

# 4주차: 모듈, 파이썬 외부 패키지(pip), 예외처리, 정규표현식

## 모듈

In [25]:

```
# 모듈은 함수, 변수, 클래스 등을 모아 놓은 것(파일).
# 이는 다른 파이썬 프로그램에서 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)
```

In [3]:

```
# 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 것 처럼 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 만들.
```

# Python 기본 라이브러리

In [27]:

```
# 위에서 배운 내장함수 외에도,  
# 모듈 import를 통해 이용할 수 있는  
# 파이썬 기본 라이브러리(Python Standard Library)들도 존재합니다.  
# Python Documentation  
# https://docs.python.org/2/library/index.html에서도 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. 오늘  
print formatted_now  
  
### 해봅시다!  
# 오늘 날짜를 기준으로 n일 후가 며칠인지 알려주는 함수를 만들어봅시다. strftime()을 이용하여 예쁘게
```

2016-01-29 03:48:14.396070

2016-01-29 00:48:14.396070

오늘은 2016년 01월 29일(Fri) AM 03시 48분 14초입니다. 오늘은 올해 중 04번째 주입니다.

In [6]:

```
#####  
# 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
```

3.14159265359

1.0

2.0

In [28]:

```
#####
# random
# 임의의 난수를 만들어줍니다.

import random

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

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.
```

6

당신이 좋아하는 과일은 Lemon입니다.

[2, 3, 1, 5, 7, 8, 6, 9, 10, 4]

기존에 우리가 사용하던 리스트와 다르게 딕셔너리는 순서가 보장되지 않습니다. 그 이유는 컴퓨터에서 자료를 저장하는 구조 상, 딕셔너리는 가장 최적의 탐색 성능을 가지기 위해(쉽게 풀이하면, "키"가 주어졌을 때, 가장 빠르게 "값"을 찾아내기 위해) 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!

In [12]:

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

In [13]:

```
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"] # 딕셔너리 처럼 쓸 수가 없습니다.

{'melon': 10000, 'lemon': 5000, 'apple': 3000}
OrderedDict([('apple', 3000), ('lemon', 5000), ('melon', 10000)])
3000
```

# 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 파이참으로 작업하고 있는 main.py 파일이 있는 곳으로 한다.
3. 파이참으로 돌아가, View -> Tools Windows -> Terminal (Alt+F12)를 누른다.
4. 아래 검은 터미널 창이 뜨면, dir(맥은 ls)을 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 [4]:

```
# 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
```

# 지속 가능한 프로그래밍 학습법

- 프로그래밍의 실력 == '검색력'

얼마나 구글에서 검색을 잘 하는지는 프로그래밍의 실력과 직접적으로 연관됩니다. 가령 어떠한 문제가 주어졌을 때, 본인이 이전에 가지고 있는 지식과 기술로 문제를 해결할 수도 있겠지만, 상당수의 경우 인터넷을 찾아보는 것이 문제 해결을 가장 빨리 할 수 있는 방법입니다. 일반적으로 구글에 검색을 하게 되면 해외 프로그래밍계의 지식인 같은 서비스인 Stack overflow에 좋은 선례들이 많이 나와있음을 알 수 있습니다. 따라서 안타깝게도 영어는 프로그래밍을 배우고 실력을 늘리는데 필수 불가결한 요소랍니다. 더불어 이렇게 검색해서 나온 솔루션을 최종적으로는 본인의 것으로 만들어야 진짜 실력자가 될 수 있겠지요!

처음에는 어떠한 검색으로 검색을 해야할지 조차 막막할 것입니다. 구글의 검색 엔진이 워낙 좋다보니, 처음에는 해당 상황에 맞는 특수한 전문 용어(단어)가 꼭 들어가 있지 않아도 검색이 잘 됩니다. 가령 우리 예전에 배웠던 소설에서 가장 많이 사용된 단어와 그 숫자를 딕셔너리로 리턴해주는 함수를 만들 때, 이를 최빈(가장 많이 나온) 순서대로 정렬을 하는 프로그램으로 바꾸고 싶다고 생각을 해봅시다. 가장 핵심 키워드는 '정렬(order)' 정도겠죠? 그럼 구글에 'how to order dictionary by value in python'이라고 검색을 하면 좋은 아티클들이 무수히 쏟아져 나옵니다. 그리고 나중에 점점 프로그래밍과 검색에 능숙해지면 'python dictionary sorting' 정도로 검색을 할 수도 있겠죠.

- Python 언어 자체에 대해 더 공부하고 싶다면?

지금까지 수업을 잘 따라왔음에도 불구하고, 아직 본인이 Python 기초에 대한 지식이 많이 부족하다고 생각된다면 인터넷에 무료로 공개되어있는 점프 투 파이썬 (<https://wikidocs.net/book/1>)을 읽어보시기를 권장합니다. 해외의 Team Treehouse (<https://teamtreehouse.com/>) 등의 유수의 강의를 다시 한 번 보시는 것도 추천드립니다. 더불어 만약 Python에 대해 더 깊게 알고 싶은 분들은 Intermediate Python (<http://book.pythontips.com/en/latest/index.html>)이라는 아티클을 한 번 참고해보시고, 예전에 수업 중에 말씀드렸던 것처럼, Python 기본 Documentation (<https://docs.python.org/2.7/>)을 일독하는 것도 한 번쯤은 권장합니다. 무어든지 기본이 중요하니까요.

- 무엇을 만들어보고 싶나요?

