2014 SPARCS Seminar

# PYTHON SEMINAR 2

박준우 zoonoo

# 0 지난 시간에 뭘 배웠지?
# PYTHON 1 복습

Template Code:

```
print "최대 별 개수는?"
n=input()
print "1, 2, 3, 4"
t=input()
star(n,t)
```

쓰는 개념 : 함수, if, for

```
최대 별 개수는?
10
1, 2, 3, 4
1
*
**
***
****
*****
******
*******
********
*********
**********
```

```
최대 별 개수는?
10
1, 2, 3, 4
2
         *
        **
       ***
      ****
     *****
    ******
   *******
  ********
 *********
**********
```

```
최대 별 개수는?
10
1, 2, 3, 4
3
**********
*********
********
*******
******
*****
****
***
**
*
```

```
최대 별 개수는?
10
1, 2, 3, 4
4
**********
 *********
  ********
   *******
    ******
     *****
      ****
       ***
        **
         *
```

# 0 지난 시간에 뭘 배웠지?
# PYTHON 1 복습

함수 이름은 ahae()로!
아해가 몇 명인가?
7
제1의아해도무섭다고그리오
제2의아해도무섭다고그리오
제3의아해도무섭다고그리오
제4의아해도무섭다고그리오
제5의아해도무섭다고그리오
제6의아해도무섭다고그리오
제7의아해도무섭다고그리오

```
>>> def ahae():
...         print("아해가 몇 명인가?")
...         n = input()
...         for i in range(1, n+1):
...                 print("제"+ str(i) + "의아해도무섭다고그리오")
...
```

# 0 지난 시간에 뭘 배웠지?
# PYTHON 1 복습

- str, bool 같은 자료 구조들
- if문, for문
- 함수, 변수, input()
- .py파일 이용 등등

# 0 지난 시간에 뭘 배웠지?
# PYTHON 1 복습

## 오늘은...

- List, Tuple and Dictionary
- Module
- Class
- Exception
- 객체 지향 프로그래밍

# 1 기본적인 자료구조들
# List, Tuple and Dictionary

## 두개 이상의 값을 저장하는 구조들

- List

```
l = [0, 1, 2, 3, 4]
```

- Tuple

```
t = (0, 1, 2, 3, 4)
```

- Dictionary

```
d = {'zoonoo': 12, 'egg': 13, 'chaos': 12}
```

# 1 기본적인 자료구조들
## List, Tuple and Dictionary

# Sequential Data Type

- List

```
>>> a = [0, 1, 2, 3, 4]
>>> type(a)
<type 'list'>
```

  - Mutable: 내부의 값을 나중에 바꿀 수 있다.

```
>>> a = [1, 2, 3]
>>> a[1] = 5
>>> a
[1, 5, 3]
```

# 1 기본적인 자료구조들
# List, Tuple and Dictionary

## Sequential Data Type

- Tuple

```
>>> b = (0, 1, 2, 3, 4)
>>> type(b)
<type ‘tuple’>
```

  - Immutable: 한 번 값을 정하면 내부의 값을 바꿀 수 없다.

```
>>> b = (1, 2, 3)
>>> b[1] = 5
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
TypeError: 'tuple' object does not support item assignment
>>> b
(1, 2, 3)
```

# 1 기본적인 자료구조들
## List, Tuple and Dictionary

# Sequential Data Type

- List
  - Mutable: 내부의 값을 나중에 바꿀 수 있다.
- Tuple
  - Immutable: 한 번 값을 정하면 내부의 값을 바꿀 수 없다.

# 1 기본적인 자료구조들
## List, Tuple and Dictionary

# 그렇다면 이런 경우는?

```
>>> c = ([1, 2, 3], [2, 3], [5, 6])
>>> type(c)
<type ‘tuple’>
>>> type(c[0])
<type ‘list’>
>>>
>>> c[0][0] = 4
???
```

# 1 기본적인 자료구조들
# List, Tuple and Dictionary

## 그렇다면 이런 경우는?

```
>>> c = ([1, 2, 3], [2, 3], [5, 6])
>>> type(c)
<type 'tuple'>
>>> type(c[0])
<type 'list'>
>>>
>>> c[0][0] = 4
>>> c = ([4, 2, 3], [2, 3], [5, 6])
```

- 튜플 내부의 리스트를 편집하는 것은 가능하다. 리스트의 reference를 그대로 가지고 있지만, 리스트 자체는 편집 되는 것.

