2주차: 함수, 논리 연산, 문자열 다루기

지난 주 도전과제 해답

In [6]:

```
coffee_menu = [
    {
        "name": "아메리카노",
        "price": 4000
    },
    {
        "name": "카페라떼",
        "price": 5000
    },
    {
        "name": "카페모카",
        "price": 6000
    }
]
```

In [10]:

```
# for 문을 두 번 사용하는 방법.
def calculate_total_price(coffees):
    total_price = 0

    for menu in coffee_menu:
        for target in coffees:
            if menu['name'] == target:
                total_price = total_price + menu['price']
                # 이 부분은 += 단항 연산자로 고쳐쓸 수 있습니다.

    return total_price
```

In [9]:

```
# Python 좀 하는 사람들이 사용하는 방법.
# if 문에도 'in'을 사용할 수 있습니다.
def calculate_total_price_using_in(coffees):
    total_price = 0

    for menu in coffee_menu:
        if menu['name'] in coffees:
            total_price += menu['price']

    return total_price

# 호출
# 아메리카노가 4000원, 카페모카가 6000원이니 10000이 찍혀야 합니다.
print "2중 for문 방법: ", calculate_total_price(["아메리카노", "카페모카"])
print "if에서 in을 사용하는 방법: ", \
        calculate_total_price_using_in(["아메리카노", "카페모카"])
```

```
2중 for문 방법:  10000
if에서 in을 사용하는 방법:  10000
```

함수와 return

In [2]:

```
# 함수는 말그대로 '매개변수(x)'를 받아서 특정 작업을 거친 '출력값(y)'을 내놓는 것입니다.
# 따라서 특정 출력값을 내놓기 위해 사용하는 키워드가 바로 'return' 입니다.

def sum_func(a, b):
    return a+b

result = sum_func(sum_func(1, 2), sum_func(3, sum_func(4, 5)))
print result

# 형태를 막론하고 return 할 수 있습니다.
def calc(a, b):
    return [a+b, a-b]

print calc(1, 2)[1]
```

```
15
-1
```

함수도 하나의 자료형이다

In [2]:

```
# 함수도 일반 변수로 취급할 수 있답니다.

some_value = 1

def sum_func(a, b):
    return a+b

def minus_func(a, b):
    return a-b


print some_value
print sum_func  # 실제 해당 함수가 가르키고 있는 메모리의 주소 값을 보여줍니다.
print type(some_value)  # type() 내장 함수를 이용하면,
                        # 해당 변수의 형태를 볼 수 있습니다.
print type(sum_func)

# 따라서 함수를 매개변수로 넣을 수도 있습니다.
def calc(input_1, input_2, what_to_do):
    return what_to_do(input_1, input_2)

print calc(10, 20, sum_func)
print calc(10, 20, minus_func)
```

```
1
<function sum_func at 0x10f799938>
<type 'int'>
<type 'function'>
30
-10
```

In [3]:

```
# 함수를 리턴하는 것도 가능합니다.
# 함수 안에 함수를 선언하는 것도 역시 가능합니다.
def get_calc_module(module_name):

    def inner_sum_func(a, b):
        return a+b

    def inner_minus_func(a, b):
        return a-b

    if module_name == 'sum':
        return inner_sum_func
    elif module_name == 'minus':
        return inner_minus_func

# 리턴 받은 함수를 변수에 할당할 수 있습니다.
guess_what = get_calc_module('sum')
print type(guess_what)

# 이렇게 실행도 가능합니다.
print guess_what(100, 200)

# 좀 더 머리를 쓰면, 따로 변수에 할당하지 않고 한 줄로 쓸 수도 있겠죠.
print get_calc_module('minus')(500, 300)
```

```
<type 'function'>
300
200
```

반복문을 대신하는 함수의 재귀 호출 (Recursive)

In [4]:

```
# 함수는 자기 스스로를 부를수도 있답니다.

# for 문을 이용해 팩토리얼을 구하는 함수를 만들어봅시다.
def factorial_for(n):
    result = 1
    for i in range(1, n+1):
        result = result * i
    return result

print factorial_for(5)

# 재귀호출을 이용해 함수를 만들어봅시다.
def factorial(n):
    if(n == 1):
        return 1
    else:
        return n * factorial(n-1)

print factorial(5)

# 개념을 이해하기 어렵기 때문에, 직접 해보는게 좋습니다.
```

```
120
120
```

