

컴퓨터과학 합격자반 4차시

서울과학고
김서진 T

CONTENTS

함수

파일 처리

예외 처리

모듈

문제

답지

함수

함수란?

```
def greet(name):  
    print(type(name))  
    return "Hello" + name
```

```
print(greet("Alice"))
```

```
<class 'str'>
```

```
HelloAlice
```

함수의 형식

```
def 함수명(매개변수):  
    수행문장  
    return 반환값
```

greet("Alice") 실행 시

- **수행문장:** " Alice " 의 type을 출력하고
- **return 반환값:** "HelloAlice"를 반환한다.

함수

default value 매개변수

```
def func1(a, b = 5, c = 10):  
    return a * b + c
```

```
print(func1(1, 2, 3))  
print(func1(1, 2))  
print(func1(1))
```

5
12
15

default value 매개변수

```
def 함수명(매개변수1, 매개변수2 = A):  
    수행문장  
    return 반환값
```

- 파이썬에서 매개변수의 디폴트 값을 지정할 수 있다.
- 이때, 디폴트 값이 없는 매개변수보다 디폴트 값이 지정된 매개변수가 앞에 위치해서는 안된다.

함수

가변 매개변수

```
def func2(*args):  
    a = 0  
    for i in args:  
        a = a + i  
    return a  
  
print(func2(1, 2, 3))  
print(func2(1, 0, 21, 23))
```

6
45

가변 매개변수

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

- 파이썬 함수에서 매개변수의 개수가 고정되지 않은 가변적인 경우도 가능
- 이때, **가변 매개변수는 튜플 형식으로 함수에 전달된다.**

함수

가변 매개변수

```
def student(n, *args):  
    print(f"학생의 수는 총{n}명입니다.")  
    print("학생 이름: ", end = "")  
    for name in args:  
        print(name, end = ' ')
```

```
student(3, 'rebro', 'john', 'minsu')
```

학생의 수는 총3명입니다.

학생 이름: rebro john minsu

- **n = 3**
- **args = ('rebro', 'john', 'minsu')**

함수

키워드 인자

```
def func3(**kwargs):  
    print(kwargs)  
  
func3(a = 2, b = True, c = 'abc')
```

```
{'a': 2, 'b': True, 'c': 'abc'}
```

키워드 인자

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

- 키워드 인자도 받을 수 있으며 이때, **딕셔너리 형식으로 함수에 전달된다.**
- 함수를 정의할 때는 매개변수 앞에 **를 붙인다.

```
def func4(a, b, c):  
    print(a+2, b-3, c+5)  
  
func4(b = 1, a = 2, c = 3)
```

4 -2 8

키워드 매개변수

- 지금까지는 함수에 인수를 넘겨주면 작성한 순서대로 매개변수에 인수가 할당됨
- 매개변수의 순서를 알고 있어야 한다는 단점이 있음
- 이를 해결하기 위해 파이썬에서는 '키워드 매개변수' 기능 제공
- '**매개변수명 = 값**' 으로 원하는 매개변수에 직접 값을 넣어줄 수 있다.

함수 정의 시 argument의 순서

```
def g1(a, b = 2, *args, c = 2, d, **kwargs):  
    pass  
  
def g2(a, b = 2, c = 3, d = 6):  
    pass  
  
def g3(*, a = 3, b = 2):  
    pass  
  
def g4(a = 3, b = 2, c):  
    pass
```

함수 정의 시 argument의 순서

- (1) 위치 변수 (a,b,...)
- (2) 디폴트 변수 (c=1, d=2,...)
- (3) 가변 매개변수 (*args)
- (4) 키워드로만 받을 수 있는 변수 (both default and non-default)
- (5) 키워드 인자 (**kwargs)

```
def g1(a, b = 2, *args, c = 2, d, **kwargs):  
    print(f'a = {a}, b = {b}')
```

```
g1(3, d = 0)  
a = 3, b = 2
```

```
g1(3, 12, d = 0)  
a = 3, b = 12
```

```
g1(a = 3, b = 12, d = 0)  
a = 3, b = 12
```

```
g1(2, a = 3, b = 12, d = 0)  
ERROR(a에 두 번 할당했기 때문)
```

위치 변수, 디폴트 변수

- **가변 매개변수 앞에 오는 변수들을 의미**
- 디폴트 지정값이 없는 변수들, 즉 **위치 변수가 반드시 앞에** 와야 함
- 주로 앞에서부터 채워지는 방식으로 값들이 할당되지만 키워드로 주어질 수도 있음

함수

가변 변수

```
def g1(a, b = 2, *args, c = 2, d, **kwargs):  
    print(f'a = {a}, b = {b}')  
    for i in range(len(args)):  
        print(f'args[{i}]: {args[i]}')
```

```
g1(2, 4, 12, 2, 3, 4, d = 2)
```

```
a = 2, b = 4  
args[0]: 12  
args[1]: 2  
args[2]: 3  
args[3]: 4
```