# 1 기본적인 자료구조들
# List, Tuple and Dictionary

## List의 다양한 메소드들

- list.append(1)

```
>>> a = [0, 1, 2, 3, 4]
>>> a.append(5)
>>> print a
[0, 1, 2, 3, 4, 5]
```

# 1 기본적인 자료구조들
# List, Tuple and Dictionary

## List의 다양한 메소드들

- ', '.join(list)

```
>>> a = ['0', '1', '2', '3', '4']
>>> print ', '.join(a)
0, 1, 2, 3, 4
```

```
0, 1, 2, 3, 4
```

', ' 쉼표 + 띄우기

# 1 기본적인 자료구조들
## List, Tuple and Dictionary

# List의 다양한 메소드들

- map

```
>>> a = [0, 1, 2, 3, 4]
>>> square = lambda x: x ** 2
>>> print map(square, a)
[0, 1, 4, 9, 16]
```

```
map(<함수>, <원래 리스트>)
```

# 1 기본적인 자료구조들
# List, Tuple and Dictionary

## Slicing

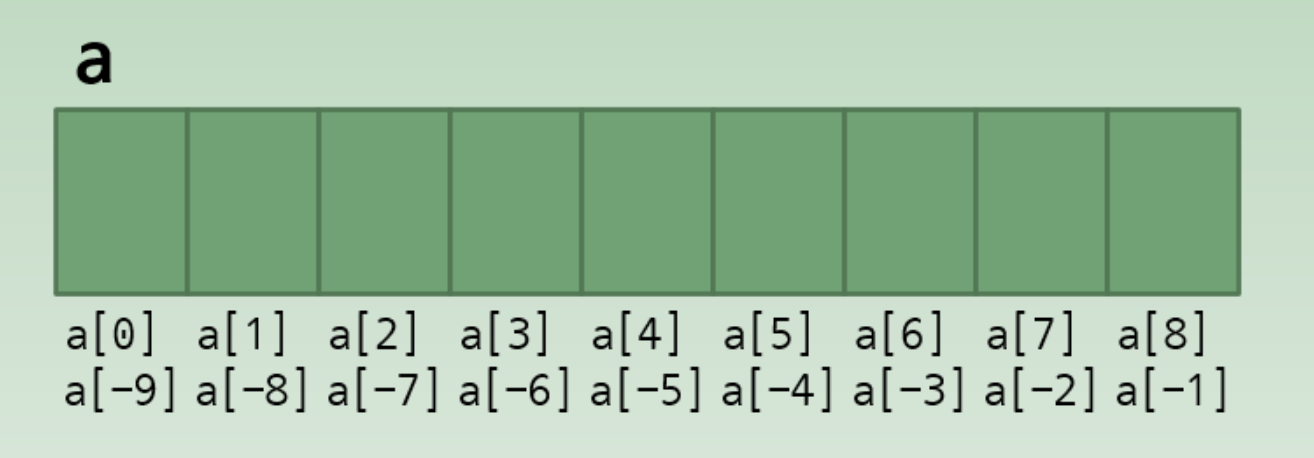
- Sequential Data Type에 한해서 문자열 혹은 리스트의 부분을 취할 수 있다.

```
>>> a = [0, 1, 2, 3, 4]
>>> b = a[0 : 3]
[0, 1, 2]
>>> c = a[2 : 4]
[2, 3]
```

# 1 기본적인 자료구조들
# List, Tuple and Dictionary

## Slicing

- Sequential Data Type에 한해서 문자열 혹은 리스트의 부분을 취할 수 있다.



