AI Programming

Lecture 13

```
def display_menu():
    print("₩n1. 연락처 추가")
    print("2. 연락처 삭제")
    print("3. 연락처 검색")
    print("4. 연락처 출력")
    print("5. 종료")
    sel = int(input("메뉴 항목을 선택하시오: "))
    return sel

def insert():
    global address_book
    name, number = get_contact()
    address_book[name] = number

def delete():
    name = get_name()
    if is_key(name):
        address_book.pop(name)
    else:
        print_error()

def search():
    name = get_name()
    if is_key(name):
        print_address(name)
    else:
        print_error()
```

```
def get_contact():
    name, number = get_name(), get_number()
    return name, number

def get_name():
    name = input("이름: ")
    return name

def get_number():
    number = input("전화번호: ")
    return number

def is_key(key):
    return address_book.get(key) != None

def print_error():
    print("Error: 주소록에 없는 이름!")

def print_address(name):
    print(f"{name}의 전화번호: {address_book[name]}")
```

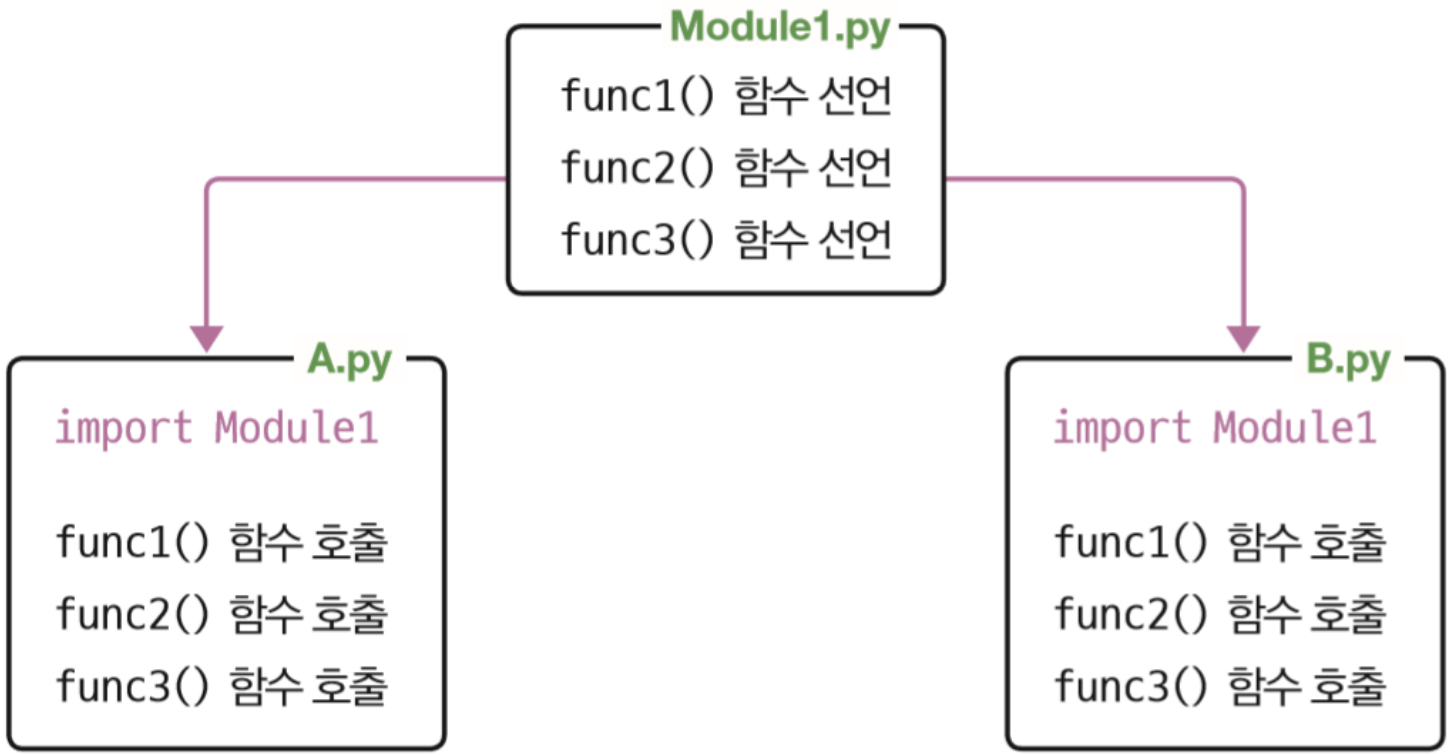
Preview

- **Ch. 9 함수와 모듈**
  - 9.5 모듈
  - 9.6 함수의 심화 내용

9.5 모듈

Modules

- **Module**
  - A set of functions
    - `import random`



Modules

- **Syntax**

Module1.py

```
## 함수 선언 부분 ##
def func1() :
    print("Module1.py의 func1()이 호출됨.")

def func2() :
    print("Module1.py의 func2()가 호출됨.")

def func3() :
    print("Module1.py의 func3()이 호출됨.")
```

