Python Programming for Science and Machine Learning

서울대학교 자연과학대학 물리천문학부

2018 가을학기

전산물리 (884.310) & 응용전산물리 (3342.618)

4 주차 강의노트 내용

Python

- Class (objects & instances, definition, call, namespace) – 1/N
- math functions in (built-in functions /math & random modules)

공지

- 9 월 넷째주 (9/24-27, 추석연휴) *연습반 실습없음*
  - 화상대체실습 (9/26, 8:00 pm-) [https://zoom.us/j/326917405]

- Exercise: 03 차 실습과제 (*제출없음*)

- Homework: 02 차 정규과제
  - [9/17(월) ETL 공지 - 9/28(금) 24:00, 제출]

- 중간고사 일정확인 : 중간고사 기간 전 주 [10/19 (금) 1:00 – 4:00]

파이썬 클래스 1

1) 클래스란 ?

2) 클래스의 선언

3) 클래스와 인스턴스의 이름공간

4) 클래스와 인스턴스의 관계

5) 생성자 , 소멸자 메소드

파이썬 클래스 1 노트

- Jupyter notebook on Python class 1-1
- Jupyter notebook on Python class 1-2

파이썬 클래스 2&3 노트

- Jupyter notebook on Python class 2
- Jupyter notebook on Python class 3

" 내가 그의 이름을 불러주기 전에는
그는 다만
하나의 몸짓에 지나지 않았다 .

내가 그의 이름을 불러주었을 때 ,
그는 내게로 와
꽃이 되었다 .
..."

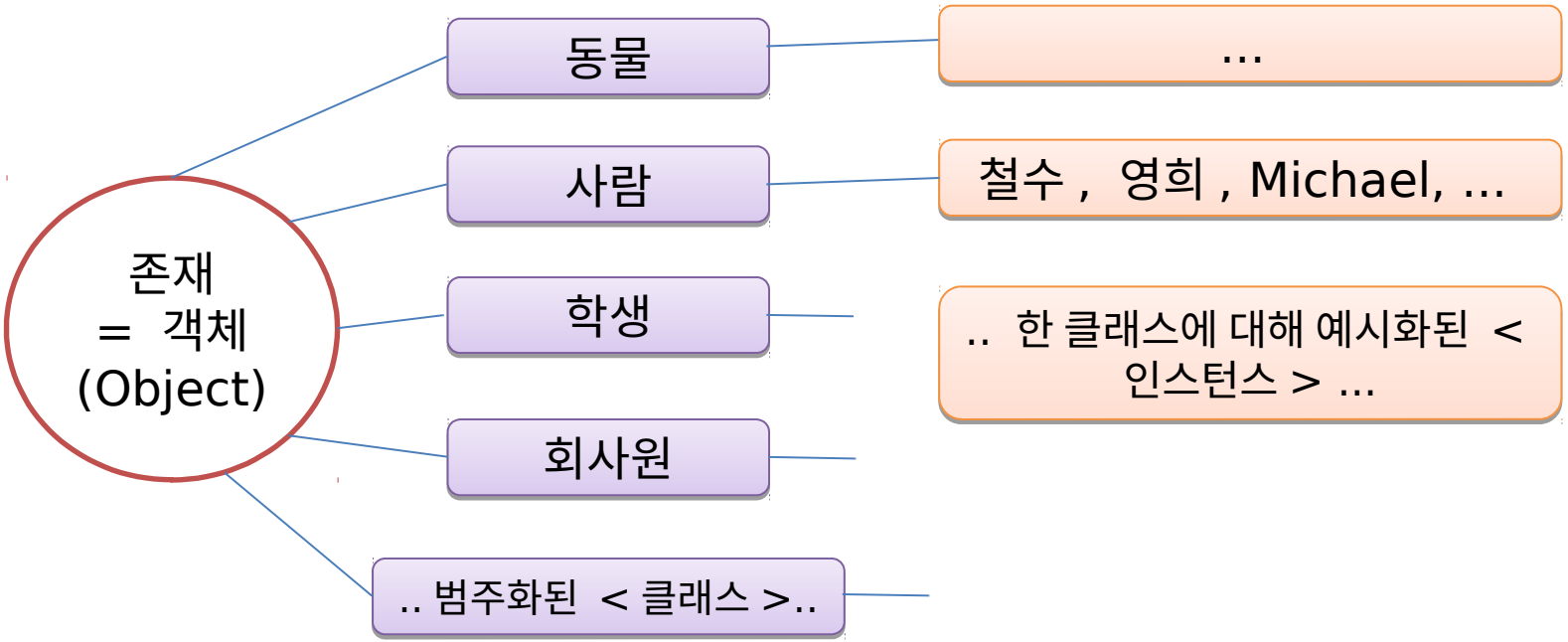
김춘수의 '꽃' 중에서

클래스란 ?