# 1 기본적인 자료구조들
## List, Tuple and Dictionary

# List Comprehension

- 이미 만들어진 리스트를 이용해서 새로운 리스트를 만드는 것.

```
>>> orig = [0, 1, 2, 3, 4]
>>> b = [2 * x for x in orig if x % 2 = 0]
```

# 1 기본적인 자료구조들
## List, Tuple and Dictionary

# List Comprehension 실습


- 현재 페이지 앞 뒤로 5개 페이지

```
>>> page = 13 #현재 페이지
>>> orig = range(page-5, page+5)
```

- 이렇게 짰다. 그런데... 총 게시글 수가 충분치 않다면? 현재 페이지가 5 이하라면?

```
-3 -2 -1 0 1 2 3 4 5 6
```

# 1 기본적인 자료구조들
# List, Tuple and Dictionary

## List Comprehension 실습


```
>>> page = 13 #현재 페이지
>>> orig = range(page-5, page+5)
>>> articles = 160 # 총 게시글 수
>>> new_list = ???

Hint: (x>0) and (x<(2+(articles-1)/4))
```

```
>>> b = [2 * x for x in orig if x % 2 = 0]
```

함수 원래 리스트 조건문

# 1 기본적인 자료구조들
## List, Tuple and Dictionary

# List Comprehension 실습


- 한 페이지에 열개의 게시글

```
>>> page = 13 #현재 페이지
>>> orig = range(page-5, page+5)
>>> articles = 160 # 총 게시글 수
>>> new_list = [x for x in orig if (x>0) and
(x<(2+(articles-1)/4))]
>>> print new_list
[8, 9, 10, 11, 12, 13, 14, 15, 16]
```

# 1 기본적인 자료구조들
# List, Tuple and Dictionary

## Dictionary

- Key–Value Pair

```
>>> book1 = {"key1": "value1", "key2": "value2"}
>>> type(book1)
<type ‘dict’>

>>> book2 = dict()
>>> type(book2)
<type ‘dict’>
```

# 1 기본적인 자료구조들
# List, Tuple and Dictionary

## Dictionary

- Key–Value Pair

```
>>> book1 = {"key1": "value1", "key2": "value2"}

>>> book1["key2"] = "value4" #수정
>>> book1
{'key2': 'value4', 'key1': 'value1'}
```

# 1 기본적인 자료구조들
# List, Tuple and Dictionary

## Dictionary

- Key–Value Pair

```
>>> book1 = {"key1": "value1", "key2": "value2"}

>>> book1["key3"] = "value3" #추가
>>> book1
{'key3': 'value3', 'key2': 'value2', 'key1': 'value1'}
```

# 1 기본적인 자료구조들
# List, Tuple and Dictionary

## Dictionary

- Key–Value Pair

```
>>> book1 = {"key1": "value1", "key2": "value2"}

>>> book1["key3"] = "value3" #추가
>>> book1
{'key3': 'value3', 'key2': 'value2', 'key1': 'value1'}

>>> del book1[“key1”]          #제거
>>> book1
{'key3': 'value3', 'key2': 'value2'}
```

# 1 기본적인 자료구조들
# List, Tuple and Dictionary

## Dictionary

- Key–Value Pair
  - 정수로 인덱싱을 하는 것이 아니라 key로 value에 접근한다.
  - 순서가 고정되어 있지 않다.
  - 검색 속도를 빠르게 할 수 있다.

# List, Tuple and Dictionary

- for 문과 함께 쓰면 효과적!

```
>>> SPARCS = ('onion', 'chocho', 'leejeok', 'mandu',
'daedoo')
>>> for member in SPARCS:
        print member + "는 스팍스 회원이지롱"
onion는 스팍스 회원이지롱
chocho는 스팍스 회원이지롱
mandu는 스팍스 회원이지롱
daedoo는 스팍스 회원이지롱
```

# 1 기본적인 자료구조들
## List, Tuple and Dictionary

# for문 사용할 때 tip

```
for i in range(n):
```

보다,

```
for i in xrange(n):
```

가 더 효율적이다. 실제로 list를 생성하지 않고 같은 일을 실행한다.