재귀호출을 이용한 알고리즘 설계 (문제 해결)

In [5]:

```
# 직접 해봅시다: 피보나치 수열 구하기

# 피보나치 수열: 0,1,1,2,3,5,8,13,21,34,55,89, ...
# 수학적 정의: Fn = Fn-1 + Fn-2 (단, F0 = 0 and F1 = 1)

# n번째는 얼마일까? (0부터 시작함을 유념)

# 예상 결과
# print fib(1)  # 1
# print fib(6)  # 8

# 샘플
def fib(n):
    # ...
    return
```

args, kwargs: 가변길이 매개변수

In [6]:

```
# 우리는 항상 함수에서 미리 정의된 매개 변수만을 사용했습니다.

def var_sum(*args):  # 가변기리 매개변수를 이용하기위해 *를 붙여줍니다. (args)

    sum = 0
    for i in args:
        sum += i

    return sum

print var_sum(10, 20, 1, 5, 7)  # 무한히 쓸 수 있습니다.
print var_sum(1, 2)


# 키워드 가변길이 매개변수도 사용 가능합니다. (kwargs)

def fruits(**kwargs):

    for key, val in kwargs.items():  # 딕셔너리를 for 문으로 순회할 때에는
                                     # .items() 라는 내장 변수를 사용합니다.
        print key, val

fruits(grape=3000, lemon=5000, apple=1000)
```

```
43
3
lemon 5000
grape 3000
apple 1000
```

List comprehension

In [7]:

```
# 저번 주 도전과제 중 파이썬 초 고수들이 쓰는 방법: List comprehension
# 위의 for 문과 비교해보세요.
print sum([menu['price'] if menu['name'] in ["아메리카노", "카페모카"]
           \else 0 for menu in coffee_menu])

# 파이썬의 강점 중 하나인 List comprehension:
# 여러 줄의 for 문과 if 문을 한 줄로 줄여줍니다.
# 어려운 로직에서는 가독성이 조금 떨어지는 단점이 있지만(안티패턴),
# 단순한 로직에서는 대체로 선호하는 편

# result = []
# for value in list:
#     result.append(value)
# ==> result = [value for value in list] 형식임.


# 직접 해봅시다: 주어진 숫자 리스트를 무조껀 2배로 곱해 다시 저장하기.
input = [1, 3, 5, 7, 9, 3, 4, 5]
output = []  # 여기에 구현
print output

# if 문이 들어가는 경우
# ==> [참일때값 if 조건식 else 거짓일때값 for value in list] 형식
# 직접 해봅시다: 위의 문제에서, 원래 값이 홀수 일 때만 2배로 곱해 저장. 짝수면 그냥 바로 저장.
# 힌트: 홀수인지 알아내는 법은 2로 나눠서 나머지가 1이면 되겠지요.
# 나머지 연산자(%)를 사용하면 됩니다. e.g. 9 % 2 ==> 1
```

```
10000
[]
```

파이참에서 디버깅 하는 방법

프로그래밍에서 버그란?

- 프로그램을 실행하는 과정에서 발생하는 '오류'
- 실제 1945년에 MarkII 컴퓨터의 회로에 나방이 들어가 합선을 일으켜 비정상적으로 동작한 것이 인류 최초의 컴퓨터 버그. (그래서 오류가 버그라고 불리게 됨.)
- 오류의 종류
  - Syntax Error: 말 그대로 프로그래밍 문법이 틀려서 나는 문제. Python 인터프리터에서 문법 오류가 발생했다고 알려줌. *인터프리터: Python 소스 코드를 실행시 분석해 실제 컴퓨터가 알아들을 수 있는 2진수 코드로 바꿔주는 장치
  - Semantic Error: 문법은 잘 맞았지만, 개발자가 의도하는대로 작동하지 않은 경우. 대부분의 버그라고 생각하면 쉬움.

In [8]:

```
# Semantic Error의 예
result = 0
for i in range(1, 11):
    result *= i
print result    # 개발자는 1부터 10까지의 곱을 구하려고 의도하였으나,
                # 초기값이 0인 관계로 정상 작동하지 않음.
```

0

디버깅이란

- 프로그램 개발에서 버그 발생은 필수불가결함. 즉, '작성하자 마자 바로 돌아가는 프로그램은 없다'라는 것. 가령 위에서 진행했던 재귀 호출 알고리즘 문제같은 아주 간단한 프로그램도 처음 작성한 버전은 제대로 작성했을 가능성이 매우 낮음. 여러번 고쳐가며 돌려봐야 제대로 된 해답을 찾을 수 있게 된다는 것.
- 따라서 개발자에게 버그를 찾아내고 수정하는 '디버깅' 능력은 매우 중요하게 여겨짐.

