함수_및_람다(lambda)

- description : 함수 및 람다(lambda) 이해
- author : 도봉산핵주먹
- email : hylee@repia.com
- lastupdate : 2020-06-25

함수 및 람다(lambda) 이해

예제 코드

```
# Section06
# 파이썬 함수식 및 람다(lambda)

# 함수 정의 방법
# def function_name(parameter):
#     code

# 함수 호출
# function_name()
# 함수 선언 위치 중요


# 예제1
print("#==== 일반함수 ====")
print("#=== 기본예제 ===")
def hello(world):
    print("Hello, ", world)


param1 = "Niceman"

hello(param1)

print()

# 예제2
print("#=== 기본 리턴 ===")
def hello_return(world):
    value = "Hello, " + str(world)
    return value


str = hello_return("Niceman")

print(str)
```

```
print()

# 예제3(다중리턴)
print("#=== 다중 리턴 ===")

def func_mul1(x):
    y1 = x * 2
    y2 = x * 4
    y3 = x * 6
    return y1, y2, y3


val1, val2, val3 = func_mul1(3)

print(val1, val2, val3)

print()

# 튜플 리턴
print("#=== 튜플 리턴 ===")
def func_mul2(x):
    y1 = x * 2
    y2 = x * 4
    y3 = x * 6
    return (y1, y2, y3)


tup = func_mul2(4)

print(type(tup), tup, list(tup))

print()

# 리스트 리턴
print("#=== 리스트 리턴 ===")
def func_mul2(x):
    y1 = x * 2
    y2 = x * 4
    y3 = x * 6
    return [y1, y2, y3]


lis = func_mul2(6)

print(type(lis), lis, set(lis))

print()
```

```
# 딕셔너리 리턴
print("#=== 딕셔너리 리턴 ===")
def func_mul3(x):
    y1 = x * 2
    y2 = x * 4
    y3 = x * 6
    return {'ret1': y1, 'ret2': y2, 'ret3': y3}

# 리스트 리턴
# return [y1, y2, y3]

# 튜플 리턴
# return (y1, y2, y3)


dic = func_mul3(8)
print(type(dic), dic, dic.get('ret3'), dic.items(), dic.keys(),
dic.values())


print()

# 예제4
# *args, **kwargs 이해
print("#=== *args, **kwargs 이해 ===")

# *args
# 매개변수명 자유롭게 변경 가능
print("#== *args 이해 ==")
def args_func(*args):
    for i, v in enumerate(args): # enumerate -> index를 만들어서 순회한다.
        print('{}'.format(i), v, end=' ')


args_func('Kim')
args_func('Kim', 'Park')
args_func('Kim', 'Park', 'Lee')

print('\n')


# kwargs
print("#== **kwargs 이해 ==")
def kwargs_func(**kwargs):  # 매개변수명 자유롭게 변경 가능
    for v in kwargs.keys():
        print('{}'.format(v), kwargs[v], end=' ')


kwargs_func(name1='Kim')
kwargs_func(name1='Kim', name2='Park')
kwargs_func(name1='Kim', name2='Park', name3='Lee')
```

```
print('\n')


# 전체 혼합
print("#== 혼합 이해 ==")
def example(arg_1, arg_2, *args, **kwargs):
    print(arg_1, arg_2, args, kwargs)


example(10, 20)
example(10, 20, 'park', 'kim', 'lee')
example(10, 20, 'park', 'kim', 'lee', age1=33, age2=34, age3=44)

print()

# 예제5
# 중첩함수 (클로저)
print("#=== 중첩함수 이해 (클로저) ===")
def nested_func(num):
    def func_in_func(num):
        print(num)

    print("In func")
    func_in_func(num + 100)


nested_func(1)

print()

# 실행불가
# func_in_func(1)


# 예제6
# Hint
print("#=== Hint 이해 ===")
def tot_length1(word: str, num: int) -> int:
    return len(word) * num

# word: str, num: int
#     : Hint    : Hint

print('hint exam1 : ', tot_length1("i love you", 10))


def tot_length2(word: str, num: int) -> None:
    print('hint exam2 : ', len(word) * num)
```

```
tot_length2("niceman", 10)

print('\n\n')


print("#==== 람다식 함수 ====")

# 람다식 예제
# 메모리 절약, 가독성 향상, 코드 간결
# 함수는 객체 생성 -> 리소스(메모리) 할당
# 람다는 즉시 실행 함수(Heap 초기화) -> 메모리 초기화

# 예제7
# def mul_10(num):
#     return num * 10

# def mul_10_one(num): return num * 10
#
# lambda x: x * 10

# 일반적 함수 -> 변수 할당
print("#=== 일반적 함수 -> 변수 할당 ===")
def mul_10(num):
    return num * 10


mul_func = mul_10

print(mul_func(5))
print(mul_func(6))

print()
# 람다 함수 -> 할당

print("#=== 람다 함수 -> 할당 ===")
lambda_mul_func = lambda x: x * 10


def func_final(x, y, func):
    print(x * y * func(10))


func_final(10, 10, lambda_mul_func)
```

실행 콘솔

```
#==== 일반함수 ====
#=== 기본예제 ===
Hello,  Niceman

#=== 기본 리턴 ===
```

```
Hello, Niceman

#=== 다중 리턴 ===
6 12 18

#=== 튜플 리턴 ===
<class 'tuple'> (8, 16, 24) [8, 16, 24]

#=== 리스트 리턴 ===
<class 'list'> [12, 24, 36] {24, 12, 36}

#=== 딕셔너리 리턴 ===
<class 'dict'> {'ret1': 16, 'ret2': 32, 'ret3': 48} 48 dict_items([('ret1',
16), ('ret2', 32), ('ret3', 48)]) dict_keys(['ret1', 'ret2', 'ret3'])
dict_values([16, 32, 48])

#=== *args, **kwargs 이해 ===
#== *args 이해 ==
0 Kim 0 Kim 1 Park 0 Kim 1 Park 2 Lee

#== **kwargs 이해 ==
name1 Kim name1 Kim name2 Park name1 Kim name2 Park name3 Lee

#== 혼합 이해 ==
10 20 () {}
10 20 ('park', 'kim', 'lee') {}
10 20 ('park', 'kim', 'lee') {'age1': 33, 'age2': 34, 'age3': 44}

#=== 중첩함수 이해 (클로저) ===
In func
101

#=== Hint 이해 ===
hint exam1 :  100
hint exam2 :  70



#==== 람다식 함수 ====
#=== 일반적 함수 -> 변수 할당 ===
50
60

#=== 람다 함수 -> 할당 ===
10000
```

Tip

도봉산핵주먹, python, 함수식, 람다, lambda

From:
http://rwiki.repia.com/ - **문제를 잘 정의하는 것은 문제를 절반 해결한 것이다. - 2023.12**

Permanent link:
http://rwiki.repia.com/doku.php?id=wiki:ai:python:%ED%95%A8%EC%88%98_%EB%B0%8F_%EB%9E%8C%EB%8B%A4&rev=1593063976

Last update: **2022/03/10 19:52**