 일회성 함수
- 형식 : **lambda** 인수1, 인수2, ... : 실행코드 (**리턴 값**)

◆ 함수(Function) 다양한 형태

▪ 8) Lambda(람다) 표현식/함수

```
add=lambda x, y : x+y
```

```
print( add(1,2) )
```

```
cal=[ lambda x,y:x+y, lambda x,y: x-y, lambda x,y:x/y ]
```

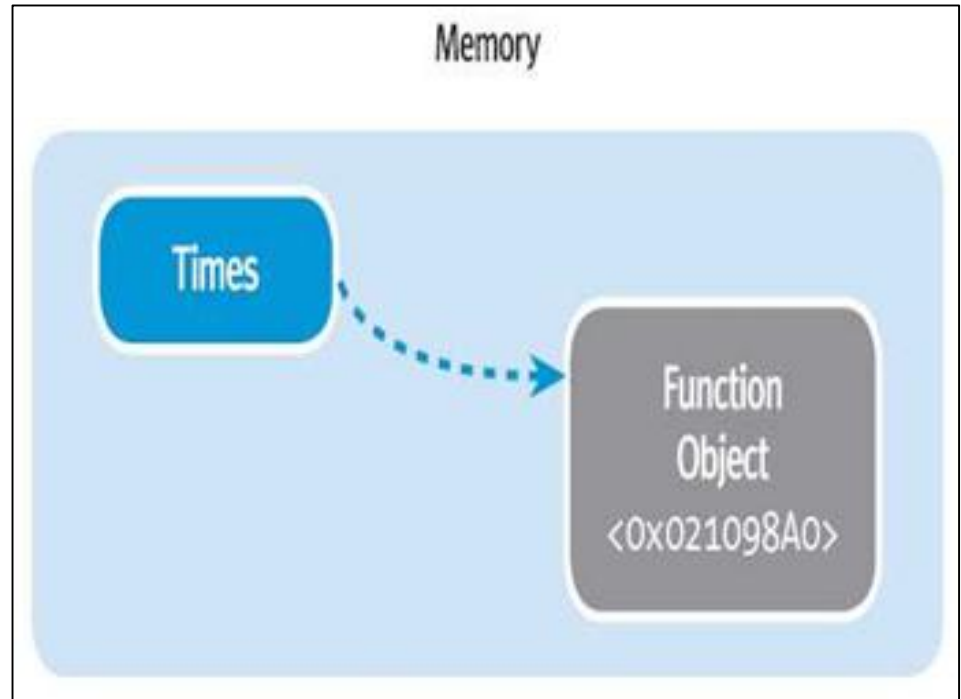
```
print( cal[0](1,2) )
```

```
print( cal[1](1,2) )
```

```
print( cal[2](1,2) )
```

◆ 변수에 함수 담기

- 메모리에 객체 생성
- 레퍼런스 생성



◆ 변수에 함수 담기

➤ 함수도 class 'function' 타입으로 변수에 대입 가능

```
def something(a):  
    print(a)  
  
p=something  
print("Wntype(p)= ", type(p))  
  
p(123)  
p("Good Happy")
```

```
type(p)= <class 'function'>  
123  
Good Happy
```

◆ 변수에 함수 담기

```
def plus(a,b):  
    return a*b
```

```
def minus(a,b):  
    return a-b
```

```
first=[plus, minus]  
print("Wntype(first)=", type(first))  
  
print("first[0](1,2)=", first[0](1,2))  
print("first[1](1,2)=", first[1](1,2))
```

◆ 함수와 변수 관계

➤ 변수 위치에 따른 분류

지역변수 (Local Variable)

- 함수 안에 존재하는 함수
- 함수에서만 사용 가능
- 함수 실행 후 메모리에서 사라짐

전역변수 (Global Variable)

- 함수 밖에 존재하는 함수
- 프로그램 어디서나 사용 가능
- 프로그램 종료 시 메모리에서 사라짐

◆ 함수와 변수 관계

test.py

```
a=10                                # 전역변수

def test(a):                        # 지역변수
    a += 1
    print('a =>', a)

# 함수 호출(사용)
test(20)
print("a is %d" %a)
```


