파이썬프로그래밍 2분반 ## <11주차 과제> ---

정보보안공학과 ### 202121556 ### 곽지현

2023-05-21

In [7]:
```
# 1. enumerate 함수의 이해를 위해 다음 코드를 실행하여 결과를 확인해보세요.

items = ["item1", "item2", "item3", "item4", "item5"] # 리스트 선언
for i, item in enumerate(items): # 현재 인덱스가 몇 번째인지 확인하는 함수 사용
    print("{}. {}" .format(chr(65+i), item)) # chr(65+i)와 item 값을 출력
```

```
A. item1
B. item2
C. item3
D. item4
E. item5
```

In [10]:
```
# 2. 딕셔너리의 items 함수의 이해를 위해 다음 코드를 실행하여 결과를 확인해보세요.

dict1 = {
    "key1" : ["item1.1", "item1.2", "item1.3", "item1.4", "item1.5"],
    "key2" : ["item2.1", "item2.2", "item2.3", "item2.4", "item2.5"]
} # 딕셔너리 선언

for i, (key, list_value) in enumerate(dict1.items()):
    # for 반복문과 items() 함수 조합해서 사용
    print("{}. {}" .format(chr(65+i), key)) # chr(65+i)와 key 값을 출력

    for j, value in enumerate(list_value):
        # 현재 인덱스가 몇 번째인지 확인하는 함수 사용
        print("Wt{}-{}. {}" .format(chr(65+i), j, value))
        # chr(65+i)와 j, value 값을 출력
```

```
A. key1
        A-0. item1.1
        A-1. item1.2
        A-2. item1.3
        A-3. item1.4
        A-4. item1.5
B. key2
        B-0. item2.1
        B-1. item2.2
        B-2. item2.3
        B-3. item2.4
        B-4. item2.5
```

In [14]:
```
# 3. 리스트 내포(list comprehensions)의 이해를 위해 다음 코드를 실행하여 결과를 확인해보

list_comprehension1 = [x for x in range(10)]
# 리스트 선언, 리스트안에 for문을 사용, for 앞에 최종 결과를 작성
```

```
list_comprehension2 = ["{} (짝수)".format(x) if x % 2 == 0 else "{} (홀수)".format(x)
# 리스트 선언, 리스트안에 for문과 if문 사용, for문 앞에 if문을 넣어 최종 결과를 작성

list_comprehension3 = [print(x, end=" ") for x in range(10)]
# 리스트 선언, 리스트안에 for문을 사용, for 앞에 최종 결과를 작성

print("₩n") # 줄 바꿈
print(list_comprehension1) # (list_comprehension1) 리스트 출력
print(list_comprehension2) # (list_comprehension2) 리스트 출력
print(list_comprehension3) # (list_comprehension3) 리스트 출력
```

```
0 1 2 3 4 5 6 7 8 9

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
['0 (짝수)', '1 (홀수)', '2 (짝수)', '3 (홀수)', '4 (짝수)', '5 (홀수)', '6 (짝수)', '7
(홀수)', '8 (짝수)', '9 (홀수)']
[None, None, None, None, None, None, None, None, None, None]
```

In [15]:
```
# 4. 함수의 매개변수(parameter)의 이해를 위해 다음 코드를 실행하여 결과를 확인해보세요.

def func1(parameter1, parameter2): # func1 함수 생성
    print("함수1번 작동") # "함수1번 작동"을 출력
    print("parameter1 :", parameter1) # 매개변수 parameter1에 10이 들어감
    print("parameter2 :", parameter2) # 매개변수 parameter2에 string이 들어감
    print() # 줄 바꿈

def func2(parameter1): # func2 함수 생성
    print("함수1번 작동") # "함수1번 작동"을 출력
    print("parameter1 :", parameter1)
    # 매개변수 parameter2에 ['list', 'item']이 들어감
    print() # 줄 바꿈

func1(1, "string") # func1 함수 호출
func2(["list", "item"]) # func2 함수 호출
```

```
함수1번 작동
parameter1 : 1
parameter2 : string

함수1번 작동
parameter1 : ['list', 'item']
```

In [1]:
```
# 5. 기본 매개변수의 이해를 위해 다음 코드를 실행하여 결과를 확인해보세요.

def cumulative_sum(end_num, debug = False): # cumulative_sum 함수 생성
    result = 0 # result에 0을 대입

    print("[comulative_sum] (info) Debug print :", debug) # debug 출력
    for x in range(1, end_num+1): # 1 부터 6 까지 반복
        result += x # result에 x를 더하여 대입
        if debug:
            print("[cumulative_sum] (DEBUG) result += {}, result : {}" .format(x, resul
            # x와 result의 값을 출력

    print("[cumulative_sum] {}까지의 누적합 : {}" .format(end_num, result), "₩n")
    # end_num과 resutl의 값을 출력

# 함수 호출
```

```
cumulative_sum(5) # end_num에 5가 들어감
cumulative_sum(5, True) # end_num에 5가 들어가고 debug에 True가 들어감
```

```
[comulative_sum] (info) Debug print : False
[cumulative_sum] 5까지의 누적합 : 15

[comulative_sum] (info) Debug print : True
[cumulative_sum] (DEBUG) result += 1, result : 1
[cumulative_sum] (DEBUG) result += 2, result : 3
[cumulative_sum] (DEBUG) result += 3, result : 6
[cumulative_sum] (DEBUG) result += 4, result : 10
[cumulative_sum] (DEBUG) result += 5, result : 15
[cumulative_sum] 5까지의 누적합 : 15
```

