

3주차: 파일 입출력, 클래스, 외부 모듈(pip) 사용

입출력 (I/O)

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

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 [1]:

```
# 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 # 곧 배울 '외부 모듈(패키지)' 사용입니다.
```

무엇이든 입력해주세요.안녕
당신은 안녕라고 입력하셨습니다.

In [3]:

```
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()
```

1~10 숫자를 맞춰보세요! (남은 목숨: 3)5

5보다 큽니다!

1~10 숫자를 맞춰보세요! (남은 목숨: 2)8

8! 정답입니다! 게임을 종료합니다.

In [2]:

```
# 파일 I/O를 해봅시다.

f = open("/Users/LyuGGang/Desktop/lecture/temporary.txt", "w")
# 윈도우에서는 "C:\\temporay.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 [3]:

```
# 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 [4]:

```
# 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 사용합니다.  
    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 [5]:

```
# 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.

In [6]:

```
# 한 번 해봅시다  
  
# 파일을 열고 전체 소설에서 사용된 모든 단어와 각 단어가 몇 번 씩  
# 사용되었는지를 딕셔너리 형태로 구한다.  
# 단어의 기준: 띄어쓰기  
  
# 샘플 파일  
# 위대한 개츠비 (영문 소설)  
# http://bit.ly/1PWzLUm  
# 파일 -> 다른 이름으로 저장..  
  
# 템플릿  
def get_words_count(file_path):  
    result = {}  
    f = open(file_path)  
    # ...  
    f.close()  
    return result  
  
# 예상 결과  
print get_words_count("./the_great_gatsby.txt")  
# ==> {"hello": 1234, "hi": 252, "bye": 122, "yo": 53, "good": 10 ...}  
{}
```

Python 내장 함수 (Built-in Functions)

여태까지 우리가 사용했던 `print()`, `del()`, `type()`, `len()`, `sum()` 등을 'Python 내장 함수'라고 한답니다. 유용한 Python 내장 함수들이 어떤 것들이 있는지 알아봅시다.

In [4]:

```
# 절대값 구하기
a = -3
print abs(a)

# abs()가 없었다면..
if a < 0:
    a *= -1
print a

#####
# 리스트의 모든 것이 참인지 구하기 (리스트 AND)
a = [1, 2, 3, True, 'a']
b = [1, 2, 3, False, 'a']
c = [False, None, {}] # False, None(空), 빈 딕셔너리

print all(a)
print all(b)
print all(c)

#####
# 리스트 중 하나라도 참이 있는지 구하기 (리스트 OR)
print any(a)
print any(b)
print any(c)
```

```
3
3
True
False
False
True
True
False
```

In [6]:

```
#####
# 나누기의 몫과 잔을 함께 구하기
print divmod(10, 3) # 결과값인 (,) 형태는 '튜플'이라고 불리움.
# 리스트와 동일하지만 한 번 선언되면 값을 바꾸거나 순서를 바꿀 수 없는 '상수' 타입.

# divmod()가 없었다면..
result = (10 / 3, 10 % 3)
print result

#####
# 딕셔너리 for 문에서 그 순서도 같이 알고 싶다면..
fruits = {
    'grape': 10000,
    'apple': 2000,
    'lemon': 3000
}

for i, key in enumerate(fruits):
    print i, key, fruits[key] # 순서, 키, 값을 동시에 출력함

#####
# 숫자<->문자열 치환하기
a = "3"
b = 4
# print a + b # 문자열과 숫자를 더하였기에 당연히 오류가 발생함
print int(a) + b
print a + str(b)
```

```
(3, 1)
(3, 1)
0 lemon 3000
1 grape 10000
2 apple 2000
7
34
```

In [7]:

```
#####
# 문자열이나 리스트 길이 재기
a = [1, 2, 3, 4, 5]
favorite_fruit = 'strawberry'
print len(a)
print len(favorite_fruit)

#####
# 다 리스트로 만들어버리기.
print list(fruits) # 딕셔너리의 key만 리스트화 함.
print list(favorite_fruit)

#####
# 리스트를 돌면서 뭔가를 해보자.
a = [1, 2, 3, 4, 5]

def make_double(x):
    return x*2

print map(make_double, a)
print [x * 2 for x in a] # List comprehension으로 비슷.
```

```
#####
# 리스트 중 최대/최소값은?
```

```
a = [1, 5, 2, 3, 4, 7, 2]
print max(a)
print min(a)
```

```
5
10
['lemon', 'grape', 'apple']
['s', 't', 'r', 'a', 'w', 'b', 'e', 'r', 'r', 'y']
[2, 4, 6, 8, 10]
[2, 4, 6, 8, 10]
7
1
```