◆ 함수와 변수 관계

test.py

```
a=10                                # 전역변수

def test( ):
    # global a                      #전역변수 사용알림
    a += 1                          # 전역변수
    a = a + 1
# 함수 호출(사용)
test(a)
print("a is %d" %a)
```

◆ 함수 안에 함수

```
# 함수 선언 -----  
def print_hello():  
    hello = 'Hello, world!'  
    def print_message():  
        print(hello)  
    print_message()  
  
# 함수 호출 -----  
print_hello()
```

다양한 BUILTIN FUNC

◆ Func

chr(code_value)

아스키(ASCII) 코드 값(0~127)을 입력받아 해당하는 문자 출력

```
print(f'chr(97)={chr(97)}, chr(65)={chr(65)}')
```

ord(문자)

문자의 아스키(ASCII) 코드 값(0~127) 반환 출력

```
print(f'ord("a")={ord("a")}, ord("Z")={ord("Z")}')
```

divmod(a, b)

a를 b로 나눈 몫과 나머지를 튜플 형태로 돌려주는 함수

```
print(f'divmod(11,2) => {divmod(11,2)}')
```

◆ Func

```
# zip( 반복 가능한(iterable) 자료형 )  
# 동일한 개수로 이루어진 자료형을 묶어 주는 역할  
datas=zip([1, 2, 3], [4, 5, 6])  
print(f'datas ={datas}')  
for x, y in datas:  
    print(x,y)  
  
datas=list(zip("abc", "def"))  
print(f'datas ={datas}')  
for x, y in datas:  
    print(x,y)
```

PART

파일 처리와 모듈 활용

파일 입출력

파일 입출력

◆ 바이너리 & 텍스트

[데이터 분류]

데이터 타입	장점	단점
텍스트	<ul style="list-style-type: none">- 텍스트 편집기로 편집 가능- 데이터 설명 추가 가능	<ul style="list-style-type: none">- 바이너리에 비해 크기가 큼- 문자 인코딩 주의 (대부분 UTF-8)
바이너리	<ul style="list-style-type: none">- 텍스트 데이터에 비해 크기 작음- WEB에서 사용되는 데이터	<ul style="list-style-type: none">- 텍스트 편집기로 편집 불가- 데이터 설명 추가 불가

[텍스트 데이터 파일]

파일	특징
XML 파일	<ul style="list-style-type: none">- 범용적인 형식, 웹 API 활용 형식
JSON 파일	<ul style="list-style-type: none">- 자바스크립트 객체 표기 방법 기반 형식 파일- 데이터 교환에 활용
YAML	<ul style="list-style-type: none">- JSON 대응으로 사용, 어플리케이션 설정 파일에 많이 사용되는 파일
CSV/TSV	<ul style="list-style-type: none">- WEB상에서 많이 사용되는 파일

파일 입출력

◆ 인코딩 & 디코딩

[사람 중심]

UNICODE

가을입니다.

인코딩(Encoding)
코드화/암호화

디코딩(Decoding)
역코드화/복호화

[기계 중심]

UTF-8, ASCII 등등 형식의 BYTE

₩xea₩xb0₩x80₩xec₩x9d₩x84
₩xec₩x9e₩x85₩xeb₩x8b₩x88
₩xeb₩x8b₩xa4

0b100000000b11101100
0b100111010b10000100
0b111011000b10011110
0b100001010b11101011
0b100010110b10001000
0b111010110b10001011
0b101001000b10111000

파일 입출력

◆ 파일 읽기& 쓰기

쓰기 : file_Object = **open**(file , mode='w', encoding=None)

읽기 : file_Object = **open**(file , mode='r', encoding=None)

racter	Meaning
'r'	open for reading (default)
'w'	open for writing, truncating the file first
'x'	open for exclusive creation, failing if the file already exists
'a'	open for writing, appending to the end of the file if it exists
'b'	binary mode
't'	text mode (default)
'+'	open a disk file for updating (reading and writing)