OOP(=Object Oriented Programming, 객체지향프로그래밍) 에서의 이해 :

- 객체 (object): 상태 (state/data/variable) 와 기능 (function/behavior/method) 이 정의되는 그 어떤 존재는 각각이 모두 하나의 객체이다 .
- 클래스 (class): 클래스는 그 상태정보와 기능이 보다 구체적으로 분류 및 범주화된 표현형 (representation/type) 을 말한다 .
- 인스턴스 (instance): 그 어떤 객체가 어떤 클래스 (표현형) 를 통해 실질적으로 예시화된 것을 말한다 .

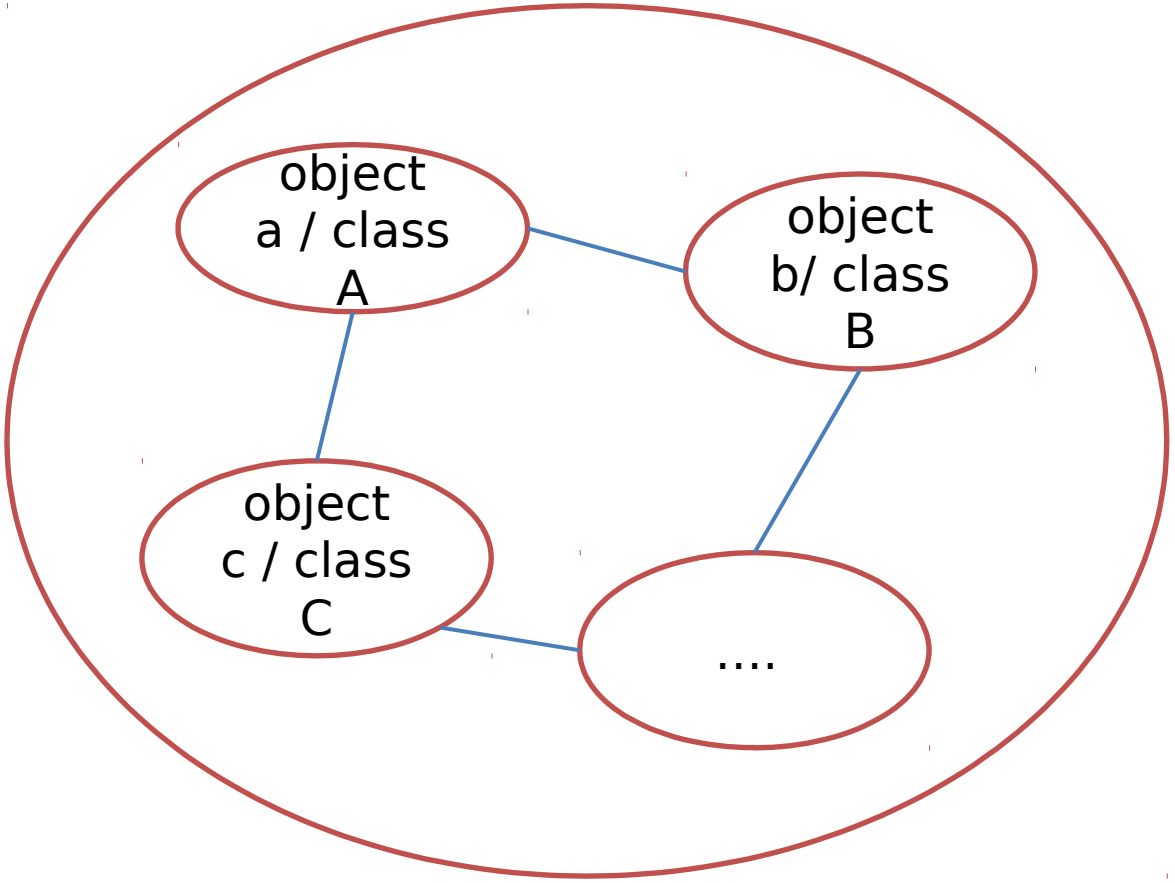
> 클래스 (class) 의 구성 = 상태정보 (state/data/variable) + 기능 (method/function)



객체지향프로그래밍

Object Oriented Programming (객체지향프로그래밍) :

- (상태 /data + 기능) 을 갖는 객체들을 기본 단위로 하여 , 다양한 클래스로 발현된 객체들의 조합으로 , 원하는 기능을 구현하는 프로그래밍 철학 .