가변 변수

- 위치 변수, 디폴트 변수들을 모두 채운 후 **남은 변수들은 모두 args에 할당**(무조건 이름을 args로 할 필요는 없음. 단, 이것이 관례)
- args은 해당 값들을 담은 튜플임

함수

가변 변수

```
def g1(a, b = 2, *args, c = 2, d, **kwargs):  
    print(f'a = {a}, b = {b}')  
    for i in range(len(args)):  
        print(f'args[{i}]: {args[i]}')
```

```
g1(2, 4, 12, 2, 3, 4, d = 2)
```

```
a = 2, b = 4  
args[0]: 12  
args[1]: 2  
args[2]: 3  
args[3]: 4
```

가변 변수

- 위치 변수, 디폴트 변수들을 모두 채운 후 **남은 변수들은 모두 args에 할당**(무조건 이름을 args로 할 필요는 없음. 단, 이것이 관례)
- args은 해당 값들을 담은 **튜플임**

함수

키워드 변수

```
def g1(a, b = 2, *args, c = 2, d, **kwargs):  
    print(f'c = {c}, d = {d}')
```

```
g1(3, c = 4, d = 2)  
c = 4, d = 2
```

```
g1(2, d = 8)  
c = 2, d = 8
```

```
g1(2, d = 3, c = 1)  
c = 1, d = 3
```

```
g1(1, c = 3)  
ERROR(d 지정 X)
```

키워드 변수

- 가변 변수 뒤에 오는 변수 변수들
- 디폴트 값이 있는 변수와 디폴트 값이 없는 변수가 있음
(순서는 상관 X)
- 반드시 **(변수) = (값)**의 방식으로 값을 지정
- 디폴트 값이 없는 변수는 값을 지정해주지 않으면 ERROR 발생

함수

키워드 인자

```
def g1(a, b = 2, *args, c = 2, d, **kwargs):  
    print(kwargs)  
    for key in kwargs.keys():  
        print(f'kwargs[{key}] = {kwargs[key]}')
```

```
g1(3, c = 8, d = 10, e = 4, f = 1)
```

```
{'e': 4, 'f': 1}  
kwargs['e'] = 4  
kwargs['f'] = 1
```

키워드 인자

- **(변수) = (값)**의 방식으로 값을 지정해준 다음 남은 변수들은 kwargs 딕셔너리에 저장
- 함수 정의시 argument 중 가장 마지막에 등장

함수

종합

```
def g1(a, b = 2, *args, c = 2, d, **kwargs):  
    print(f'a = {a}, b = {b}')  
    for i in range(len(args)):  
        print(f'args[{i}]: {args[i]}')  
    print(f'c = {c}, d = {d}')  
    for key in kwargs.keys():  
        print(f'kwargs[{key}] = {kwargs[key]}')
```

```
a = 3, b = 4  
args[0]: 8  
args[1]: 2  
args[2]: 7  
c = 8, d = 10  
kwargs['e'] = 4  
kwargs['f'] = 1  
kwargs['h'] = 3
```

다음 중 올바르게 출력되도록 함수를 호출한 경우를 모두 고르시오.

1. `g1(3, 4, 8, 2, 7, c = 8, d = 10, e = 4, f = 1, h = 3)`
2. `g1(8, 2, 7, a = 3, b = 4, c = 8, d = 10, e = 4, f = 1, h = 3)`
3. `g1(a = 3, b = 4, 8, 2, 7, c = 8, d = 10, e = 4, f = 1, h = 3)`
4. `g1(3, 4, 8, 2, 7, e = 4, f = 1, h = 3, c = 8, d = 10)`

함수

예시: print 함수

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

- `*objects`: str로 변환하여 print할 값들을 받음
- `sep=' '`: objects의 값들을 출력할 때 사이에 넣을 구분자를 받음. default 값은 ' ', 즉 한칸 띄어쓰기
- `end='\n'`: 앞선 값들을 모두 출력한 후 마지막으로 출력할 값을 받음. default 값은 '\n', 즉 개행
- `file=sys.stdout`: 출력할 파일을 지정. default 값은 sys.stdout, 즉 우리가 볼 수 있는 콘솔

```
print('Hello')  
print('a', 'b', 'c', sep = '*', end = ' ')  
print('c', 'd', end = '!', sep = '&')
```

```
Hello  
a*b*c c&d!
```