파일 입출력

◆ 파일 데이터 쓰기

파일 전체 데이터 쓰기 : alldata=f.**write**(str)

파일 줄 단위 데이터 쓰기 : line = f.**writeline**(list)

**** 자동 개행 안됨**

파일 입출력

◆ 파일 데이터 읽기

파일 전체 데이터 : alldata=f.read()

파일 n만큼 데이터 : alldata=f.read(n)

파일 줄 단위 데이터 : line = f.readline()

파일 줄 단위 전체 리스트 반환 : lines = f.readlines()

파일 입출력

◆ 파일 포인터 위치

파일 위치 설정/이동 → `f.seek(offset)` # `f.seek(0)` 파일 처음

파일 위치 읽기 → `f.tell()`

파일 닫힘 여부 반환 → `f.closed`

파일모드 반환 → `f.mode`

파일이름 반환 → `f.name`

파일 입출력

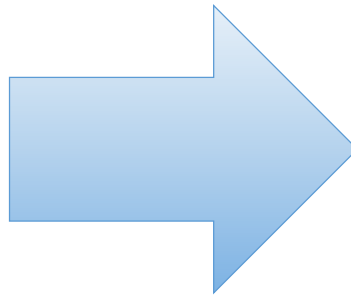
◆ 파일 데이터 읽기

```
alldata=f.read( )
```

```
alldata=f.read( n )
```

```
line = f.readline( )
```

```
lines = f.readlines( )
```



1바이트 크기 배열
bytes 객체

파일 입출력

◆ with 파일 as 구문

```
with open(file , mode='w', encoding=None) as file_obj:  
    file_obj.write( ' data ' )
```

file_obj.close() 생략 가능

파일의 close() 자동으로 처리됨!!!

예외처리

◆ 예외처리란

- 프로그램 오류 발생 시 중단을 막기 위해 처리해주는 방법

```
num1=10
```

```
num2=0
```

```
print(f'{num1}/{num2} = {num1/num2}')
```

```
ex_exception_01 ×
```

```
C:\Users\anece\anaconda3\envs\EV_PY37\python
```

```
Traceback (most recent call last):
```

```
File "D:/BEGINNER_AI_2ND/EXAM_PY/EXAM_EXCE
```

```
    print(f'{num1}/{num2} = {num1/num2}')
```

```
ZeroDivisionError: division by zero
```

◆ 예외처리

■ 예시

```
file = open("../Data/address.txt", 'r')  
print(file.read())  
file.close()
```

Traceback (most recent call last):