# 1 기본적인 자료구조들
# List, Tuple and Dictionary

## 실습

- 영화, 가수, 수강과목을 key로 가지는 딕셔너리를 만들고, 그 value를 리스트로 구성하기
- 그리고 그 리스트 중 하나를 수정하기

```
movies = ['les Miserables', 'Dead Poets Society', 'Love Actually']
singers = ['Lee Jeok', 'Jaurim', 'Norah Jones', 'Frank Sinatra']
courses = ['SP', 'Analysis I', 'Calculus II', 'DiscreteMath',
'Environmentalism']

zoonoo = {}
zoonoo['movies'] = movies
zoonoo['singers'] = singers
zoonoo['courses'] = courses

zoonoo['movies'].append('300')

print zoonoo['movies']
```

# 2 맨날 쓰는 함수 가져다 쓰기
# Module

## 모듈의 정의

- “다른 파일에서 정의된 함수나 변수의 이름을 사용하도록 허용하는 것”
- “서로 연관된 작업을 하는 코드들의 모임”
- “ 코드를 적당한 단위로 쪼개는 것. ”

# 2 맨날 쓰는 함수 가져다 쓰기
# Module

## 모듈이 없다면...

```
#calc.py
>>> pi = 3.14
>>> def area(r):
        return pi * (r ** 2)
>>> area(3)
28.26
>>> def perimeter(r):
        return pi * 2 * r
>>> perimeter(3):
18.84
```

# 2 맨날 쓰는 함수 가져다 쓰기
# Module

## 모듈이 있다면!

```
# math.py
pi = 3.14
          ...
```

```
# calc.py
import math

>>> def area(r):
          return (math.pi) * (r ** 2)
>>> area(3)
28.26

>>> def perimeter(r):
          return (math.pi) * 2 * r
>>> perimeter(3)
18.84
```

# 2 맨날 쓰는 함수 가져다 쓰기
# Module

## 모듈이 있다면!

```
# math.py
pi = 3.14159
          ...
```

```
>>> area(3)
28.27431
>>> perimeter(3)
18.84954
```

# 2 맨날 쓰는 함수 가져다 쓰기
# Module

## Import

- import math
- import datetime
- import cs1graphics

- 미리 짜 놓은 코드들을 이용해서 좀 더 다양한 것을 할 수 있도록 한다.

# 2 맨날 쓰는 함수 가져다 쓰기
# Module

## Import

### from math import pi

```
>>> print pi
3.14
```

### import math

```
>>> print math.pi
3.14
```

# 2 맨날 쓰는 함수 가져다 쓰기
# Module

## Import

math 모듈에서 모든 것을 불러온다.

```
>>> from math import *
>>> pi
3.141592653589793
```

그러나 반드시 지양할 것.

같은 변수에 서로 다른 의미가 부여될 수 있다. E.g) 'right'

# 2 맨날 쓰는 함수 가져다 쓰기
# Module

## 실습

math라는 모듈에서 'sin()'과 'cos()'를 import해서 어떤 값을 받아도 1을 리턴하는 함수를 만들자!

$$sin^2(x) + cos^2(x) = 1$$

# 2 맨날 쓰는 함수 가져다 쓰기
# Module

## 실습

math라는 모듈에서 'sin()'과 'cos()'를 import해서 어떤 값을 받아도 1을 리턴하는 함수를 만들자!

$$sin^2(x) + cos^2(x) = 1$$

```
import math

def one(x):
    return (math.sin(x)**2 + math.cos(x)**2)

number = input('x: ')
print one(number)
```

# 3 붕어빵 찍어내기 Class

# 클래스란?

- 새로 정의한 자료형

```
>>> class person(object) :
        def __init__(self, name, year, major):
                self.name = name
                self.year = year
                self.major = major
                …
```