A.py

```
import Module1

## 메인 코드 부분 ##
Module1.func1()
Module1.func2()
Module1.func3()
```

출력 결과

```
Module1.py의 func1()이 호출됨.
Module1.py의 func2()가 호출됨.
Module1.py의 func3()이 호출됨.
```

Modules

- **Syntax**
  - `import Module1`
    - `Module1.func1()`
  - `import Module1 as M1`
    - `M1.func1()`
  - `from Module1 import func1, func2, func3`
  - `from Module1 import *`
    - `func1(), func2(), func3()`

Modules

- **Import**

B.py

```
1  from Module1 import func1, func2, func3
2
3  ## 메인 코드 부분 ##
4  func1()
5  func2()
6  func3()
```

```
# 또는 from Module1 import *
```

출력 결과

```
Module1.py의 func1()이 호출됨.
Module1.py의 func2()가 호출됨.
Module1.py의 func3()이 호출됨.
```

Modules

- **Main part in module**

Module1.py - C:₩Users₩user₩Dropbox₩수업₩2

File Edit Format Run Options Window He

```
## 함수 선언 부분 ##
def func1():
    print("Call func1() in Module1.py")

def func2():
    print("Call func2() in Module1.py")

def func3():
    print("Call func3() in Module1.py")

## 메인 코드 부분 ##
print("Module1이 호출됨.")
```

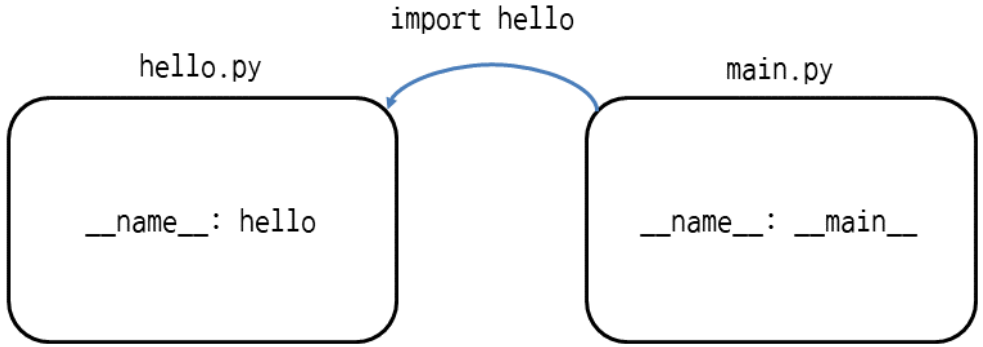
A.py

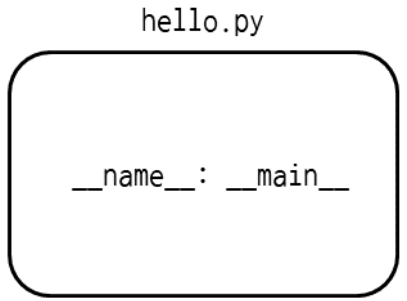
```
1  import Module1
2
3  ## 메인 코드 부분 ##
4  Module1.func1()
5  Module1.func2()
6  Module1.func3()
```

```
Module1이 호출됨.
Call func1() in Module1.py
Call func2() in Module1.py
Call func3() in Module1.py
```

Modules

- **Main part in module**
  - `if __name__ == '__main__' :`
    - `__name__`: Python 내부의 특수한 변수
  - Python 파일을 직접 실행
    - `__name__ = '__main__'` → 코드 실행
  - Python 파일을 `import`
    - `__name__ =` 모듈 이름 → 코드 미실행





Modules

- **Main part in module**

Module1.py - C:₩Users₩user₩Dropbox₩수업₩2

File Edit Format Run Options Window He

```
## 함수 선언 부분 ##
def func1():
    print("Call func1() in Module1.py")

def func2():
    print("Call func2() in Module1.py")

def func3():
    print("Call func3() in Module1.py")

## 메인 코드 부분 ##
if __name__ == "__main__":
    print("Module1이 호출됨.")
```