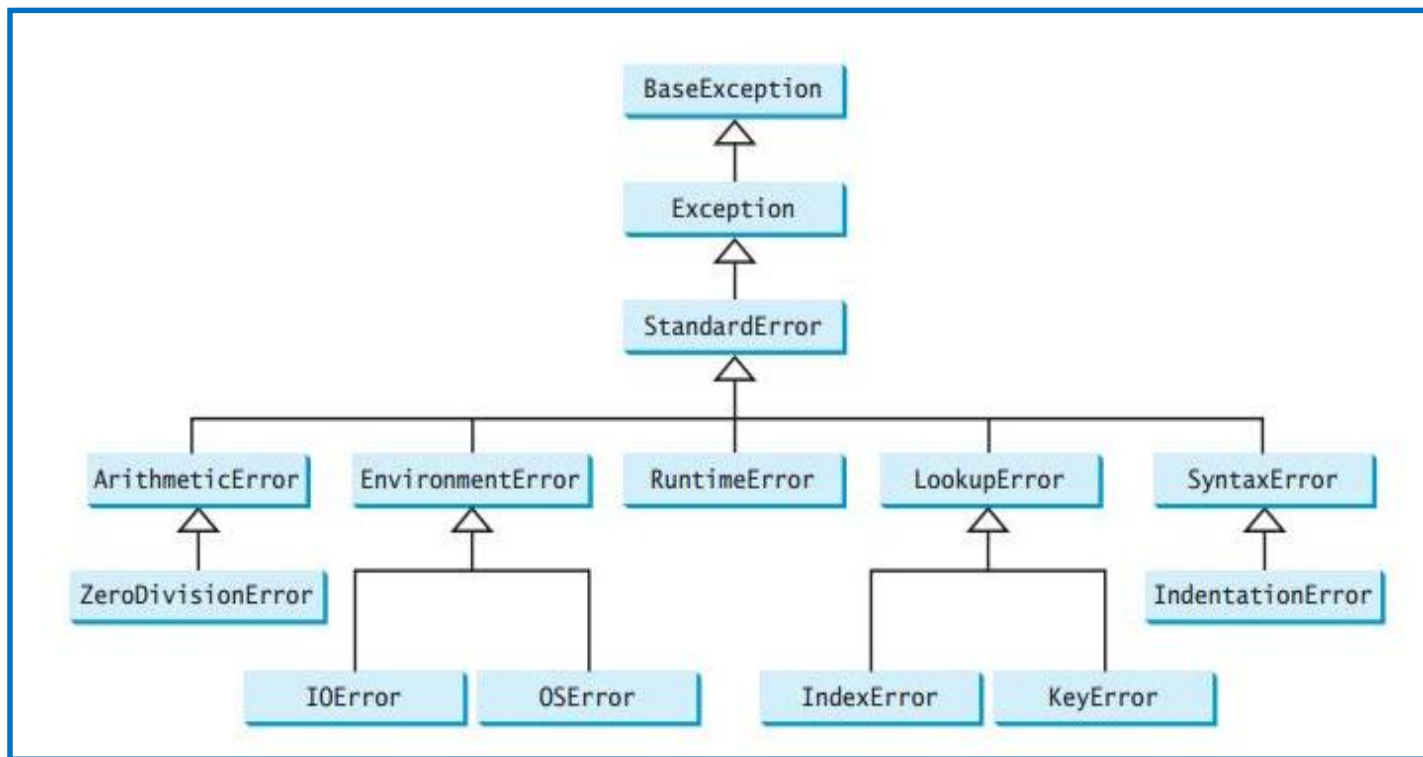
File "C:/Users/RNU/PycharmProjects/PY_BASIC/FILE_IO/ex_read.py", line 48, in <module>

file = open("../Data/address.txt", 'r')

FileNotFoundError: [Errno 2] No such file or directory: '../Data/address.txt'

◆ 예외처리 분류

- BaseException 클래스 하위 클래스로 존재



◆ 예외처리 구문

■ try ~ except 문

try:

예외발생가능 코드

except 오류:

예외발생 처리코드

try:

예외발생가능 코드

except 오류:

예외발생 처리코드

else:

예외 발생하지 않은 경우

try:

예외발생가능 코드

except 오류:

예외발생 처리코드

else:

예외 발생하지 않은 경우

finally:

무조건실행코드

◆ 예외처리 예

```
try:
    x = int( input('나눌 숫자를 입력하세요: ') )
    y = 10 / x
    except ZeroDivisionError: # 0으로 나눠서 에러 발생 때 실행
        print('숫자를 0으로 나눌 수 없습니다.')
    else:
        print( y )
    finally:
        print(" - 끝 -- ")
```

◆ 예외처리 예

```
try:
    x = int( input('나눌 숫자를 입력하세요: ') )
    y = 10 / x
except ZeroDivisionError as ze :      # 0으로 나눠서 에러 발생 때 실행
    print('숫자를 0으로 나눌 수 없습니다.', ze )
else:
    print( y )
finally:
    print(" - 끝 -- ")
```

◆ 예외처리 예

```
y = [10, 20, 30]
```

```
try:
```

```
    index, x = map(int, input('인덱스와 나눌 숫자를 입력하세요: ').split())
```

```
    print(y[index] / x)
```

```
except ZeroDivisionError as ze:
```

```
    print('숫자를 0으로 나눌 수 없습니다.', ze)
```

```
except IndexError as ie:
```

```
    print('잘못된 인덱스입니다.', ie)
```

```
finally:
```

```
    print(" 무조건 끝")
```