# 3 붕어빵 찍어내기 Class

# 클래스 예시 – OTL의 강의

zoonoo@bit: ~/otl/otl/apps/timetable/models.py (63/226) - VIM

```
18
19 class Lecture(models.Model):
20     """특정 년도 학기에 개설된 과목 instance를 가리키는 모델"""
21     code = models.CharField(max_length=10,db_index=True)                    # 과목코드 (12.123 형식)
22     old_code = models.CharField(max_length=10, db_index=True)               # 과목코드 (ABC123 형식)
23     year = models.IntegerField(db_index=True)                               # 개설년도 (4자리 숫자)
24     semester = models.SmallIntegerField(choices=SEMESTER_TYPES,db_index=True) # 개설학기 (1=봄, 2=여름, 3=가을, 4=겨울)
25     department = models.ForeignKey(Department)                  # 학과
26     class_no = models.CharField(max_length=4, blank=True)   # 분반
27     title = models.CharField(max_length=100, db_index=True)                  # 과목이름 (한글)
28     title_en = models.CharField(max_length=200, db_index=True)               # 과목이름 (영문)
29     type = models.CharField(max_length=12)                      # 과목구분 (한글; '전필', '전선', '기필', ...)
30     type_en = models.CharField(max_length=36)                   # 과목구분 (영문; 'Major Required', 'Major Elective', ...)
31     audience = models.IntegerField(choices=AUDIENCE_TYPES)  # 학년구분
32     credit = models.IntegerField(default=3)                     # 학점
33     num_classes = models.IntegerField(default=3)                # 강의 시간
34     num_labs = models.IntegerField(default=0)                   # 실험 시간
35     credit_au = models.IntegerField(default=0)                  # AU
36     limit = models.IntegerField(default=0)                      # 인원제한
37     num_people = models.IntegerField(default=0, blank=True, null=True)  #신청인원
38     professor = models.ManyToManyField('dictionary.Professor', related_name='lecture_professor', null=True) # 교수님
39     notice = models.CharField(max_length=200, blank=True, null=True)        # 비고
40     is_english = models.BooleanField()                          # 영어강의 여부
41     deleted = models.BooleanField(default=False)                # 과목이 닫혔는지 여부
42     rating = models.ForeignKey('dictionary.LectureRating', related_name='lecture_rating', null=True, blank=True)
43     #''' lecture에서 rating찾아갈때 null인경우 고려 '''
44     course = models.ForeignKey('dictionary.Course', related_name='lecture_course')
45
46     timetable_relation = models.ManyToManyField(User, through='Timetable', null=True, blank=True)
47
```

# 3 붕어빵 찍어내기 Class

# 클래스 예시 – OTL의 강의

```
38    professor = models.ManyToManyField('dictionary.Professor', related_name='lecture_professor', null=True) # 교수님
39    notice = models.CharField(max_length=200, blank=True, null=True)        # 비고
40    is_english = models.BooleanField()                    # 영어강의 여부
41    deleted = models.BooleanField(default=False)          # 과목이 닫혔는지 여부
42    rating = models.ForeignKey('dictionary.LectureRating', related_name='lecture_rating', null=True, blank=True)
43    #''' lecture에서 rating찾아갈때 null인경우 고려 '''
44    course = models.ForeignKey('dictionary.Course', related_name='lecture_course')
45
46    timetable_relation = models.ManyToManyField(User, through='Timetable', null=True, blank=True)
47
48    def __unicode__(self):
49        return u'%s (%d:%s) %s' % (self.code, self.year, self.get_semester_display(), self.title)
50
51    def update_num_people(self):
52        self.num_people = len(set(self.timetable_relation.all()))
53        self.save()
54        return self.num_people
55
56    def check_classtime_overlapped(self, another_lecture):
57        """이 과목과 주어진 다른 과목의 강의 시간 중 겹치는 것이 있는지 검사한다."""
58        my_times = self.classtime_set.all()
59        their_times = another_lecture.classtime_set.all()
60
61        for mt in my_times:
62            for tt in their_times:
63                if not (mt.end <= tt.begin or mt.begin >= tt.end) and mt.day == tt.day:
64                    return True
65        return False
66
67    def check_examtime_overlapped(self, another_lecture):
68        '''이 과목과 주어진 다른 과목의 시험 시간 중 겹치는 것이 있는지 검사한다.'''
69        my_times = self.examtime_set.all()
70        their_times = another_lecture.examtime_set.all()
71
72        for mt in my_times:
73            for tt in their_times:
74                if not (mt.end <= tt.begin or mt.begin >= tt.end) and mt.day==tt.day:
75                    return True
76        return False
77
78    def dictionary_url(self):
79        return 'dictionary/' + self.code
80
81    class Meta:
82        unique_together = ('code', 'year', 'semester', 'department', 'class_no')
```