In [3]:
```
# 6. 가변 매개변수의 이해를 위해 다음 코드를 실행하여 결과를 확인해보세요.

def variable_func1(*variable_parameter, parameter1): # variable_func1 함수 생성
    print("[variable_func1] variable_parameter :", variable_parameter)
    # variable_parameter은 리스트처럼 활용
    print("[variable_func1] parameter1 :", parameter1)
    # parameter1에 4가 들어감
    print() # 줄 바꿈

def variable_func2(parameter1, *variable_parameter): # variable_func2 함수 생성
    print("[variable_func2] variable_parameter :", variable_parameter)
    # variable_parameter은 리스트처럼 활용
    print("[variable_func2] parameter1 :", parameter1)
    # parameter1에 1이 들어감
    print() # 줄 바꿈

variable_func1(1, 2, 3, parameter1 = 4) # variable_func1 함수 호출
variable_func2(1, 2, 3, 4) # variable_func2 함수 호출
```

```
[variable_func1] variable_parameter : (1, 2, 3)
[variable_func1] parameter1 : 4

[variable_func2] variable_parameter : (2, 3, 4)
[variable_func2] parameter1 : 1
```

In [8]:
```
# 7. 키워드 인자의 이해를 위해 다음 코드를 실행하여 결과를 확인해보세요.

def keword_argument(parameter1, parameter2, parameter3 = 0):
    # keword_argument 함수 생성
    print("[keword_argument] parameter1 :", parameter1) # parameter1 출력
    print("[keword_argument] parameter2 :", parameter2) # parameter2 출력
    print("[keword_argument] parameter3 :", parameter3) # parameter3 출력
    print() # 줄 바꿈

# 매개변수 이름을 지정해서 입력
keword_argument(1, parameter2 = 3)
# parameter1에 1이 들어가고 parameter2에 3이 들어감
keword_argument(parameter1 = 1, parameter2 = 2, parameter3 = 3)
# parameter1에 1이 들어가고 parameter2에 2가 들어가고 parameter3에는 3이 들어감
keword_argument(parameter3 = 1, parameter2 = 2, parameter1 = 3)
# parameter3에 1이 들어가고 parameter2에 2가 들어가고 parameter1에는 3이 들어감
```

```
[keword_argument] parameter1 : 1
[keword_argument] parameter2 : 3
[keword_argument] parameter3 : 0

[keword_argument] parameter1 : 1
[keword_argument] parameter2 : 2
[keword_argument] parameter3 : 3

[keword_argument] parameter1 : 3
[keword_argument] parameter2 : 2
[keword_argument] parameter3 : 1
```

In [9]:
```python
# 8. 함수 리턴의 이해를 위해 다음 코드를 실행하여 결과를 확인해보세요.

def return_func(return_type): # return_func 함수 생성
    print("[return_func] return_type parameter:", return_type) # return_type을 출력
    if return_type == "int": # return_type이 "int"와 같으면
        return 10 # 10을 리턴
    elif return_type == "string": # return_type이 "string"과 같으면
        return "return_value_string" # "return_value_string"을 리턴
    elif return_type == "tuple": # return_type이 "tuple"과 같으면
        return ("tuple_item", 1) # ('tuple_item', 1)을 리턴
    elif return_type == "list": # return_type이 "list"와 같으면
        return ["list_item", 2] # ['list_item', 2]를 리턴
    elif return_type == "dict": # return_type이 "dict"와 같으면
        return {"key1" : "value1"} # {'key1' : 'value1'}을 리턴
    else:
        return # 리턴 값이 없다.

print("return : {}Wn" .format(return_func("int"))) # "int"를 넣고 함수 호출
print("return : {}Wn" .format(return_func("string"))) # "string"을 넣고 함수 호출
print("return : {}Wn" .format(return_func("tuple"))) # "tuple"을 넣고 함수 호출
print("return : {}Wn" .format(return_func("list"))) # "list"를 넣고 함수 호출
print("return : {}Wn" .format(return_func("dict"))) # "dict"를 넣고 함수 호출
print("return : {}Wn" .format(return_func("another"))) # "another"을 넣고 함수 호출
```

```
[return_func] return_type parameter: int
return : 10

[return_func] return_type parameter: string
return : return_value_string

[return_func] return_type parameter: tuple
return : ('tuple_item', 1)

[return_func] return_type parameter: list
return : ['list_item', 2]

[return_func] return_type parameter: dict
return : {'key1': 'value1'}

[return_func] return_type parameter: another
return : None
```

In [18]:
```python
# 9. 숫자를 받아 2를 더해주는 함수를 구현한 프로그램입니다. 다음과 같은 결과가 나오도록

def plus_2(num): # plus_2 함수 생성
    num = num + 2 # num에 num+2를 대입
```

```
    return num # num을 리턴

num = 4 # num에 4를 대입
print(num) # num 값을 출력
num = plus_2(num) # plus_2 함수 호출해서 num에 대입
print(num) # num 값을 출력
```

```
4
6
```