함수

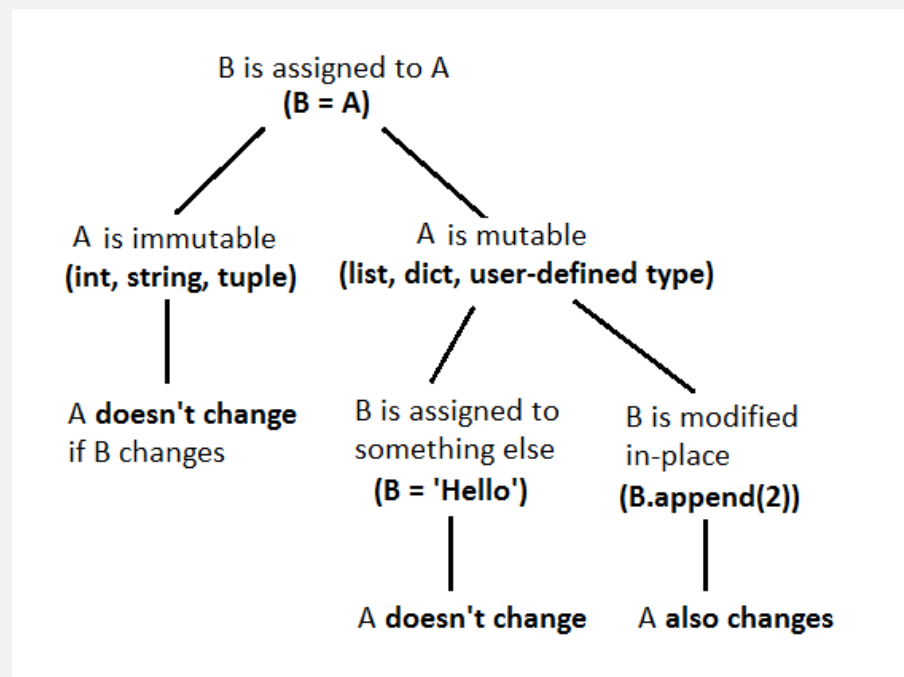
Call by object reference

```
def list_modify1(arr):  
    arr.append(5)
```

```
def list_modify2(arr):  
    arr = [5, 6]
```

```
list1 = [1, 2, 3, 4]  
list2 = [1, 2, 3, 4]  
list_modify1(list1)  
list_modify2(list2)  
print(list1)  
print(list2)
```

```
[1, 2, 3, 4, 5]  
[1, 2, 3, 4]
```



파이썬의 모든 객체 전달 방식은 call by object reference

- **list는 immutable 객체**
- list를 modify하면 해당 list를 참조하고 있는 모든 변수가 바뀐다.

함수

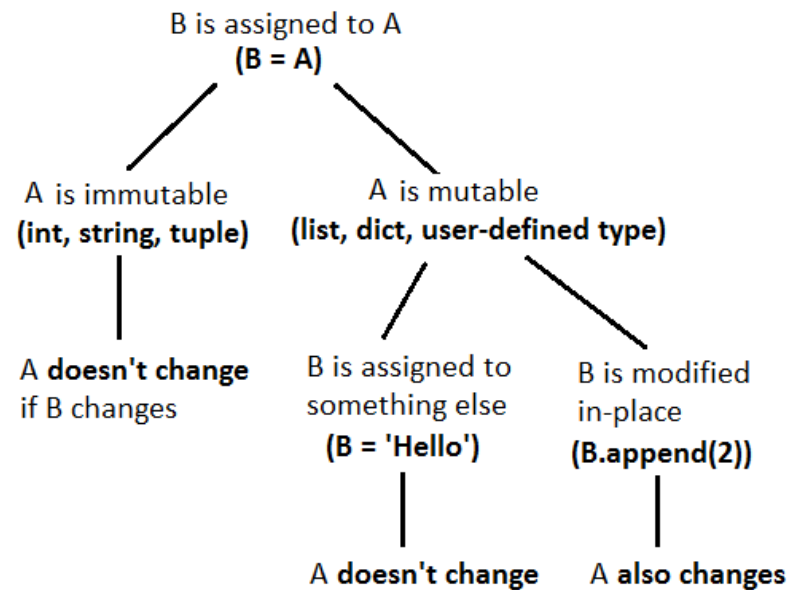
Call by object reference

```
def int_modify(x):  
    x += 3  
    print(f'x in function: {x}')
```

```
x = 1  
print(f'initial x: {x}')
```

```
int_modify(x)  
print(f'final x: {x}')
```

```
initial x: 1  
x in function: 4  
final x: 1
```



파이썬의 모든 객체 전달 방식은 call by object reference

- **int는 immutable 객체**
- int의 값을 바꾸면 새로운 객체를 가리키게 됨

함수

함수를 인자로 전달

```
def square(x):  
    return x ** 2
```

```
def cube(x):  
    return x ** 3
```

```
def sum_f(n, f):  
    sum = 0  
    for i in range(n):  
        sum += f(i)  
    return sum
```

```
print(f'sum_f(5, square): {sum_f(5, square)}')  
print(f'sum_f(5, cube): {sum_f(5, cube)}')
```

```
sum_f(5, square): 30  
sum_f(5, cube): 100
```

함수를 인자로 전달

- 파이썬에서 함수는 일급 객체
- 다른 함수에 인자로 함수를 전달할 수 있음
- 이러한 일들을 편리하게 해주는 것이 lambda

함수

lambda

```
square = lambda x: x ** 2
cube = lambda x: x ** 3

def sum_f(n, f):
    sum = 0
    for i in range(n):
        sum += f(i)
    return sum

print(f'sum_f(5, square): {sum_f(5, lambda x: x ** 2)}')
sum_f(5, square): 30

print(f'sum_f(5, cube): {sum_f(5, lambda x: x ** 3)}')
sum_f(5, cube): 100
```

```
sum_f(5, square): 30
sum_f(5, cube): 100
```

lambda

- **lambda (인자): (표현식)**
- 간단한 함수를 선언하고 이름 붙이는 과정 없이 편리하게 사용할 수 있음
- 인자가 여러 개인 것도 가능