# 3 붕어빵 찍어내기 Class

# Method 메소드/메서드

- 객체 안에서 정의된 함수
  - 자료 형이나 값과 연관된 함수

```
>>> class person(object) :
        def __init__(self, name, year, major):
                                …
        def __str__(self):
                                …
        def change_major(self, new):
                self.major = new
```

3 붕어빵 찍어내기
Class

# 생성자 constructor

- 특별한 기능을 하는 메소드

```
>>> class person(object) :
        def __init__(self, name, year, major):
                self.name = name
                self.year = year
                self.major = major
        …
>>> zoonoo = person('박준우', 12, '전산학과')

>>> print zoonoo
<__main__.person object at 0x7fe700977710>
```

# 3 붕어빵 찍어내기 Class

## __str__

- 또 하나의 특별 메소드

```
>>> class person(object) :
        def __init__(self):
                ...
        def __str__(self):
        return "%s %d학번 %s" %  (self.major,
self.year, self.name)

>>> zoonoo = person('박준우', 12, '전산학과')
>>> print zoonoo
전산학과 12학번 박준우
                        …
```

이외에도 다양한 메소드가 있으니 파이선 documentation 참고!

# 3 붕어빵 찍어내기 Class

# Inheritance

```
class Rectangle(object):
        def __init__(self, x, y):
                self.x = x
                self.y = y
        def area(self):
                return self.x*self.y
class Square(Rectangle):
        def __init__(self, x):
                self.x = x
                self.y = x

sq1 = Square(5)
print sq1.area()
```

# 3 붕어빵 찍어내기 Class

## 실습

- 이름, 학번, 학과를 가지는 KAISTIAN 클래스를 만들기
- 스팍스 ID와 IP를 가지는 subclass SPARCSIAN 클래스 만들기

4 문제가 생기지 않도록
Exception

# Exception이란?

- 우리가 생각하지 못했던 부분에서 문제가 발생할 때

# 4 문제가 생기지 않도록 Exception

# Exception이란?

- 두가지 문제
  1. Syntax Error: 문법적인 오류
  2. Exception: 문법은 맞지만 진행할 수 없는 상황
     - 0으로 나누는 경우 `ZeroDivisionError`
     - 문자열과 숫자를 더하는 경우 `TypeError`
     - 범위를 넘어선 인덱스 사용 `IndexError`
     - 딕셔너리에 없는 key를 사용할 경우 `KeyError`

4 문제가 생기지 않도록
Exception

# Exception-handling이란?

에러가 생기는 경우, 단순히 프로그램이 끝나는 게 아니라, 그에 따른 조치를 취할 수 있도록 하는 것.

# 4 문제가 생기지 않도록 Exception

# TRY, EXCEPT, ELSE

```
try:

except 무슨무슨error:

else:
```

# 4 문제가 생기지 않도록 Exception

## try, except가 없는 경우

```
#noexception.py

def inverse(x):
    print (1.0/x)

x=input("find the inverse of :")
inverse(x)
```

# 4 문제가 생기지 않도록 Exception

## try, except가 없는 경우

```
>>>python noexception.py

find the inverse of :0
Traceback (most recent call last):
  File "noexception.py", line 4, in <module>
    inverse(x)
  File "noexception.py", line 2, in inverse
    print (1.0/x)
ZeroDivisionError: float division by zero
```

- 에러가 나서 프로그램이 끝나버림!