In [8]:

```
#####
# 제곱값 구하기
print pow(2, 3)
print 2**3

#####
# 범위 구하기
# for문에서 사용하던 range는 사실..
print range(1, 11) # 이렇게 원래 리스트로 리턴되던 것.

# 결국 for .. in range()는 아래와 같음.
for i in range(1, 4):
    print i

for i in [1, 2, 3]:
    print i

#####
# 리스트 정렬(소팅) 하기
a = [1, 5, 2, 3, 5, 7, 8, 1, 3]
asc_sorted_a = sorted(a)
desc_sorted_a = sorted(a, reverse=True)
print asc_sorted_a
print desc_sorted_a

#####
# 변수의 타입(자료형) 알기
a = 1
b = [1, 2, 3]
c = "hi"
def d():
    print "hello!"

print type(a)
print type(b)
print type(c)
print type(d)
```

```
8
8
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
1
2
3
1
2
3
[1, 1, 2, 3, 3, 5, 5, 7, 8]
[8, 7, 5, 5, 3, 3, 2, 1, 1]
<type 'int'>
<type 'list'>
<type 'str'>
<type 'function'>
```

클래스(Class)와 객체지향 프로그래밍

- 우리는 여태까지 명령(프로그램)을 어떻게 논리적으로 작성(프로그래밍)하는지에 대한 공부를 해왔습니다. 그러나 이와 비슷하게 중요한 것은, 바로 컴퓨터 내부에 어떠한 정보 혹은 자료를 어떻게 취급할지에 대한 문제입니다.
- 그래서 단순히 프로그래밍 문법 공부가 아닌, 약간의 '방법론'적인 이야기를 해야합니다.
- '객체'란 무엇일까요? 말 그대로 어떠한 '대상'을 의미하는 것이지요.
- 예를 들어봅시다. 가령 우리가 '라디오'를 만든다고 생각해봅시다. 라디오는 크게 다음과 같은 기능들로 이루어져있습니다
 1. 주파수 수신부
 2. 수신한 주파수를 가청(사람이 들을 수 있는) 소리로 바꾸는 부
 3. 소리를 재생해주는 부
 4. 사용자가 주파수를 변경하는 조작(버튼)부
 5. 사용자가 볼륨을 변경하는 조작(버튼)부
- 객체 지향 프로그래밍에서는, 위의 역할을 각각 수행하는 각각의 레고 블록을 만들어 조립하는 방식으로 '라디오'를 만들 수 있습니다.
- 가령 추후에 '시계' 블록을 만들어 이 라디오에 덧붙여 '시계 기능이 있는 라디오'를 만들 수도 있고,
- 나중에 TV를 만들 때, 3번의 소리를 재생해주는 블록을 재사용할 수도 있게 될 것입니다.
- 여기서 중요한 개념들이 몇 가지 등장합니다.
 1. 우리는 블록을 만들기 전에 일종의 '금형', 그러니까 '빵틀'을 만드는 작업을 할 것입니다. 무슨 이야 기인고 하니, '주파수 수신부' 블록을 직접 만드는게 아니라, 언제든지 '주파수 수신부' 블록을 찍어 낼 수 있도록 빵틀을 만드는 작업을 할 것입니다.
 2. 주파수 버튼과 볼륨 버튼 빵틀을 만들기 전에, 우리는 먼저 이 두가지의 공통 요소인 '버튼' 빵틀을 만들고, 이를 이용해 주파수 변경 버튼, 볼륨 변경 버튼 빵틀을 만들어 낼 수 있을 것입니다. (상속)
 3. 우리는 만들어진 라디오가 내부적으로 어떻게 동작하는지 전혀 알 필요가 없습니다. 버튼만 잘 눌러 지고, 소리만 잘 나오면 되니까요. 이와 비슷하게, '주파수 수신부' 블록도, 라디오를 만드는 사람 입장에서는 내부적으로 어떻게 동작하는지 알아야 할 필요가 전혀 없습니다. 그냥 요청한 주파수에 대해 수신된 전파만 잘 받아 return 해주면 되는 것이니까요. (추상화/캡슐화/인터페이스)
 4. 결국 각각의 블록들은 어떠한 작동을 하고, 어떠한 상태를 가지고만 있는 기능별 '부품'이라고 생각 하면 좋습니다.
- 이 객체지향 프로그래밍에서, 각각의 빵틀을 '클래스(Class)', 빵틀로 찍어낸 블록을 '객체(Object)', 해당 블록이 하는 행위(함수)를 '메소드(Method)', 메소드 중에서도 이 블록을 사용하는 사람들이 꼭 사용해야해서 외부로 노출된 것(라디오로 치면 버튼)을 '인터페이스(Interface)'라고 한답니다.
- 최대한 쉽게 풀어쓴다고는 해보았지만, 결국 패러다임에 대한 이야기이고 약간은 복잡해서 처음엔 바로 이해가 되지 않을 수도 있고, 지금 설명한 내용은 정말 빙산의 일각일 뿐입니다. 하지만 개발을 계속 하다보면 언젠가 이해하게 되는 날이 오게 될 것입니다!
- 객체지향 프로그래밍에 대해 더 알고 싶으면 인터넷에 좋은 자료들이 많이 있으니 찾아보면 큰 도움이 될 것 같습니다.
- 이제 Python의 Class를 통해 객체지향에 대한 이해를 한 층 높여봅시다!

