파이썬프로그래밍 2분반 ## <13주차 과제> ---

정보보안공학과 ### 202121556 ### 곽지현

2023-06-01

In [20]:
```
# 1. 튜플의 이해를 위해 다음 코드를 실행하여 결과를 확인해보세요.

a = tuple([1, 2, 3]) # 튜플 선언
b = (4, 5, 6, 7) # 튜플 선언

print("tuple a:", a) # a를 출력
print("tuple b:", b) # b를 출력

print("tuple a[1]:", a[1]) # a의 1번 인덱스 출력
print("tuple b[1:3]", b[1:3]) # b의 1번, 2번 인덱스 출력

a[0] = 3 # 값 수정 불가
```

```
tuple a: (1, 2, 3)
tuple b: (4, 5, 6, 7)
tuple a[1]: 2
tuple b[1:3] (5, 6)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~WAppDataWLocalWTempWipykernel_14808W2111221213.py in <module>
     10 print("tuple b[1:3]", b[1:3])
     11
---> 12 a[0] = 3

TypeError: 'tuple' object does not support item assignment
```

In [2]:
```
# 2. 괄호 없는 튜플의 이해를 위해 다음 코드를 실행하여 결과를 확인해보세요.

c = 1, 2, 3, 4 # 튜플 선언

print("c:", c, type(tuple)) # c를 출력

val1, val2, val3, val4 = 1, 2, 3, 4 # 튜플 선언

print("[Unpacking] val1:", val1) # val1을 출력
print("[Unpacking] val2:", val2) # val2를 출력
print("[Unpacking] val3:", val3) # val3을 출력
print("[Unpacking] val4:", val4) # val4를 출력
```

```
c: (1, 2, 3, 4) <class 'type'>
[Unpacking] val1: 1
[Unpacking] val2: 2
[Unpacking] val3: 3
[Unpacking] val4: 4
```

In [9]:
```
# 3. 튜플의 함께 사용한 함수의 이해를 위해 다음 코드를 실행하여 결과를 확인해보세요.

def tuple_test(tuple_parameter): # tuple_test 함수 선언
    print("[tuple_test] tuple_parameter:", tuple_parameter) # tuple_parameter 출력
    val1 = tuple_parameter[0] * 2
    # tuple_parameter의 0번 인덱스에 곱하기 2를 하고 val1에 대입
    val2 = tuple_parameter[1] * 2
    # tuple_parameter의 1번 인덱스에 곱하기 2를 하고 val2에 대입

    return val1, val2 # val1, val2 리턴

return1 = tuple_test((2, 5))
# tuple_test(2, 5) 함수 호출하고 결과를 return1에 대입
print("return1:", return1) # return1의 값을 출력

return2, return3 = tuple_test(return1)
# tuple_test(return1) 함수 호출하고 결과를 각 각 return2와 return3에 대입
print("return2:", return2) # return2의 값을 출력
print("return3:", return3) # return3의 값을 출력
```

```
[tuple_test] tuple_parameter: (2, 5)
return1: (4, 10)
[tuple_test] tuple_parameter: (4, 10)
return2: 8
return3: 20
```

In [10]:
```
# 4. 함수를 매개변수로 전달하는 표준함수 filter와 map 함수의 이해를 위해 다음 코드를 실

string_list = ["1", "2", "3", "4", "a", "b", "c"] # 리스트 선언

num_string_filter = filter(str.isdecimal, string_list)
# filter() 함수 사용, string_list안에 정수형 문자열만 num_string_fliter에 대입
print(num_string_filter, type(num_string_filter))
# num_string_filter 출력, type도 출력
num_string_list = list(num_string_filter)
# num_string_filter의 리스트를 num_string_filter에 대입

int_map = map(int, num_string_list)
# map() 함수 사용, num_string_list의 정수를 int_map에 대입
print(int_map, type(int_map))
# int_map 출력, type도 출력
int_list = list(int_map)
# int_map의 리스트를 int_list에 대입

print("{} -> {} -> {}" .format(string_list, list(num_string_list), list(int_list)))
# string_list, list(num_string_list), list(int_list)를 출력
```

```
<filter object at 0x000001B3CBBBA0A0> <class 'filter'>
<map object at 0x000001B3CBBC5820> <class 'map'>
['1', '2', '3', '4', 'a', 'b', 'c'] -> ['1', '2', '3', '4'] -> [1, 2, 3, 4]
```

In [11]:
```
# 5. 람다의 이해를 위해 다음 코드를 실행하여 결과를 확인해보세요.

string_converter = lambda parameter: str(parameter)
# lambda() 함수를 선언, parameter을 문자열로 바꿈


num_list = [1, 2, 3, 4, 5] # 리스트 선언

convert_result = string_converter(num_list)
```

```
# num_list의 리스트를 lambda() 함수를 사용해 문자열로 바꾸고 convert_result에 대입
print("{} {} -> {} {}" .format(num_list, type(num_list), convert_result, type(convert_r
# num_list, type(num_list), convert_result, type(convert_result)을 출력

for x in num_list: # num_list의 값들이 x에 하나씩 들어감
    convert_result = string_converter(x)
    # x를 lambda() 함수를 사용해 문자열로 바꾸고 conver_result에 대입
    print("{} {} -> {} {}" .format(x, type(x), convert_result, type(convert_result)))
    # x, type(x), convert_result, type(convert_result)을 출력
```

```
[1, 2, 3, 4, 5] <class 'list'> -> [1, 2, 3, 4, 5] <class 'str'>
1 <class 'int'> -> 1 <class 'str'>
2 <class 'int'> -> 2 <class 'str'>
3 <class 'int'> -> 3 <class 'str'>
4 <class 'int'> -> 4 <class 'str'>
5 <class 'int'> -> 5 <class 'str'>
```