함수

map, filter

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
odds = filter(lambda x: x % 2 != 0, numbers)  
squared_odds = map(lambda x: x**2, odds)
```

```
squared_odds_list = list(squared_odds)  
print(squared_odds_list)
```

```
[1, 9, 25, 49, 81]
```

filter

- filter(함수, 리스트)
- 리스트의 각 요소들을 함수에 넣고 리턴된 값이 True인 것으로 filter 객체를 구성해주는 함수

map

- map(함수, 리스트)
- 리스트의 각 요소들을 함수에 넣고 리턴된 값으로 map 객체를 구성해주는 함수

함수

재귀함수

```
def factorial(n):  
    if n == 0:  
        return 1  
    else:  
        return n * factorial(n-1)
```

```
print(f'5! = {factorial(5)}')
```

5! = 120

```
return n * factorial(n - 1);
```

- 재귀함수: 함수에서 자기 자신을 다시 호출해 작업을 수행하는 방식

```
factorial(5)  
= 5 * factorial(4)  
= 5 * 4 * factorial(3)  
= 5 * 4 * 3 * factorial(2)  
= 5 * 4 * 3 * 2 * factorial(1)  
= 5 * 4 * 3 * 2 * 1 * factorial(0)  
= 5 * 4 * 3 * 2 * 1 * 1  
= 120
```

함수

재귀함수 예시: 하노이 탑

Problem

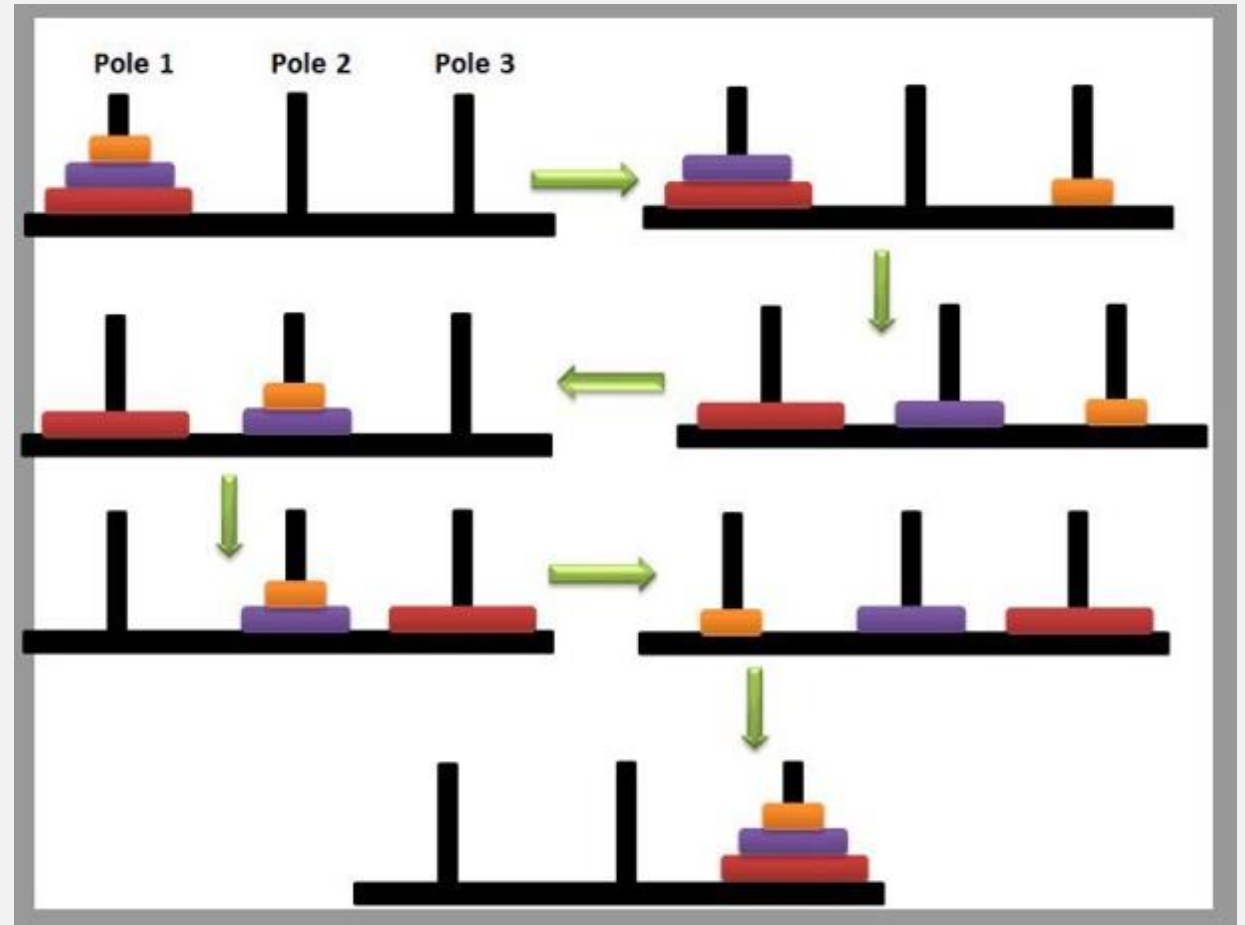
하노이의 탑(Tower of Hanoi)은 퍼즐의 일종이다.

세 개의 기둥과 이 기둥에 꽂을 수 있는 크기가 다양한 원판들이 있고, 퍼즐을 시작하기 전에는 한 기둥에 원판들이 작은 것이 위에 있도록 순서대로 쌓여 있다.

게임의 목적은 다음 두 가지 조건을 만족시키면서, 한 기둥에 꽂힌 원판들을 그 순서 그대로 다른 기둥으로 옮겨서 다시 쌓는 것이다.

1. 한 번에 한개의 원판만 옮길 수 있다.
2. 가장 위에 있는 원판만 이동할 수 있다.
3. 큰 원판이 작은 원판 위에 있어서는 안 된다.

Example



함수

재귀함수 예시: 하노이 탑