A.py

```
1 import Module1
2
3 ## 메인 코드 부분 ##
4 Module1.func1()
5 Module1.func2()
6 Module1.func3()
```

```
Call func1() in Module1.py
Call func2() in Module1.py
Call func3() in Module1.py
```

Modules

- **Types of modules**
  - Standard modules
    - Python에서 제공하는 모듈
  - Third party modules
    - Python이 아닌 외부 회사나 단체에서 제공하는 모듈
    - numpy, matplotlib, sklearn, pytorch, tensorflow
  - Custom modules

Modules

- **Python standard modules**

```
import sys
print(sys.builtin_module_names)
```

출력 결과

```
('_ast', '_bisect', '_codecs', '_codecs_cn', '_codecs_hk', '_codecs_iso2022', '_codecs_
jp', '_codecs_kr', '_codecs_tw', '_collections', '_csv', '_datetime', '_functools',
'_heapq', '_imp', '_io', '_json', '_locale', '_lsprof', '_md5', '_multibytecodec',
'_opcode', '_operator', '_pickle', '_random', '_sha1', '_sha256', '_sha512', '_signal',
'_sre', '_stat', '_string', '_struct', '_symtable', '_thread', '_tracemalloc',
'_warnings', '_weakref', '_winapi', 'array', 'atexit', 'audioop', 'binascii', 'builtins',
'cmath', 'errno', 'faulthandler', 'gc', 'itertools', 'marshal', 'math', 'mmap', 'msvcrt',
'nt', 'parser', 'sys', 'time', 'winreg', 'xxsubtype', 'zipimport', 'zlib')
```

Modules

- **Checking functions in a module**

```
import math
dir(math)
```

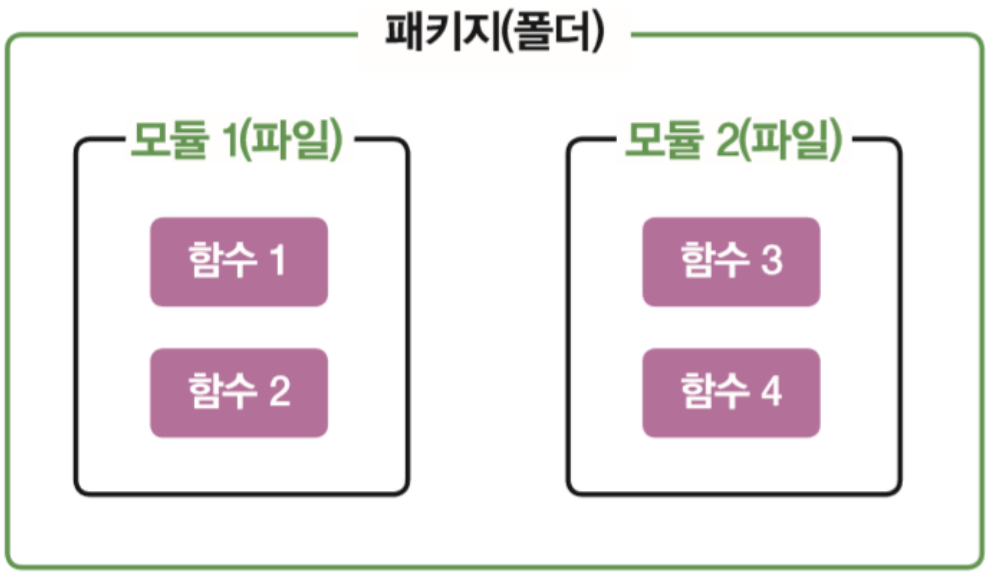
출력 결과

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin',
'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e',
'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum',
'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp',
'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin',
'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

9.6 함수의 심화 내용

Packages

- **Modules and Packages**
  - Module: a set of functions
    - *.py file
  - Package: a set of modules
    - Folder




```
from 패키지명.모듈명 import 함수명
```

```
from package.Module1 import *
```

Inner Functions

- **Inner functions**

```
def outFunc(v1, v2) :
    def inFunc(num1, num2) :
        return num1 + num2
    return inFunc(v1, v2)
print(outFunc(10, 20))
```