실제 파이참에서 디버깅하는 방법

1. 순차적으로 실행되는 소스코드 중, 중간에 멈추었으면 하는 부분에서 '중단점(Break point)'을 만든다.
2. 'Run' 대신에 'Debug'를 눌러 프로그램을 실행.
3. 실행 도중 (1)에서 지정한 중단점에서 프로그램이 중지하면, 하단 Debugger의 Watches를 통해 변수들의 값을 확인 할 수 있음. ('+' 버튼을 눌러 감시하고자 하는 변수를 추가할 수 있음.)
4. 중단점 이후에 소스 코드를 한 줄씩 실행하고 싶은 경우에는 F8을 누르고,
5. 다음 중단점, 혹은 다음 중단점이 없는 상태에서 프로그램 끝까지 실행하고 싶다면 Resume을 누르면 됨.

- 실제 파이참 환경에서 같이 해봅시다.

참과 거짓, 논리 연산

In [9]:

```
# 참과 거짓

print 1 == 1
print 1 == 2

# if 문 안의 수식은 최종적으로 True 혹은 False로 바뀌어 평가되는 것이랍니다.
if True:
    print "참이다!"

if False:
    pass
else:
    print "거짓이다!"
```

```
True
False
참이다!
거짓이다!
```

In [10]:

```
# 'not'을 붙여 강제로 뒤집을 수도 있습니다.

print not True
print not False

if not False:
    print "거짓이 아닙니다."
```

```
False
True
거짓이 아닙니다.
```

In [11]:

```
# Python에서는 and(그리고), or(또는) 논리 연산이 가능합니다.
# and는 말 그대로 '둘 다 참'이어야 최종적으로 '참'이 되는 연산
# or는 말 그대로 '둘 중 하나만 참'이어도 최종적으로 '참'이 되는 연산

print True and True
print True and False
print False and True
print False and False

print True or True
print True or False
print False or True
print False or False
```

```
True
False
False
False
True
True
True
False
```

In [12]:

```
# 2중 if문은 'and'연산으로 대체할 수 있습니다.

# 80~90점은 B등급.
score = 85

if score >= 80:
    if score < 90:
        print "B등급입니다."

# and 연산으로 대체

if score >= 80 and score < 90:
    print "B등급입니다."
```

```
B등급입니다.
B등급입니다.
```

In [13]:

```
# 연습해봅시다: '오늘 여행을 갈 수 있는지 알아보는' 함수를 만들어봅시다.

# is_weekday: 주중인지 아닌지
# is_vacation: 휴가 기간인지 아닌지
def can_travel(is_weekday, is_vacation):
    # ...
    return

# 조건1. 주중과 주말 중, 우리는 주말에만 여행을 갈 수 있습니다.
# 조건2. 주중이더라도 휴가 기간이면 우리는 여행을 갈 수 있습니다.

# 예상 결과
# can_travel(False, True) -> True
# can_travel(True, False) -> False
# can_travel(False, True) -> True
```

연산자와 그 우선순위

Python에는 다양한 연산자들이 존재하며, 이 연산자들 사이에도 누가 먼저 계산이 될지 연산자 우선순위(Operators Precedence)를 가지고 있습니다.

우선순위	연산자	설명
1	**	지수
2	*, /, %, //	곱하기, 나누기, 나머지, 몫
3	+, -	덧셈, 뺄셈
4	<=, <, >, >=	비교 연산자
5	==, !=	평등 연산자
6	=, %=, /=, //=, -=, +=, =, *=	할당 연산자
7	in, not in	멤버 연산자
8	not, or, and	논리 연산자

In [14]:

```
print 1 + 2 ** 3  # 2**3 이 먼저 시행됩니다.
a = 1 + 3  # 우리는 당연하게 생각했지만, = (할당)연산이 + 연산보다 뒤에 이루어진 것입니다.
if a == 4 and a >= 1*2:
    print "가장 먼저 우항의 곱셈 연산이 이루어지고, \
           좌항의 평등 연산이 이루어진 이후에 가운데 논리 연산이 이루어졌습니다."
```

```
9
가장 먼저 우항의 곱셈 연산이 이루어지고, 좌항의 평등 연산이 이루어진 이후에 가운데 논
리 연산이 이루어졌습니다.
```