```
def move_disk(from_peg, to_peg, i):  
    print(f"Move disk {i} from '{from_peg}' to '{to_peg}'")  
  
def solve_hanoi(n, from_peg, to_peg, aux_peg):  
    if n == 1:  
        move_disk(from_peg, to_peg, n)  
        return  
    solve_hanoi(n - 1, from_peg, aux_peg, to_peg)  
    move_disk(from_peg, to_peg, n)  
    solve_hanoi(n - 1, aux_peg, to_peg, from_peg)  
  
solve_hanoi(4, 'A', 'C', 'B')
```

Move disk 1 from 'A' to 'C'
Move disk 2 from 'A' to 'B'
Move disk 1 from 'C' to 'B'
Move disk 3 from 'A' to 'C'
Move disk 1 from 'B' to 'A'
Move disk 2 from 'B' to 'C'
Move disk 1 from 'A' to 'C'

```
moveDisk(fromPeg, toPeg, int i)
```

- "Move disk **i** from **fromPeg** to **toPeg**"를 print하는 함수

```
void solveHanoi(n, fromPeg, toPeg, auxPeg)
```

- 1 ~ n 원판을 **fromPeg**을 **toPeg**로 옮기는 작업을 실행하는 함수

1. 먼저 1 ~ n-1 원판을 **fromPeg**에서 **auxPeg**로 옮긴다.
2. n 원판을 **fromPeg**에서 **toPeg**로 옮긴다.
3. 1 ~ n-1 원판을 **auxPeg**에서 **toPeg**로 옮긴다.

CONTENTS

함수

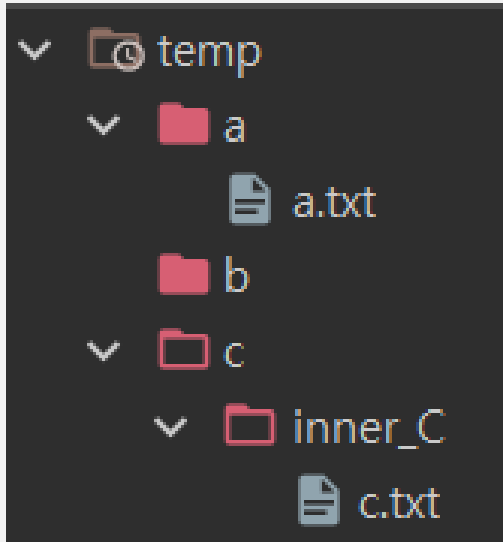
파일 처리

예외 처리

모듈

문제

답지



절대 경로

C:\temp\a\c.txt

루트에서 해당 파일까지 이르는 경로

상대 경로

c.txt에서 작업을 하고 있을 때, a.txt파일로 이동하고 싶다면

../../a/a.txt

a.txt에서 c.txt를 향하고 싶으면

../c/inner_C/c.txt

파일 처리

open()

```
str = """Hello\nWorld\nHello\nABC"""  
f = open('./data.txt', 'w')  
f.write(str)  
f.close()
```

```
≡ data.txt  
1 Hello  
2 World  
3 Hello  
4 ABC
```

```
f = open('./data.txt', 'w')
```

- 경로를 받아 해당 파일을 열 수 있음
- 'w'를 받았음(write) -> 기존에 동명의 파일이 있으면 덮어쓰고 없으면 새롭게 만들음
- f.write(str)을 통해 문자열을 쓸 수 있음
- f.close()를 하여 파일을 닫아줘야 함

파일 처리

readline()