In [10]:

```
# 사람 클래스(빵틀)를 만들어봅시다.
class Person(): # class 이름은 첫 글자를 대문자로 쓰는게 컨벤션이랍니다.

    # 클래스 내부 멤버들..
    # 사람에게는 이름과 나이라는 상태가 있지요.
    name = ""
    age = 0

    # 희안한 모양이죠? 생성자라고 객체에 존재하는 것입니다.
    # 객체가 처음 생성될 때 해야할 일들을 함수로 만든 것입니다.
    def __init__(self, name, age):
        # 클래스의 모든 메소드에는 앞에 클래스 자기 자신을 가르키는 'self' 매개변수가 붙게됩니다.
        self.name = name # 클래스 내부의 무언가(멤버)에 접근하기
                          # 위해서는 self(나 자신을 가르키는)를 이용합니다.
        self.age = age

    # 사람은 말을 하는 기능을 가지고 있습니다.
    def talk(self, what_to_say):
        print "안녕하세요. 저는 {0}이고, 제가 하고 싶은 말은 {1}입니다.".\
                           format(self.name, what_to_say)

# 사람이라는 빵틀을 이용해 아이유도 만들고..
iu = Person("아이유", 23)
print iu.name # .을 이용해 객체 내부 멤버에 접근할 수 있습니다.
print iu.age
iu.talk("반갑습니다!")

# 사람이라는 빵틀을 이용해 설현도 만들고..
sh = Person("설현", 22)
print sh.name
print sh.age
sh.talk("야호!")
```

아이유

23

안녕하세요. 저는 아이유이고, 제가 하고 싶은 말은 반갑습니다!입니다.

설현

22

안녕하세요. 저는 설현이고, 제가 하고 싶은 말은 야호!입니다.

In [11]:

```
# 사람 클래스를 확장(상속)해서 배우 클래스를 만들어봅시다.
class Actor(Person):    # Actor는 Person을 상속(확장) 받았기 때문에,
                        # 기본적으로 Person이 가지고 있는 모든 것들을 가지고 있답니다.

    casted_movies = []

# Person에서 상속 받아서 그대로 구현되어 있습니다.
tom = Actor("톰 행크스", 61)
print tom.name
print tom.age
tom.talk("Hi!")

# 영화(Movie) 클래스를 만들어서 Actor의 casted_movies에 넣어봅시다.
class Movie():

    name = ""
    year = 0

    def __init__(self, name, year):
        self.name = name
        self.year = year

tom.casted_movies.append(Movie("Forest Gump", 1994))
tom.casted_movies.append(Movie("The Terminal", 2004))
tom.casted_movies.append(Movie("Cast Away", 2000))

print tom.casted_movies # 객체 그 자체가 출력됩니다.

for movie in tom.casted_movies:
    print movie.name, movie.year

print tom.casted_movies[0].name # 이렇게 . . 으로 접근할 수도 있습니다.
                            # 객체 속에 객체라면 말이죠.
```

톰 행크스

61

안녕하세요. 저는 톰 행크스이고, 제가 하고 싶은 말은 Hi!입니다.