In [12]:
```
# 6. 파일 쓰기의 이해를 위해 다음 코드를 실행하여 결과를 확인해보세요.

file1 = open("test.txt", "wt") # 파일을 쓰기모드로 연다.

file1.write("File Write Test1!₩nFile Write Test2!")
# file1에 "File Write Test1!₩nFile Write Test2!"를 쓴다.
file1.close() # 파일을 닫는다.

with open("test2.txt", "wt") as f: # 파일을 쓰기모드로 연다.
    f.write("with open test") # f에 "with open test"를 쓴다.
    f.close() # 파일을 닫는다. (with 키워드를 쓰면 닫지 않아도 실행된다.)
```

In [13]:
```
# 7. 파일 읽기의 이해를 위해 6번을 진행한 후 다음 코드를 실행하여 결과를 확인해보세요.

file1 = open("test.txt", "rt", encoding = "utf8") # 파일을 읽기모드로 연다.
file_contents = file1.readlines()
# readlines() 함수 사용, 파일의 첫 번째 줄부터 한 줄씩 file_contents에 대입

[print(file_line) for file_line in file_contents]
# file_contents안에 한 줄씩 읽어 들여 출력

with open("test2.txt", "rt", encoding = "utf8") as f: # 파일을 읽기모드로 연다.
    [print(file_line) for file_line in f.readlines()]
    # readlines() 함수 사용, 파일의 첫 번째 줄부터 한 줄씩 읽어 들여 출력
    f.close() # 파일을 닫는다.
```

```
File Write Test1!

File Write Test2!
with open test
```

In [1]:
```
# 8. 제너레이터의 이해를 위해 다음 코드를 실행하여 결과를 확인해보세요.

import time # 시간과 관련된 기능을 가져옴

def test_generator(n): # test_generator 함수 선언
    for x in range(n): # n의 값 만큼 반복
        time.sleep(1) # 1초간 일시정지
        yield x # yield 키워드 사용

def test_return(n): # test_return 함수 선언
    result = [] # 빈 리스트 선언
    for x in range(n): # n의 값 만큼 반복
```

```
        time.sleep(1) # 1초간 일시정지
        result.append(x) # x의 값을 리스트에 추가
    return result # result 리턴

for n in test_return(3): # test_return 함수 호출
    print(n) # n을 출력

print() # 줄 바꿈

for n in test_generator(3): # test_generator 함수 호출
    print(n) # n을 출력
```

```
0
1
2

0
1
2
```

In [7]:
```
# 10. 텍스트 파일을 제너레이터로 한 줄씩 읽는 프로그램입니다. 다음과 같은 결과가 나오도

def read_lines(file_object): # read_lines 함수 선언
    while(True): # True 이므로 무한 반복
        line = file_object.readline()
        # readline() 함수 사용, file_object의 첫 번째 줄부터 한 줄씩 line에 대입
        if not line: # line이 없다면
            break; # 종료
        print("[generator] line : {}" .format(line)) # line을 출력

        yield line, len(line) # line과 line의 길이를 내보낸다.

with open("애국가.txt", "rt", encoding="utf8") as file: # 파일을 읽기모드로 연다.
    for line_content, line_data_cnt in read_lines(file):
    # read_lines 함수 호출, line_content에는 line이 들어가고,
    # line_data_cnt에는 line의 길이가 들어간다.
        print("[main] 데이터 길이 : {}" .format(line_data_cnt)) # line_data_cnt를 출력
        print("-----" * 2) # "-----"를 두번 출력
    file.close() # 파일 닫기
```

```
[generator] line : (1절)

[main] 데이터 길이 : 5
----------
[generator] line : 동해물과 백두산이 마르고 닳도록

[main] 데이터 길이 : 18
----------
[generator] line : 하느님이 보우하사 우리나라만세

[main] 데이터 길이 : 17
----------
[generator] line : (후렴)무궁화 삼천리 화려강산 대한사람 대한으로 길이보전하세

[main] 데이터 길이 : 34
----------
[generator] line :

[main] 데이터 길이 : 1
----------
[generator] line : (2절)

[main] 데이터 길이 : 5
----------
[generator] line : 남산위에 저 소나무 철갑을 두른듯

[main] 데이터 길이 : 19
----------
[generator] line : 바람서리 불변함은 우리기상 일세

[main] 데이터 길이 : 18
----------
[generator] line : (후렴)무궁화 삼천리 화려강산 대한사람 대한으로 길이보전하세

[main] 데이터 길이 : 34
----------
[generator] line :

[main] 데이터 길이 : 1
----------
[generator] line : (3절)

[main] 데이터 길이 : 5
----------
[generator] line : 가을하늘 공활한데 높고 구름없이

[main] 데이터 길이 : 18
----------
[generator] line : 밝은달은 우리가슴 일편단심일세

[main] 데이터 길이 : 17
----------
[generator] line : (후렴)무궁화 삼천리 화려강산 대한사람 대한으로 길이보전하세

[main] 데이터 길이 : 34
----------
[generator] line :

[main] 데이터 길이 : 1
----------
```

```
[generator] line : (4절)

[main] 데이터 길이 : 5
----------
[generator] line : 이 기상과 이 맘으로 충성을 다하여

[main] 데이터 길이 : 20
----------
[generator] line : 괴로우나 즐거우나 나라사랑하세

[main] 데이터 길이 : 17
----------
[generator] line : (후렴)무궁화 삼천리 화려강산 대한사람 대한으로 길이보전하세
[main] 데이터 길이 : 33
----------
```