언어는 말 그대로 '도구'입니다. 이 도구를 얼마나 능숙하게 다루느냐의 문제보다는 결국 이 도구로 무엇을 만들어낼 것인지를 중요할 것입니다. Python은 강력하고도 쉬운 언어(도구)이며, Python을 이용하면 (다양한 종류의 Python 외부 라이브러리들을 활용해서) 거의 모든 종류의 컴퓨터 프로그램을 만들어 낼 수 있습니다. 여러분들은 데이터 분석을 목적으로 Python 공부를 시작하셨지만, 이제 Python을 다룰수 있게 된 바, '웹 서비스', '게임', '윈도우 프로그램' 등을 다양하게 만들어낼 수 있습니다. 어떻게 시작할지 막막하시면, 위에서 말씀드렸던 것처럼 '구글에 한 번 검색해보세요.' 좋은 안내서들이 잔뜩 나올 것입니다.

## 예외처리하기 (try-except-finally)

In [7]:

```
# 강제로 어떠한 숫자를 '0'으로 나누어봅시다. 프로그램이 비정상적으로 종료될 것입니다.  
print 10/0
```

```
-----  
-----  
ZeroDivisionError Traceback (most recent  
call last)  
<ipython-input-7-6549ddc6b4b8> in <module>()  
      1 # 강제로 어떠한 숫자를 '0'으로 나누어봅시다. 프로그램이 비정상적으로 종료  
      될 것입니다.  
----> 2 print 10/0
```

```
ZeroDivisionError: integer division or modulo by zero
```

In [14]:

```
# 이러한 예외상황(Exception)에서 프로그램을 종료시키지 않고 프로그래머가 이를 제어할 수 있는 방법이  
  
try:  
    print "0으로 나누어볼까요?"  
    print 10/0  
    print "짜잔! 성공했습니다!"  
except:  
    print "실패했습니다! 당연히 0으로 나눌 수는 없겠지요."  
  
# 이를 디버깅을 해보면, try문 안을 실행하다가 예외가 발생하는 순간 바로 except 문으로 빠지는 것을
```

0으로 나누어볼까요?

실패했습니다! 당연히 0으로 나눌 수는 없겠지요.

In [16]:

```
# 정상적인 구문이라면 except는 실행되지 않을 것입니다.  
  
try:  
    print "Hello"  
except:  
    print "Bye"
```

Hello

In [17]:

```
# 어떠한 종류의 예외가 발생했는지 알아볼 수도 있습니다.
```

```
try:  
    print 10/0
```

```
except Exception as e:  
    print e
```

# 생겼던 문법이죠?

# 예외가 발생하면, except문에서는 이 예외를 오른쪽의 예외 객체에 담아준답니다.

# 예외는 여러가지 종류가 있기 때문에 다양한 종류의 예외 클래스들이 존재하는데, 그 예외 클래스들의 가장

# 모든 예외들의 기본 형이고, 다른 예외들은 모두 이 클래스에서 상속받은 클래스라고 할 수 있겠지요.

# 어째든 저 문법은 '지금 발생한 예외를 Exception 클래스로 받아볼꺼고, 이걸 e라는 변수에 할당하겠어'

# 실제로 해당 예외의 에러메시지인 'integer division or modulo by zero'가 나타나는 것을 알

# e는 예외 객체이기 때문에, 실제 디버깅해보면 에러 메시지 뿐만 아니라 다양한 정보를 가지고 있다는 것을

```
integer division or modulo by zero
```

In [20]:

```
# 위의 구문을 try-except (혹은 try-catch) 구조라고 한답니다.
```

# 안정성있는 프로그램을 만들기 위해, 프로그램은 기본적으로 try-catch 구조로 작성하는 것을 원칙으로

# (예측할 수 없는 오류로 인해 프로그램이 꺼지는 것을 방지하기 위해서입니다!)

# 이러한 것을 통칭 '예외처리 하다'라고 합니다.

```
# try-except 외에 try-except-finally 구조도 존재합니다.
```

```
# finally문 안의 코드들은 위에서 결과가 어찌되었든 '무조건' 실행됩니다.
```

In [22]:

```
try:  
    print "숫자형과 문자열형을 더해보겠습니다."  
    print "10" + 5  
    print "잘 더해졌습니다!"  
except Exception as e:  
    print "예외가 발생했습니다."  
    print "예외의 메시지는 다음과 같습니다: {}".format(e)  
finally:  
    print "어쨌든 프로그램의 끝!"
```

숫자형과 문자열형을 더해보겠습니다.

예외가 발생했습니다.

예외의 메시지는 다음과 같습니다: cannot concatenate 'str' and 'int' objects

어쨌든 프로그램의 끝!

In [23]:

```
try:  
    print "숫자형과 숫자형을 더해보겠습니다."  
    print 10 + 5  
    print "잘 더해졌습니다!"  
except Exception as e:  
    print "예외가 발생했습니다."  
    print "예외의 메시지는 다음과 같습니다: {}".format(e)  
finally:  
    print "어쨌든 프로그램의 끝!"  
  
# finally 구문 안에 있는 코드는 예외가 발생하든 안하든 무조건 실행된다는 것을 알 수 있습니다.  
# 만약 open()을 이용해서 파일을 열었을 때, 안에서 예외가 발생하더라도(혹은 발생하지 않더라도) 무조건  
# 이렇게 finally 안에 넣으면 되겠지요?
```

숫자형과 숫자형을 더해보겠습니다.

15

잘 더해졌습니다!

어쨌든 프로그램의 끝!

In [28]:

```
# 해봅시다  
# 아까 만든 calc.py안의 사칙연산 함수를 try-except문을 이용하여,  
# 문법적으로 계산이 불가능한 매개변수 타입이 들어왔을 때(예외가 발생했을 때) 사용자에게 안내 문구를 출력해보겠습니다.
```

## 고급 문자열 검색/치환법: 정규표현식

- 다음의 사이트를 참고하여 학습하겠습니다: <http://regexr.com/>

In [23]:

```
import re

result = re.findall("pattern", "string")
```