여러 개 처리

◆ 예외 회피/무시 구문

▪ try ~ except 문

try:

예외발생가능 코드

except 오류:

pass

try:

예외발생가능 코드

except 오류:

pass

else:

예외가없을경우처리

try:

예외발생가능 코드

except 오류:

pass

else:

예외없을경우처리코드

finally:

무조건실행코드

◆ 강제 오류 발생

- 프로그램 실행 중 강제성 가진 작업 실행을 위한 방법
- 미 실행 시 강제 오류 발생

raise 에러이름

◆ 강제 오류 발생

```
try:
```

```
x = int(input('3의 배수를 입력하세요: '))
```

```
if x % 3 != 0:
```

```
# 3의 배수 아니면
```

```
    raise Exception('3의 배수가 아닙니다.') # 예외 발생시킴
```

```
print(x)
```

```
except Exception as e:
```

```
# 예외 발생 시 실행
```

```
    print('예외가 발생했습니다.', e)
```

◆ 강제 오류 발생

```
def three_number():  
    x = int(input('3의 배수를 입력하세요: '))  
    if x % 3 != 0:                                # 3의 배수 아닌경우  
        raise Exception('3의 배수가 아닙니다.') # 예외 발생  
    print(x)  
  
try:  
    three_number()  
except Exception as e:  
    print(" 예외 발생 : ", e)
```

호출한 곳으로 예외 넘김

- 함수 내에서 예외발생
- 처리 : 함수 호출한 쪽

모듈 & 패키지

◆ 모듈(Module)

- 기능 수행 위해 변수, 함수, 클래스를 모아둔 파일
- python에 다양한 기능을 확장시켜 줄 수 있는 것
- 다른 Python 프로그램에서 불러서 사용가능하게 만든 것

분 류	설명
표준 모듈	파이썬에서 기본 제공 모듈
사용자 생성 모듈	개발자가 만든 모듈
써드 파티션 모듈	다른 사람들이 만들어서 공개하는 모듈

◆ 모듈(Module) 사용

■ 전체 사용

<code>import</code> 모듈명	# 모듈 내의 변수, 함수 호출 사용
<code>import</code> 모듈명 <code>as</code> 별칭	# 모듈 내의 변수, 함수 호출 사용

```
import math
```

```
print(" 원주율 = ", math.pi)
print(" 2의 3승 = ", math.pow(2,3) )
print(" 3! = ", math.factorial(3) )
```

```
import math as m
```

```
print(" 원주율 = ", m.pi)
print(" 2의 3승 = ", m.pow(2,3) )
print(" 3! = ", m.factorial(3) )
```

◆ 모듈(Module) 사용

■ 모듈 일부만 사용

from 모듈명 **import** 함수명1, 함수명2 <= 특정 함수만 사용

```
from math import pow
print("제곱 = ", pow(2,3))
```

```
from math import pow as p
print(" 원주율 = ", p(2,3))
```

```
from math import pow, pi
print(" 제곱 = ", pow(2,3))
print(" 원주율 = ", pi )
```

```
from math import *
print(" 제곱 = ", pow(2,3))
print(" 원주율 = ", pi )
```

◆ 모듈(Module) & 스크립트(Script)

▪ 2가지 동작 모드 제어

`__name__` <= 현재 실행 모듈 이름 저장하고 있는 내장 변수

- 현재 실행 중일 경우 변수 저장 값 : `__main__`
- 다른 파일에 포함된 경우 변수 저장 값 : **파일명**

`__name__`.py <= 패키지의 경우 동일 효과

◆ 모듈(Module) & 스크립트(Script)

▪ 2가지 동작 모드 제어

```
print( __name__)
```

```
if __name__ == "__main__":
```

```
    print("나는 현재 실행 중입니다.")
```

```
else:
```

```
    print("나의 이름은 {0}입니다.".format(__name__))
```

◆ 모듈(Module) & 스크립트(Script)