[<__main__.Movie instance at 0x112af77e8>, <__main__.Movie instance at 0x112af7a28>, <__main__.Movie instance at 0x112af7488>]

Forest Gump 1994

The Terminal 2004

Cast Away 2000

Forest Gump

모듈

In [86]:

```
# 모듈은 함수, 변수, 클래스 등을 모아 놓은 것(파일).
# 이는 다른 파일에서 불러와 사용할 수 있습니다.

# 모듈 만들기
# util.py를 만들고, 그 안에 다음과 같은 함수를 만들어봅시다.
def sum_func(*args):
    result = 0
    for i in args:
        if type(i) == int:
            result += i
    return result

def echo_func(say):
    return "You said {}".format(say)

# 그리고 원래 우리가 작업하던 main.py에서 다음과 같은 소스를 입력합니다.
import util # .py는 제외하고 파일명을 입력합니다.
print util.sum_func(1, 2, 3)
print util.echo_func("Hello!")

# "util."을 생략하기 위해서는 다음과 같이 from .. import .. 문을 사용합니다.
from util import sum_func, echo_func

sum_func(4, 5, 6) # 이 파일에서 선언한 것처럼 사용할 수 있습니다!

# from .. import *(아스테리크) 를 이용하면 해당 모듈에 있는
# 모든 것들을 바로 사용할 수 있습니다.

# 모듈을 이용하면 조금 더 확장성 있는, 규모 있는 프로그램을 작성 할 수 있게 됩니다.
# 큰 프로그램을 어떻게 모듈별로 나눌지는 개발을 하다 보면 감이 생기게 된답니다.

# 해봅시다!
# calc.py를 만들어 sum_func(), minus_func(),
# multi_func(), div_func()를 함수로 만들고, 이를 main.py에서 실행해보세요.
```

```
-----
-----  
ImportError                                     Traceback (most recent
call last)  
<ipython-input-86-99237bad07b4> in <module>()  
  14  
  15 # 그리고 원래 우리가 작업하던 main.py에서 다음과 같은 소스를 입력합니다.  
---> 16 import util # .py는 제외하고 파일명을 입력합니다.  
  17 print util.sum_func(1, 2, 3)  
  18 print util.echo_func("Hello!")
```

```
ImportError: No module named util
```

Python 기본 라이브러리

In [12]:

```
# 위에서 배운 내장함수 외에도, 모듈 import를 통해 이용할 수 있는
# 파이썬 기본 라이브러리(Python Standard Library)들도 존재한답니다.
# Python Documentation: https://docs.python.org/2/library/index.html
# 에서도 확인 할 수 있습니다.
# 그 중 유용하게 쓸 수 있는 것들을 정리해보았습니다.

#####
# datetime
from datetime import datetime

now = datetime.now()
print now # 현재 시각을 나타냅니다.

from datetime import timedelta

delta_three_hours_before = timedelta(hours=3)
three_hours_before = now - delta_three_hours_before
print three_hours_before # timedelta를 이용하면 날짜, 시간 간의 덧셈, 뺄셈이 가능합니다.

formatted_now = now.strftime("오늘은 %Y년 %m월 %d일(%a) \
    %p %I시 %M분 %S초 입니다. 오늘은 올해 중 %U번째 주입니다.")
print formatted_now

# 해봅시다!
# 오늘 날짜를 기준으로 n일 후가 며칠인지 알려주는 함수를 만들어봅시다.
# strftime()을 이용하여 예쁘게 표시해보세요.

#####
# math
import math

print math.pi # 3.14...

rad_45 = math.radians(90) # 90도에 해당하는 라디안 값을 구한 뒤
print math.sin(rad_45) # sin(90도)를 구할 수 있습니다.

print math.log10(100) # 상용로그 값 구하기
```

2016-01-21 18:09:20.549488

2016-01-21 15:09:20.549488

오늘은 2016년 01월 21일(Thu) PM 06시 09분 20초입니다. 오늘은 올해 중 03번째 주입니다.

3.14159265359

1.0

2.0

In [13]:

```
#####
# random
# 임의의 난수를 만들어줍니다.

import random

print random.randrange(1, 11) # 1~10중 아무 값이나 나타내기

fruits = ["Apple", "Lemon", "Starwberry"]
print "당신이 좋아하는 과일은 {0}입니다.".format(random.choice(fruits))

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
random.shuffle(numbers)
print numbers # 리스트의 순서를 무작위로 뒤섞습니다.

#####
# OrderedDict
"""
순서가 보장된 딕셔너리입니다.

기존에 우리가 사용하던 리스트와 다르게 딕셔너리는 순서가 보장되지 않습니다.
그 이유는 컴퓨터에서 자료를 저장하는 구조 상, 딕셔너리는 가장 최적의 탐색 성능을 가지기 위해(쉽게 풀이
내부적으로 효율적인 저장 방식으로 순서를 지정하기 때문에 인위적으로 그 순서를 변경/보장 할 수는 없습니다)
하지만 이러한 단점을 개선하기 위해 'Ordered Dictionary'라는 자료 구조가 존재하는데,
이전에는 import를 배우지 않았기 때문에 사용하지 못했지만 이제는 가능하답니다!
"""

fruits = {
    "apple": 3000,
    "lemon": 5000,
    "melon": 10000
}

print fruits # 이렇게 순서가 내 마음대로 나오지 않는 것을 볼 수 있습니다.

from collections import OrderedDict

ordered_fruits = OrderedDict([('apple', 3000), ('lemon', 5000), ('melon', 10000)])
print ordered_fruits # 이렇게 보면 이게 딕셔너리가 맞나? 싶지만
print ordered_fruits["apple"] # 딕셔너리처럼 사용할 수 있답니다.

# 결국 배열로 선언한거니까 당연히 순서가 보장되는거고, 굳이 OrderedDict()를 쓸 필요가 있나?
temp = [("apple", 3000), ("lemon", 5000), ("melon", 10000)]
print temp["apple"] # 딕셔너리처럼 쓸 수가 없습니다.
```