문자열을 만드는 네 가지 방법

In [15]:

```
# 문자열은 '(홑따옴표), "(쌍따옴표), ''' , """를 이용해 선언할 수 있습니다.

shakespeare = '"사느냐 죽느냐" 그것이 문제로다!'
print shakespeare

shakespeare = "'사느냐 죽느냐' 그것이 문제로다!"
print shakespeare

# 여러줄로 선언하기
shakespeare = '''
사느냐 죽느냐
그것이 문제로다!
'''
print shakespeare

shakespeare = """
사느냐 죽느냐
그것이 문제로다!
"""
print shakespeare
```

```
"사느냐 죽느냐" 그것이 문제로다!
'사느냐 죽느냐' 그것이 문제로다!

사느냐 죽느냐
그것이 문제로다!


사느냐 죽느냐
그것이 문제로다!
```

문자열은 리스트와 형제다!

In [16]:

```
# 문자열은 리스트와 형제라고합니다.

favorite_pizza = "Papa Jones"
print favorite_pizza[0]

# 리스트이기 때문에, 이렇게 for 문 안에 넣을수도 있죠.
for c in favorite_pizza:
    print c

# 리스트는 이렇게 뒤에서부터 셀 수도, 중간값을 가져올 수도 있습니다.
print favorite_pizza[-1]
print favorite_pizza[0:3]  # [시작:끝]
print favorite_pizza[5:10]

# 총 갯수는 이렇게 센답니다..
print len(favorite_pizza)  # 리스트와 문자열은 형제니까,
                           # 리스트의 갯수도 이렇게 셀 수 있겠죠!
```

```
P
P
a
p
a

J
o
n
e
s
s
Pap
Jones
10
```

In [17]:

```
# 연습해봅시다!

# 주어진 문자열에서, 알파벳 'e'가 얼마나 들어가 있는지 세는 함수를 만들어보세요!

def count_alphabet():
    count = 0
    Lyrics = "Yesterday, all my troubles seem so far away"
    # ...
    return count
```

In [18]:

```
# 연습해봅시다2!

# 어떠한 문자열이 주어지던지, 그걸 두번씩 쓰는 함수를 만들어보세요!

def double_alphabet(input):
    result = ""
    # ...
    return result

# 예상 결과
# double_alphabet("hello!") -> "hheellllloo!!"
```

문자열 포매팅(formatting) 하기

In [19]:

```
# 문자열 중간에 다른 내용(숫자, 문자열..)을 넣고 싶다면?

apple_mania = "나는 {0}개의 사과를 먹었습니다."
print apple_mania.format(1)
print apple_mania.format(10)

fruits_mania = "나는 {0}개의 사과와, {1}개의 수박과, {2}개의 딸기를 먹었습니다. \
                {3}도 먹었습니다."
print fruits_mania.format(2, 1, 5, "귤")
```

```
나는 1개의 사과를 먹었습니다.
나는 10개의 사과를 먹었습니다.
나는 2개의 사과와, 1개의 수박과, 5개의 딸기를 먹었습니다. 귤도 먹었습니다.
```

In [20]:

```
# 연습해봅시다!

# 1마리의 돼지가 꿀하고 웁니다.
# 2마리의 돼지가 꿀꿀하고 웁니다.
# 3마리의 돼지가 꿀꿀꿀하고 웁니다.
# 4마리의 돼지가 꿀꿀꿀꿀하고 웁니다.
# 5마리의 돼지가 꿀꿀꿀꿀꿀하고 웁니다.
# 6마리의 돼지가 꿀꿀꿀꿀꿀꿀하고 웁니다.
# 7마리의 돼지가 꿀꿀꿀꿀꿀꿀꿀하고 웁니다.
# 8마리의 돼지가 꿀꿀꿀꿀꿀꿀꿀꿀하고 웁니다.
# 9마리의 돼지가 꿀꿀꿀꿀꿀꿀꿀꿀꿀하고 웁니다.
# 10마리의 돼지가 꿀꿀꿀꿀꿀꿀꿀꿀꿀꿀하고 웁니다.

# format()과 for문을 이용해 위의 결과를 만들어보세요.
```