A program as a big object

클래스

- 어떤 객체 (상태 / 속성 + 기능) 의 특징을 표현하는 구체적인 (속성 , 기능) 의 집합으로서의 클래스

예) 클래스 = ' 사람'

  - 상태 / 속성 = 멤버변수 = 키 , 성별 , 인종 , …
  - 기능 / 속성을 변화시킴 = 멤버함수 (메소드) = 걷기 , 말하기 , 생각하기 , 잠자기 , …

- 클래스에 속한 멤버함수를 **메소드 (method)** 라 부른다 .

- 한 클래스가 정의 / 선언되면 그 클래스에 대응되는 이름공간 (namespace) 가 생긴다

- 한 클래스의 구체적인 예시 = 인스턴스 , 예를 들면 어떤 사람에 해당하는 객체가 필요할 때마다 인스턴스를 생성하여 사용 (for instance ~ for example, exemplification).

- 한 인스턴스가 정의 / 선언되면 그 인스턴스는 생성이 완료된 직후 원본 클래스와 동일한 속성 데이터와 함수를 가지고 있음 .

클래스의 선언
(예제파일 Class_1_1 참조)

- 클래스 = { 멤버변수 (데이터), 메소드 (기능)}
  - 둘 중에 하나만 있어도 되고, 둘 다 없이도 선언가능
  - 선언과 동시에 클래스의 이름공간이 생성

예 1) 가장 단순한 클래스의 선언 (변수와 메소드 모두 없음)

```
>>> class Class1:   # 클래스 선언
...     """ 이것은 제 첫 클래스 입니다""" # 클래스 주석
...     pass  # pass로 아무 정의없이 마무리
...
>>> dir()
['Class1', '__builtins__', '__cached__', '__doc__', '__loader__', '__name__', '__package__', '__spec__'
, 'readline', 'rlcompleter']
>>>
>>> class Class1:   # 클래스 선언
...     """ 이것은 제 첫 클래스 입니다""" # 클래스 주석
...     pass  # pass로 아무 정의없이 마무리
...
>>> dir() # <-- 생성된 이름공간(Class1)의 확인
['Class1', '__builtins__', '__cached__', '__doc__', '__loader__', '__name__', '__package__', '__spec__'
, 'readline', 'rlcompleter']
>>> type(Class1) # <-- Class1 타입의 확인
<class 'type'>
```

- dir() 명령을 통해 전역 이름영역에 등록되어 있는 객체들을 볼 수 있음 .

클래스의 선언
(예제파일 Class_1_1 참조)

예 2) 멤버 변수와 메서드를 동시에 가지고 있는 클래스의 선언 :

```
>>> class Human:    # <-- 클래스 정의
...     Name = "default Name"  # <-- 멤버 변수
...     def Print(self):       # <-- 멤버 함수(=메소드)
...         print("My name is {0}".format(self.Name))
...
>>> h1 = Human()    # <-- Human 클래스의 한 인스턴스 생성
>>> h1.Print()      # <-- 인스턴스를 통한 Human 클래스의 메소드를 호출=>변수값 출력
My name is default Name
```


```
>>> h1.Name = "마이콜"  # <-- 인스턴스의 멤버변수 값 변경
>>> h1.Print()   # <-- 변경된 값 출력
My name is 마이콜
```

클래스의 선언 : 클래스와 인스턴스의 이름공간의 전달 (self)
(예제파일 Class_1_1 참조)

- self ?

: 메소드의 정의를 보면 첫 인자로 self 가 있는데 , 이것은 현재 인스턴스 객체를 가르킴 (~'this' in C++ / Java)

: 이를 통하여 인스턴스 객체의 이름공간에 접근

=> (정적 메소드나 클래스 메소드를 제외하고 , 메소드의 첫 인자는 인스턴스 객체가 된다 .)