# 4 문제가 생기지 않도록 Exception

## try, except가 있는 경우

```
#exception.py

def inverse(x):
    try:
        print (1.0/x)
    except ZeroDivisionError:
        print "The number has no inverse. Try again"
        x = input('find the inverse of :')
        inverse(x)
    else:
        print 'No Error!'
x = input('find the inverse of :')
inverse(x)
```

# 4 문제가 생기지 않도록 Exception

## try, except가 있는 경우

```
>>>python exception.py

find the inverse of :0
The number has no inverse. Try again

find the inverse of :0
The number has no inverse. Try again

find the inverse of :2
0.5
No Error!
```

# 4 문제가 생기지 않도록 Exception

# raise를 통해 직접 예외처리

- 제대로 시행되지 않을 경우를 고려해서 직접 exception이 발생하도록 하는 것

# 4 문제가 생기지 않도록 Exception

## 실습

- 연산을 no1 * no2 번 실행하는 프로그램

```
for i in range(no1):
        for j in range(no2):
                print i + j
```

No1과 no2가 적절치 못한 숫자일 경우 raise로 직접 exception을 두어야 한다.

# 4 문제가 생기지 않도록 Exception

# 실습

- 연산을 no1 * no2 번 실행하는 프로그램

```
try:
    if no1<0 or no2 < 0:
        raise

    else:
        for i in range(no1):
            for j in range(no2):
                print i + j
except:
    print "It has to be a natural number."
```

# 객체 지향 프로그래밍이란?

- 프로그램을 '상호작용하는 독립된 객체들의 집합'으로 보는 것.
- 각각의 객체가 메시지를 주고받고 데이터를 처리한다.
- 프로그램을 유연하고 쉽게 변경할 수 있도록

# 6 마무리 결론

## 과제

- 지난 세미나에서 다룬 개념들을 간단하게나마 활용해 보는 것이 목적

# 6 마무리 결론

## 과제

- data.py와 main.py 두 파일을 만들어요.
- data.py에는:
  - activity라는 class를 구성
  - activity를 parent로 가지는 subclass들을 몇개 만드세요. e.g) 수강(강의), 동아리 등

# 6 마무리 결론

## 과제

- data.py에는:
  - student라는 class를 또 하나 만드세요. 강의나 동아리같은 class를 attribute로 가질 수 있도록 하세요. 리스트로 가져도 되고, 딕셔너리로 가져도 되고... Student라는 클래스가 뭘 가지면 좋을 지 생각해보세요.

# 6 마무리 결론

## 과제

- data.py

```python
# -*- coding: utf-8 -*-
class activity(object):
    pass


class Lecture(activity):
    def __init__(self, name, load,... ):
        ...
    def __str__(self):
        return self.name

class Dongari(activity):
    def __init__(self, ...):
        ...

class Student(object):
    def __init__(self, name, year, major):
        self.name = name
        self.year = year
        self.major = major
    def __str__(self):
        return "%s %d학 번  %s" %  (self.major, self.year, self.name)

    lectures = []
    dongari = []

    def hyoohak(self):
        ...
    def take(self, subject):
        ...
    def drop(self, subject):
        ...

```

# 6 마무리 결론

## 과제

- main.py에는:
  - 이 class들을 import해와서 간단한 상황들을 만들어보아요. main.py를 실행하면 간단한 상황들이 출력되도록…

    e.g) 준우가 해석학을 수강했습니다.
    준우가 해석학을 드랍했습니다.
    준우가 휴학을 했습니다.

# 6 마무리 결론

# 과제

- main.py

```
 1 # -*-coding:utf-8-*-
 2 from data import *
 3
 4 zoonoo = Student("박 준 우 ", 12, "전 산 학 과 ")
 5
 6 MAS241 = Lecture("Analysis I", ...)
 7 CS230 = Lecture("System Programming", ...)
 8
 9 zoonoo.take(MAS241)
10 zoonoo.take(CS230)
11
12 ...
13 ...
14 ...
15 ...
```