필수로 알아야 하는 문자열 관련 함수들

In [21]:

```
a = "hello"

# 소문자 -> 대문자
print a.upper()

# 대문자 -> 소문자
print "HEY!".lower()

# 문자 갯수 세기
print a.count("l")

# 문자 위치 알기
print a.find("e")  # 0부터 시작, 없으면 -1 리턴

# 문자열 삽입(합치기)
print "-".join("hello")

# 좌, 우 공백 지우기
print "   Hey!!      ".strip()

# 문자열 바꾸기
print "sugar is so sweet.".replace("sugar", "salt").\
        replace("sweet", "salty")

# 문자열 나누기
gump = "Life is like a box of chocolate"
splited_gump = gump.split()  # 공백을 기준으로 나눕니다.
print splited_gump
dashed_gump = "-".join(splited_gump)
print dashed_gump
re_splited_gump = dashed_gump.split("-")
print re_splited_gump
```

```
HELLO
hey!
2
1
h-e-l-l-o
Hey!!
salt is so salty.
['Life', 'is', 'like', 'a', 'box', 'of', 'chocolate']
Life-is-like-a-box-of-chocolate
['Life', 'is', 'like', 'a', 'box', 'of', 'chocolate']
```

입출력 (I/O)

모든 프로그램은 결국 사용자의 입력을 받아 -> 처리 하고 -> 출력하는 과정을 거치는 것이라고 해도 과언이 아닐 것입니다. 이를 프로그래밍에서는 I/O (Input/Ouput) 이라고 부릅니다. 한국어로는 입출력이라고 하지요.

Input

컴퓨터(프로그램)가 사용자의 입력을 받는 수단은 무엇이 있을까요?

1. 키보드를 통한 입력 (stdin, Standard Input)
2. 파일 읽기
3. 기타 등등 (스캐너, 마이크..)

Output

컴퓨터(프로그램)가 사용자에게 출력을 해주는 수단은 무엇이 있을까요?

1. 모니터로 출력 (stdout, Standard Output)
2. 파일 쓰기
3. 기타 등등 (프린터, 스피커..)

In [49]:

```
# 먼저 키보드 입력부터 받아볼까요?
user_input = raw_input("무엇이든 입력해주세요.")
print "당신은 {0}라고 입력하셨습니다.".format(user_input)

# 입력을 받을 수 있게 되었기 때문에, 사용자 Interactive한 프로그램을 만들 수 있게 되었습니다.
import random

def number_game():

    life = 3
    answer = random.randrange(1,11)  # 1~10의 랜덤값을 생성시킵니다.

    while life > 0:
    # while은 처음 보시나요? 간략화된 for문이라고 생각하시면 됩니다.
    # 안의 조건이 'True'인 경우에 계속해서 반복하게 되지요.
        user_input = raw_input("1~10 숫자를 맞춰보세요! \
        (남은 목숨: {0})".format(life))
        int_user_input = int(user_input)
        if int_user_input == answer:
            print "{0}! 정답입니다! 게임을 종료합니다.".format(int_user_input)
            return  # 강제로 함수를 종료시킵니다.
        elif int_user_input < answer:
            life -= 1
            print "{0}보다 큽니다!".format(int_user_input)
        elif int_user_input > answer:
            life -= 1
            print "{0}보다 작습니다!".format(int_user_input)

    print "정답은 {0} 이었습니다. 아쉽군요.".format(answer)
    return

number_game()
```

```
무엇이든 입력해주세요.안녕
당신은 안녕라고 입력하셨습니다.
1~10 숫자를 맞춰보세요! (남은 목숨: 3)5
5보다 작습니다!
1~10 숫자를 맞춰보세요! (남은 목숨: 2)3
3보다 작습니다!
1~10 숫자를 맞춰보세요! (남은 목숨: 1)2
2! 정답입니다! 게임을 종료합니다.
```

In [47]:

```
# 파일 I/O를 해봅시다.

f = open("/Users/LyuGGang/Desktop/lecture/temporary.txt", "w")
# 윈도우에서는 "C:\\temproray.txt"로 해보세요.
f.close()

# 파일이 생성 되었을 것입니다.
# 뒤에 쓰인 "파일 열기 모드"에 따라서 다양한 방법으로 열게됩니다.
```

열기 모드	설명
"r"	"읽기 전용 모드"
"w"	"쓰기 전용 모드, 파일이 있으면 덮어 쓰고, 없으면 새로 씀."
"a"	"추가 전용 모드, 파일이 있으면 덧붙여 쓰고, 없으면 새로 씀."