▪ 2가지 동작 모드 제어

```
def get_sum(a, b):  
    return a+b  
  
def main():  
    data_list=[[1,1], [2,2], [3,3], [4,4]]  
    sum =0  
    for i in range(0,len(data_list)):  
        sum += get_sum(data_list[i][0], data_list[i][1])  
  
    print("sum = %d" %sum)
```

◆ 모듈(Module) & 스크립트(Script)

▪ 2가지 동작 모드 제어

```
if __name__ == "__main__":  
    print("나는 현재 실행 중입니다.")  
    print("aaa.py 시작합니다.")  
    main()  
else:  
    print("나는 {0}입니다.".format(__name__))
```

◆ 패키지(Package)

- 특정 기능과 관련된 여러 모듈을 묶은 것
- python에 다양한 기능을 확장시켜 줄 수 있는 것
- 다른 Python 프로그램에서 불러서 사용가능하게 만든 것
- 설치 작업이 추가로 필요한 경우도 있음

- 파이썬 패키지 인덱스(Python Package Index, PyPI)
- www.pypi.org

◆ 패키지(Package)

- 파이썬 패키지 인덱스(Python Package Index, PyPI)



◆ 패키지(Package)

■ 전체 사용

```
import 패키지명.모듈명  
import 패키지명.모듈명1, 모듈명2  
import 패키지명.모듈명 as 별칭
```

```
import urllib.request  
  
import urllib.request as r
```

◆ 패키지(Package)

■ 전체 사용

```
import urllib                # urllib 전체 패키지

req = urllib.request.Request('http://www.google.co.kr')
response = urllib.request.urlopen(req)
```

```
import urllib.request as request    # urllib 패키지 request 모듈

req = request.Request('http://www.google.co.kr')
response = request.urlopen(req)
```

◆ 패키지(Package)

■ 일부만 사용

```
from 패키지명.모듈명 import 변수명
```

```
from 패키지명.모듈명 import 함수명
```

```
from 패키지명.모듈명 import 클래스명
```

```
# 클래스, 함수만 사용
```

```
from urllib.request import Request, urlopen
```


◆ 패키지(Package)

■ 일부만 사용

```
# urlopen 함수, Request 클래스 가져옴
from urllib.request import Request, urlopen

# Request 클래스를 사용하여 req 생성
req = Request('http://www.google.co.kr')
response = urlopen(req)
```

◆ 패키지(Package)

■ 일부만 사용

```
# urllib의 request 모듈의 모든 변수, 함수, 클래스  
from urllib.request import *  
  
req = Request('http://www.google.co.kr')  
response = urlopen(req)
```

◆ 패키지(Package)

- 설치 명령어 : `pip install` 패키지명
- 버전 2가지 존재 : `pip --version`

Terminal: Local x +

(c) 2020 Microsoft Corporation. All rights reserved.

(EV_PY37) D:\BEGINNER_AI_2ND\EXAM_PY>pip --version

pip 20.2.2 from C:\Users\anece\anaconda3\envs\EV_PY37\lib\site-packages\pip (python 3.7)

(EV_PY37) D:\BEGINNER_AI_2ND\EXAM_PY>

◆ 패키지(Package)

- 명령어 사용법 확인: `pip --help`

```
Terminal: Local × +  
  
(EV_PY37) D:\BEGINNER_AI_2ND\EXAM_PY>pip --help  
  
Usage:  
  pip <command> [options]  
  
Commands:  
  install          Install packages.  
  download         Download packages.  
  uninstall        Uninstall packages.  
  freeze           Output installed packages in requirements  
  list             List installed packages.  
  show            Show information about installed packages.  
  check            Verify installed packages have compatible
```

◆ 패키지(Package)

▪ pip 명령어 옵션

- | | |
|-----------------------|--|
| • pip search 패키지 | 패키지 검색 |
| • pip install 패키지==버전 | 특정 버전 패키지 설치
(예: pip install requests==2.9.0) |
| • pip list | 패키지 목록 출력 |
| • pip freeze | 패키지 목록 출력 |
| • pip uninstall 패키지 | 패키지 삭제 |