출력 결과

```
30
```

```
print(inFunc(10, 20))
```

출력 결과

```
NameError: name 'inFunc' is not defined
```

Lambda Function

- **Lambda function (함수 축약)**
  - `lambda parameters: expression`

```
def hap(num1, num2) :
    res = num1 + num2
    return res
print(hap(10, 20))
```

출력 결과

```
30
```

```
hap2 = lambda num1, num2 : num1 + num2
print(hap2(10, 20))
```

Lambda Function

- **Default parameter**

```
hap3 = lambda num1 = 10, num2 = 20 : num1 + num2
print(hap3())
print(hap3(100, 200))
```

출력 결과

```
30
300
```

Lambda Function

- **Lambda function and `map()`**
  - `map(함수명, 리스트명)`

```
myList = [1, 2, 3, 4, 5]
def add10(num) :
    return num + 10

for i in range(len(myList)) :
    myList[i] = add10(myList[i])
print(myList)
```

출력 결과

```
[11, 12, 13, 14, 15]
```

```
1  myList = [1, 2, 3, 4, 5]
2  add10 = lambda num : num + 10
3  myList = list(map(add10, myList))
4  print(myList)
```

```
myList = [1, 2, 3, 4, 5]
myList = list(map(lambda num : num + 10, myList))
print(myList)
```

Lambda Function

- `map()` **with multiple list inputs**

```
list1 = [1, 2, 3, 4]
list2 = [10, 20, 30, 40]
hapList = list(map(lambda n1, n2 : n1 + n2, list1, list2))
print(hapList)
```

출력 결과

```
[11, 22, 33, 44]
```

Recursion Function

- **Recursion function (재귀함수)**
  - A function that calls itself
  - Iterative calculations

```
def selfCall() :
    print('하', end = '')
    selfCall()
selfCall()
```

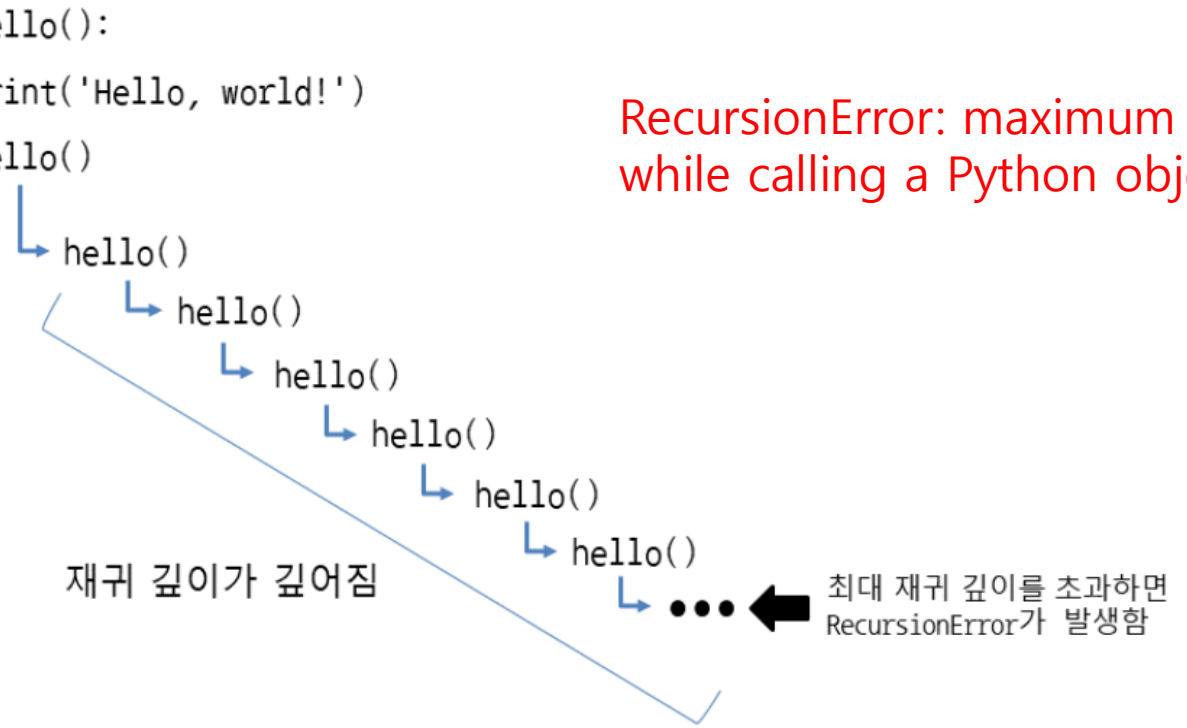
출력 결과

하하하하하하하하…

Recursion Function

- **RecursionError**

```
def hello():
    print('Hello, world!')
    hello()
```