```
f = open('./data.txt', 'r')  
print(f.readline(), end = "")  
print(f.readline(), end = "")  
f.close()
```

Hello
World

readline()

- 파일의 각 줄을 읽어옴(\n이 나올 때까지 쪽 읽어옴)

```
f = open('./data.txt', 'r')  
for line in f:  
    print(line, end = "")  
f.close()
```

```
Hello  
World  
Hello  
ABC
```

for line in f:

- f를 iter하면 각 객체들은 str(각 줄들을 iter하게 됨)

파일 처리

seek

```
f = open('./data.txt', 'r')
print(list(f))
f.seek(0)
print(f.readlines())
f.close()
```

```
['Hello\n', 'World\n', 'Hello\n', 'ABC']
['Hello\n', 'World\n', 'Hello\n', 'ABC']
```

f.readlines()

- 각 줄들을 담은 리스트를 반환

f.seek(0)

- f.seek(0)이 없으면 두 번째 리스트는 비어있음
- list(f)를 하며 이미 파일을 전부 읽었기 때문
- f.seek(0)를 하면 파일의 처음으로 돌아가 다시 읽을 수 있음

파일 처리

with문

```
with open('./data.txt', 'r') as f:
```

```
    f.read()
```

```
print(f.closed)
```

```
f = open('./data.txt', 'r')
```

```
print(f.closed)
```

True

False

```
with open('./data.txt', 'r') as f:
```

- with문을 사용하면 파일 처리 작업이 끝난 후 파일을 닫는 작업이 자동으로 이루어지도록 할 수 있음

파일 처리

%%writefile

```
%%writefile a.txt  
Hello
```

≡ a.txt

```
1 Hello  
2 |
```

```
%%writefile a.txt
```

- %%writefile을 이용하면 셸 내에서 a.txt 파일에 넣을 텍스트 내용을 적어줌으로써 파일을 만들 수 있다.

파일 처리

파일 열기 모드

```
s = "\nxxx"
with open('./data.txt', 'a+') as f:
    f.write(s)
    f.seek(0)
    for line in f:
        print(line, end = "")
```

```
Hello
World
Hello
ABC
xxx
```

a+

- 파일 열기 모드를 a+로 지정해주면 파일의 마지막에 새로운 내용을 추가할 수 있으며 파일을 처음부터 읽을 수도 있다.

파일 열기모드	의미
r	읽기 모드: 파일을 읽기만 할 때 사용한다.
w	쓰기 모드: 파일에 내용을 쓸 때 사용한다.
a+	추가 모드: 파일의 마지막에 새로운 내용을 추가할 때 사용한다. 또한 파일을 처음부터 읽을 수도 있다.

파일 처리

pickle

```
import pickle
list = ['a', 'b', 'c']
with open('list.pickle', 'wb') as f:
    pickle.dump(list, f)
with open('list.pickle', 'rb') as f:
    data = pickle.load(f)
data
```

['a', 'b', 'c']

```
import pickle
```

- **pickle 모듈**을 이용하면 객체를 바이너리 파일로 저장할 수 있음
- **pickle.dump(list, f)**: 객체를 파일 경로에 바이너리 파일로 저장
- **pickle.load(f)**: 경로에 저장되어 있는 바이너리 파일을 다시 객체에 저장

CONTENTS

함수

파일 처리

예외 처리

모듈

문제

답지

예외 처리

예외 처리

```
try:
    number = int(input('나눌 숫자를 입력하세요: '))
    result = 10/number
except ValueError:
    print('숫자만 입력하세요.')
except ZeroDivisionError:
    print('0으로 나누면 안됩니다.')
except:
    print('그 외의 에러')
else:
    print(result)
finally:
    print('무조건 실행되는 문장')
```

try, except, else, finally

- **try:** 에러가 발생한 위험이 있는 구문
- **except:** try 내에서 해당 에러가 발생하면 실행
- **else:** 에러가 발생하지 않으면 실행
- **finally:** 반드시 실행

```
class MyException(Exception):  
    def __str__(self):  
        return "홀수"
```

```
def say_even(number):  
    if number % 2 == 1:  
        raise MyException()  
    print("짝수")
```

```
try:  
    say_even(1)  
except MyException as err:  
    print(err)
```

홀수

```
class MyException(Exception):
```

- 모든 Exception들은 Exception 클래스를 상속받음
- 커스텀한 Exception 클래스를 만들 수 있음

예외 처리

assert

```
try:  
    number = -42  
    assert number > 0  
except AssertionError as err:  
    print("AssertionError")
```

AssertionError

```
assert number > 0
```

- 특정 조건을 만족하지 않으면 에러가 뜨도록 assert문을 사용할 수 있음

CONTENTS

함수

파일 처리

예외 처리

모듈

문제

답지

모듈

모듈

a.py

```
def add(a, b):  
    return a + b  
  
def sub(a, b):  
    return a - b  
  
if __name__ == "__main__":  
    print("This is main file")  
else:  
    print("a.py is imported")
```

b.py