In [12]:

```
# 또한 파일을 모두 사용한 이후에는 close()를 이용해 꼭 파일을 닫아주어야 합니다.
# 그렇지 않으면 외부에서 사용할 수 없음!

# 뭔가 써봅시다.
f = open("/Users/LyuGGang/Desktop/lecture/temporary.txt", "w")
for i in range(1, 11):
    data = "여기는 {0}번 째 줄입니다.\n".format(i)
    # \n은 한 줄 띄우라는(개행) 의미입니다.
    f.write(data)
f.close()
```

In [13]:

```
# 파일을 직접 열어 한 번 확인해봅시다.

# 쓴 파일을 한 번 읽어볼까요?
# 파일은 한 줄 씩 읽는게 기본입니다.
f = open("/Users/LyuGGang/Desktop/lecture/temporary.txt", "r")
while True:  # 무한 루프를 만듭니다.
    line = f.readline()
    if not line:  # 더 이상 내용이 없으면. ==> 끝까지 다 읽었으면
        break  # while문을 빠져 나갑니다. break는 처음 들어보셨겠지만,
               # while이나 for 같은 반복문을 임의로 빠져나갈 때 사용한답니다.
    print line
f.close()
```

```
여기는 1번 째 줄입니다.

여기는 2번 째 줄입니다.

여기는 3번 째 줄입니다.

여기는 4번 째 줄입니다.

여기는 5번 째 줄입니다.

여기는 6번 째 줄입니다.

여기는 7번 째 줄입니다.

여기는 8번 째 줄입니다.

여기는 9번 째 줄입니다.

여기는 10번 째 줄입니다.
```

In [15]:

```
# 하지만 여러 줄을 한 번에 읽을 수도 있죠.
f = open("/Users/LyuGGang/Desktop/lecture/temporary.txt", "r")
lines = f.readlines()  # line's' 인게 다릅니다.
                       # 모든 줄을 읽어 한 번에 리스트로 만들어줍니다.
for line in lines:
    print line
f.close()
```

```
여기는 1번 째 줄입니다.

여기는 2번 째 줄입니다.

여기는 3번 째 줄입니다.

여기는 4번 째 줄입니다.

여기는 5번 째 줄입니다.

여기는 6번 째 줄입니다.

여기는 7번 째 줄입니다.

여기는 8번 째 줄입니다.

여기는 9번 째 줄입니다.

여기는 10번 째 줄입니다.
```

In [16]:

```
# 사실 가장 쉽게 읽을 수도 있습니다.
f = open("/Users/LyuGGang/Desktop/lecture/temporary.txt", "r")
all_data = f.read()
print all_data
f.close()

# 그럼에도 불구하고 한 줄 씩 읽는 이유는,
# 개발자가 라인 by 라인으로 문자열을 처리하기 쉽게 하기 위함이지요.
# 가령 줄 별로 "이름", "나이", "직업" 이 써있는 파일이라면,
# 전체를 다 읽으면 해당 내용을 Parsing 하기 힘들기 때문입니다.
```

```
여기는 1번 째 줄입니다.
여기는 2번 째 줄입니다.
여기는 3번 째 줄입니다.
여기는 4번 째 줄입니다.
여기는 5번 째 줄입니다.
여기는 6번 째 줄입니다.
여기는 7번 째 줄입니다.
여기는 8번 째 줄입니다.
여기는 9번 째 줄입니다.
여기는 10번 째 줄입니다.
```

오늘의 파이널 과제

In [57]:

```
# 파일을 열고 전체 소설에서 가장 많이 사용된 단어가 무엇인지,
# 몇 번 사용되었는지 TOP 5를 딕셔너리 형태로 구한다.
# 단어의 기준: 띄어쓰기

# 샘플 파일
# 위대한 개츠비 (영문 소설)
# http://bit.ly/1PWzLUm
# 파일 -> 다른 이름으로 저장..

# 템플릿
def get_top5_used_words(file_path):
    result = {}
    f = open(file_path)
    # ...
    f.close()
    return result


# 예상 결과
print get_top5_used_words("./the_great_gatsby.txt")
# ==> {"hello": 1234, "hi": 252, "bye": 122, "yo": 53, "good": 10}
```

```
{}
```