8

당신이 좋아하는 과일은 Lemon입니다.

```
[7, 3, 9, 6, 2, 10, 8, 4, 1, 5]
{'melon': 10000, 'lemon': 5000, 'apple': 3000}
OrderedDict([('apple', 3000), ('lemon', 5000), ('melon', 10000)])
3000
```

TypeError

Traceback (most recent)

```
call last)
<ipython-input-13-0635b5bf21db> in <module>()
    43 # 결국 배열로 선언한거니까 당연히 순서가 보장되는거고, 굳이 OrderedDict
t()를 쓸 필요가 있나?
    44 temp = [ ("apple", 3000), ("lemon", 5000), ("melon", 1000
0)]
--> 45 print temp["apple"] # 딕셔너리 처럼 쓸 수가 없습니다.

TypeError: list indices must be integers, not str
```

파이썬 패키지 활용 (PIP)

파이썬에는 기본 라이브러리 외에도 많은 개발자들이 만들어놓은 수 많은 라이브러리들이 존재한답니다. 우리가 차후에 데이터 분석을 할 때에도, 이러한 데이터 분석용 라이브러리를 다운로드 받아 활용할 예정입니다.
(<https://pypi.python.org/> (<https://pypi.python.org/>)에서 다른 파이썬 라이브러리들이 무엇이 있는지 목록들을 확인 할 수 있습니다.)

pip 설치하기

외부 파이썬 라이브러리들을 이용하기 위해서는 pip라는 툴을 설치해야 합니다.

윈도우 혹은 맥이신 분!

1. 다음의 링크에 접속한다: <https://bootstrap.pypa.io/get-pip.py> (<https://bootstrap.pypa.io/get-pip.py>)
2. 마우스 오른쪽 -> 다른이름으로 저장. 위치는 파일참으로 작업하고 있는 main.py 파일이 있는 곳으로 한다.
3. 파일참으로 돌아가, View -> Tools Windows -> Terminal (Alt+F12)를 누른다.
4. 아래 검은 터미널 창이 뜨면, dir(맥은 ls)를 입력해 아까 다운로드 받은 get-pip.py가 잘 있는지 확인하고, python get-pip.py를 입력한다.
5. 설치가 다 되면, 터미널에 pip --version이라고 입력해 설치가 잘 되었는지 확인한다.

리눅스(우분투)이신 분!

1. 터미널에서 sudo apt-get update 후에, sudo apt-get install python-pip python-dev build-essential를 입력한다.

외부 라이브러리를 이용해 양력<->음력 변환 프로그램 만들어보기

In [84]:

```
# 외부 라이브러리를 이용해 양력<->음력 변환 프로그램을 만들어봅시다.  
# 자세한 라이브러리에 대한 설명은 아래 페이지를 참고하세요.  
# https://github.com/lidaobing/python-lunardate  
  
# 터미널에서 다음과 같이 입력합니다.  
# pip install lunardate  
  
# main.py  
from lunardate import LunarDate  
  
print LunarDate.fromSolarDate(2016, 1, 23)  
# 올해 설날을 구해봅시다.  
print LunarDate(2016, 1, 1, 0).toSolarDate() # 4번째 파라미터는 윤달인지 아닌지  
  
LunarDate(2015, 12, 14, 0)  
2016-02-08
```

도전과제!

- 위에서 해본 소설에서 사용된 단어들의 목록과 그 횟수를 나타내는 프로그램을 응용하여, 그 중 가장 많이 사용된 단어 5개를 순서대로 나타내고, 몇 번씩 사용되었는지를 딕셔너리 형태로 나타내는 프로그램을 만들어봅시다.
- 힌트: 구글에서 "python dictionary sorting" 라고 검색해보기.

In [85]:

```
# 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"  
{}
```