```
import a  
print(a.add(1, 3))  
print(a.sub(1, 3))
```

c.py

```
from a import add, sub  
print(add(1, 3))  
print(sub(1, 3))
```

d.py

```
from a import *  
print(add(1, 3))  
print(sub(1, 3))
```

Terminal

```
gimjiho@gimjihoui-MacBookAir 서울 1-1 % python3 a.py  
This is main file  
gimjiho@gimjihoui-MacBookAir 서울 1-1 % python3 b.py  
a.py is imported  
4  
-2  
gimjiho@gimjihoui-MacBookAir 서울 1-1 % python3 c.py  
a.py is imported  
4  
-2  
gimjiho@gimjihoui-MacBookAir 서울 1-1 % python3 d.py  
a.py is imported  
4  
-2
```

__name__이란?

- 파일을 직접 실행하면 `__name__ == "__main__"`
- 파일이 다른 파일에 의해 `import` 되었으면 `__name__`은 모듈명
- a.py의 경우 `__name__ == "a"`

import

- 다른 모듈의 함수, 클래스, 변수 등을 사용할 수 있음
- 터미널에서 python을 실행시키는 디렉토리에 a.py, b.py, c.py, d.py가 있으므로 import 가능

모듈

random

```
import random
```

```
a = random.random()
```

```
b = random.randint(0, 100)
```

```
c = random.randrange(0, 100, 2)
```

```
print(a, b, c, sep = '\n')
```

```
0.6651708893593375
```

```
69
```

```
30
```

random 모듈

- 난수 생성 등 랜덤과 관련된 함수들을 포함하고 있는 built-in 모듈
- **random.random()**: 0~1의 랜덤한 실수를 반환
- **random.randint(0, 100)**: 0 이상 100 이하의 랜덤한 정수를 반환
- **random.randrange(0, 100, 2)**: range(0, 100, 2)에 해당되는 범위의 랜덤한 정수를 반환

CONTENTS

함수

파일 처리

예외 처리

모듈

문제

답지

문제

P	1. 다음은 많은 사람들이 파이썬으로 프로그램을 개발하다가 막히는 대표적인 코드이다. 빈칸을 채워서 실행 결과처럼 출력되게 만들어보자.
C	<pre>numbers = [1, 2, 3, 4, 5, 6] print('::'.join(-----A-----))</pre>
R	1::2::3::4::5::6
A	

P	2. 주어진 리스트를 flatten해서 반환하는 함수를 작성해보자. 단, isinstance, exten를 활용해도 좋다.
C	<pre>def flatten_list(nested_list): # to do nested_list = [1, [2, [3, 4], 5], 6, [[7], [8, 9]]] flattened_list = flatten_list(nested_list) flattened_list</pre>
R	[1, 2, 3, 4, 5, 6, 7, 8, 9]
A	

문제

P	3. 짝수는 그대로, 홀수는 제곱하여 리스트를 반환해보시오
C	<pre>numbers = [1, 2, 3, 4, 5, 7] print(list(-----A-----))</pre>
R	[1, 2, 9, 4, 25, 49]
A	

P	4. 앞뒤로 대칭인 문자열을 팰린드롬이라 칭한다. 팰린드롬인 문자열만 남기시오.
C	<pre>words = ["radar", "python", "level", "hello", "world"] print(list(-----A-----))</pre>
R	['radar', 'level']
A	

문제

P

5. 두 리스트의 원소들을 묶어서 튜플들로 만들어 리스트를 반환하시오.

C

```
list1 = [1, 2, 3]
list2 = ['a', 'b', 'c']
print(list(-----A-----))
```

R

`[(1, 'a'), (2, 'b'), (3, 'c')]`

A

P

6. operations는 순서대로 적용할 함수의 리스트이다.
모두 적용한 후 result를 반환하는 코드를 작성하시오.

C

```
operations = [lambda x: x + 2, lambda x: x * 3, lambda x: x - 4]

start_value = 10

result = start_value
# to do

print(result)
```

R

`32`

A

문제

P	7. 재귀함수를 이용하여 문제 6과 같은 결과를 출력할 수 있는 코드를 작성하시오.
C	<pre>operations = [lambda x: x + 2, lambda x: x * 3, lambda x: x - 4] start_value = 10 def do_operations(start_value, operations): return (-----A-----) do_operations(start_value, operations)</pre>
R	32
A	

P	8. 주어진 집합의 멱집합을 구하는 코드를 작성하시오.
C	<pre>def power_set(s): # to do original_set = [1, 2, 3] power_set_result = power_set(original_set) power_set_result</pre>
R	[[], [1], [2], [1, 2], [3], [1, 3], [2, 3], [1, 2, 3]]
A	

문제

P	9. filter function과 map function을 받아 순차적으로 filter, map을 진행하는 코드를 작성하시오.
C	<pre>def filter_map(numbers, filter_function, map_function): return (-----A-----) numbers = [1, 2, 3, 4, 5] result = filter_map(numbers, lambda x: x % 2 != 0, lambda x: x ** 2) result</pre>
R	32
A	

P	10. list comprehension, filter, map을 이용하지 않고 재귀함수로 9를 구현해보시오.
C	<pre>def recursive_filter_map(numbers, filter_function, map_function): # to do numbers = [1, 2, 3, 4, 5] result = recursive_filter_map(numbers, lambda x: x % 2 != 0, lambda x: x ** 2) result</pre>
R	32
A	

CONTENTS

함수

파일 처리

예외 처리

모듈

문제

답지

문제

P

1. 다음은 많은 사람들이 파이썬으로 프로그램을 개발하다가 막히는 대표적인 코드이다. 빈칸을 채워서 실행 결과처럼 출력되게 만들어보자.

C