RecursionError: maximum recursion depth exceeded while calling a Python object

Recursion Function

- **Exercise**

```
def count(num) :
    if num >= 1 :
        print(num, end = ' ')
        count(num - 1)
    else :
        return
count(10)
count(20)
```

출력 결과

```
10 9 8 7 6 5 4 3 2 1
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

Recursion Function

- **Exercise: factorial**

$$\texttt{factorial(num)} = \begin{cases} \texttt{num} & \text{if } \texttt{num} \leq 1 \\ \texttt{num} \times \texttt{factorial(num-1)} & \text{else} \end{cases}$$

```
def factorial(num) :
    if num <= 1 :
        return num
    else :
        return num * factorial(num - 1)
print(factorial(4))
print(factorial(10))
```

출력 결과

```
24
3628800
```

Recursion Function

- **Exercise: Fibonacci**
  - 0과 1을 제외하고 자신의 앞 숫자와 그 이전 숫자를 더하는 수열
  - 0, 1, 1, 2, 3, 4, 8, 13, 21, 34, 55, 89, 144,…

$$fib(n)=\begin{cases} 0 & \text{if } n==0 \\ 1 & \text{elif } n==1 \\ fib(n-2)+fib(n-1) & \text{else} \end{cases}$$

$$fibo(0)=0$$

$$fibo(1)=1$$

$$fibo(n)=fibo(n-1)+fibo(n-2),\ \text{for } n\geq 2$$

```
def fibo(n) :
    if  n == 0 :
        return 0
    elif n == 1 :
        return 1
    else  :
        return fibo(n-1) + fibo(n-2)

print(fibo(7))
print(fibo(20))
```

13
6765

Generator

- `return` **and** `yield`
  - `return`: return values and terminate a function.
  - `yield`: return values without terminating a function.

- **Generator (생성자)**
  - A function that includes `yield`

```
def genFunc() :
    yield 1
    yield 2
    yield 3

print(list(genFunc()))
```

출력 결과

```
[1, 2, 3]
```

Generator

- **Exercise**

```
1  def genFunc(num) :
2      for i in range(0, num) :
3          yield i
4          print('제너레이터 진행 중')
5  for data in genFunc(5) :
6      print(data)
```

출력 결과

```
0
제너레이터 진행 중
1
제너레이터 진행 중
2
제너레이터 진행 중
… 생략 …
```

Assignment 11

- **Recursion을 이용한 테일러 급수 모듈 구현**

$$\exp(x) \simeq \sum_{n=0}^{N} \frac{x^n}{n!} = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots + \frac{x^N}{N!}$$

$$\sin(x) \simeq \sum_{n=0}^{N} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots + \frac{(-1)^N}{(2N+1)!} x^{2N+1}$$

$$\cos(x) \simeq \sum_{n=0}^{N} \frac{(-1)^n}{(2n)!} x^{2n} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots + \frac{(-1)^N}{(2N)!} x^{2N}$$

Assignment 11

- **Recursion을 이용한 테일러 급수 모듈 구현**

```
함수의 입력 x: 2
테일러 급수의 차수 N: 3

----exp(x) 계산결과-----
math.exp(x): 7.3890561
테일러 근사: 6.3333333

----sin(x) 계산결과-----
math.sin(x): 0.9092974
테일러 근사: 0.9079365

----cos(x) 계산결과-----
math.cos(x): -0.4161468
테일러 근사: -0.4222222
```

```
함수의 입력 x: 2
테일러 급수의 차수 N: 10

----exp(x) 계산결과-----
math.exp(x): 7.3890561
테일러 근사: 7.3889947

----sin(x) 계산결과-----
math.sin(x): 0.9092974
테일러 근사: 0.9092974

----cos(x) 계산결과-----
math.cos(x): -0.4161468
테일러 근사: -0.4161468
```

Assignment 11

- **Recursion을 이용한 테일러 급수 모듈 구현**
  - 모듈 (Module2.py)
    - `factorial(N)`
    - `exponential(x, N)`
    - `sine(x, N)`
    - `cosine(x, N)`

```
def factorial(N):
    if N <= 1:
        return N
    else:
        return N*factorial(N-1)

def exponential(x, N):


def sine(x, N):


def cosine(x, N):


```