```
numbers = [1, 2, 3, 4, 5, 6]
print('::'.join(-----A-----))
```

R

1::2::3::4::5::6

A

```
map(lambda x: str(x), numbers)
```

P

2. 주어진 리스트를 flatten해서 반환하는 함수를 작성해보자.
단, isinstance, extend를 활용해도 좋다.

C

```
def flatten_list(nested_list):
    # to do
```

```
nested_list = [1, [2, [3, 4], 5], 6, [[7], [8, 9]]]
flattened_list = flatten_list(nested_list)
flattened_list
```

R

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

A

```
def flatten_list(nested_list):
    flat_list = []
    for item in nested_list:
        if isinstance(item, list):
            flat_list.extend(flatten_list(item))
        else:
            flat_list.append(item)
    return flat_list
```

문제

P

3. 짝수는 그대로, 홀수는 제곱하여 리스트를 반환해보시오

C

```
numbers = [1, 2, 3, 4, 5, 7]
print(list(-----A-----))
```

R

[1, 2, 9, 4, 25, 49]

A

```
map(lambda x: x**2 if x % 2 != 0 else x, numbers)
```

P

4. 앞뒤로 대칭인 문자열을 팰린드롬이라 칭한다.
팰린드롬인 문자열만 남기시오.

C

```
words = ["radar", "python", "level", "hello", "world"]
print(list(-----A-----))
```

R

['radar', 'level']

A

```
filter(lambda word: word == word[::-1], words)
```

문제

P 5. 두 리스트의 원소들을 묶어서 튜플들로 만들어 리스트를 반환하시오.

C

```
list1 = [1, 2, 3]
list2 = ['a', 'b', 'c']
print(list(-----A-----))
```

R [(1, 'a'), (2, 'b'), (3, 'c')]

A list(map(lambda x, y: (x, y), list1, list2))

P 6. operations는 순서대로 적용할 함수의 리스트이다.
모두 적용한 후 result를 반환하는 코드를 작성하시오.

C

```
operations = [lambda x: x + 2, lambda x: x * 3, lambda x: x - 4]

start_value = 10

result = start_value
# to do

print(result)
```

R 32

A

```
for operation in operations:
    result = operation(result)
```

문제

P	7. 재귀함수를 이용하여 문제 6과 같은 결과를 출력할 수 있는 코드를 작성하시오.
C	<pre>operations = [lambda x: x + 2, lambda x: x * 3, lambda x: x - 4] start_value = 10 def do_operations(start_value, operations): return (-----A-----) do_operations(start_value, operations)</pre>
R	32
A	<pre>operations[0](start_value) if len(operations) == 1 else do_operations(operations[0](start_value), operations[1:])</pre>

P	8. 주어진 집합의 멱집합을 구하는 코드를 작성하시오.
C	<pre>def power_set(s): # to do original_set = [1, 2, 3] power_set_result = power_set(original_set) power_set_result</pre>
R	[[], [1], [2], [1, 2], [3], [1, 3], [2, 3], [1, 2, 3]]
A	<pre>def power_set(s): if len(s) == 0: return [[]] else: main_subset = power_set(s[:-1]) additional_subsets = [subset + [s[-1]] for subset in main_subset] return main_subset + additional_subsets</pre>

문제

P

9. filter function과 map function을 받아 순차적으로 filter, map을 진행하는 코드를 작성하시오.

C

```
def filter_map(numbers, filter_function, map_function):  
    return (-----A-----)  
  
numbers = [1, 2, 3, 4, 5]  
result = filter_map(numbers, lambda x: x % 2 != 0, lambda x: x ** 2)  
result
```

R

32

A

```
return [map_function(number) for number in numbers if  
filter_function(number)]
```

P

10. list comprehension, filter, map을 이용하지 않고 재귀함수로 9를 구현해보시오.

C

```
def recursive_filter_map(numbers, filter_function, map_function):  
    # to do  
  
numbers = [1, 2, 3, 4, 5]  
result = recursive_filter_map(numbers, lambda x: x % 2 != 0, lambda x: x ** 2)  
result
```

R

32

A

```
def recursive_filter_map(L, filter_function, map_function):  
    if not L:  
        return []  
    else:  
        head = L[0]  
        tail = L[1:]  
        filtered_tail = recursive_filter_map(tail, filter_function, map_function)  
        if filter_function(head):  
            return [map_function(head)] + filtered_tail  
        else:  
            return filtered_tail
```