이름공간과 메소드의 호출 방식 : 바운드 vs 언바운드 호출 (예제파일 Class_1_1 참조)

- 기본적으로 클래스의 메소드는 클래스의 이름공간에 선언됨
- 인스턴스가 클래스의 메소드를 호출하면 자신 이름공간에 대한 정보를 메소드에 전달 필요 .

```
>>> h1.Print()  # <-- 바운드 메소드 호출 : 메소드 정의시 첫 인자=인스턴스 객체, 호출시 자동반영
My name is 마이콜

>>> Human.Print(h1)  # <-- 언바운드 메소드 호출 : 메소드 호출시 명시적으로 인스턴스를 전달.
My name is 마이콜
```

(이름공간에 따라) 변수 / 메소드를 찾는 순서 & 멤버변수 추가 / 삭제 (삭제는 del 이용 . 예제파일 Class_1_2 참조)

- 어떤 인스턴스나 클래스를 통하여 변수에 접근하면 , 인스턴스 / 클래스의 이름 공간을 찾고 , 그 다음에 전역 영역의 순으로 변수를 찾게됨 .

인스턴스 / 클래스 영역 ==> 전역 영역

- 런타임에 각 클래스나 인스턴스의 이름공간에 멤버변수를 추가하거나 삭제할 수 있다 (<==> C++ / Java):

```
>>> class Human:  # <-- 클래스 정의
...     name = "default Name"
...
>>> h1 = Human()  # <-- 인스턴스 생성
>>> h2 = Human()  # <-- 또 다른 인스턴스 생성
>>>
>>> print("h1's name = {}".format(h1.name))  # <-- h1의 name 출력
h1's name = default Name
>>> print("h2's name = {}".format(h2.name))
h2's name = default Name
>>>
>>>
>>> h1.name = "마이콜"
>>> print("h1's name = {}".format(h1.name))
h1's name = 마이콜
>>> print("h2's name = {}".format(h2.name))
h2's name = default Name
```

(이름공간에 따라) 변수를 찾는 순서 & 멤버변수 추가 / 삭제 (삭제는 del 이용 . 예제파일 Class_1_2 참조)

- 런타임에 각 클래스나 인스턴스의 이름공간에 멤버변수를 추가하거나 삭제할 수 있다 2:

예) 클래스에 멤버변수 추가

```
>>> Human.title = "New title"
>>> Human.title = "New title"  # <-- 클래스에 새로운 멤버변수(title) 추가
>>> print("h1's title = {0}".format(h1.title))  # <-- 두 인스턴스에서 모두 접근 가능
h1's title = New title
>>> print("h2's title = {0}".format(h2.title))  # <-- 두 인스턴스에서 모두 접근 가능
h2's title = New title
>>> print("Human's title = {0}".format(Human.title)) # <-- 클래스에서도 접근 가능
Human's title = New title
```

예) 한 인스턴스에 멤버변수 추가 & 다른 인스턴스에서 에러 발생

```
>>> h1.age = 20  # <-- h1 인스턴스에만 멤버변수(age) 추가
>>> print("h1's age = {0}".format(h1.age))
h1's age = 20
>>>
>>> print('h2's age = {0}'.format(h2.age)) # <-- h2와 상위 Human클래스에age없음
  File "<stdin>", line 1
    print('h2's age = {0}'.format(h2.age)) # <-- h2와 상위 Human클래스에age없음
               ^
SyntaxError: invalid syntax
```

메소드에서 self 가 누락된 경우 (예제 파일 Class_1_1 참조)

- 코드 작성시 자주 실수하는 경우가 많음 : 클래스 메소드 내에서 인스턴스 (self) 를 통하지 않고 변수에 접근하는 경우 => 전역변수와 이름이 겹칠경우, 에러발생 안하기 때문에, 문제를 찾기 쉽지 않다.

```
>>> string = "Global String"
>>> class FString:
...     string = "Class String"  # <-- 클래스 멤버 변수
...     def Set(self, message):
...         self.string = message
...     def PrintVar(self):
...         print(string)  # self.string이 아닌 string은 전역에서 참조
...
>>> f = FString()  # <-- 인스턴스 선언
>>> f.Set("First message.")
>>> f.PrintVar()
Global String
```

클래스와 인스턴스의 관계를 확인하기 : isinstance()

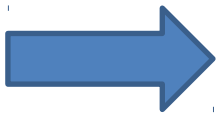
- 인스턴스가 어떤 클래스로부터 생성되었는지 확인하는 방법으로 isinstance() 내장 함수를 사용할 수 있다. 결과값은 Boolean(True/False)

```
# isinstance()

isinstance(인스턴스 객체, 클래스 객체)
```

- 클래스간의 상속관계가 있는 경우에도 자식 클래스의 인스턴스는, 부모 클래스의 인스턴스로 평가된다.

```
>>> class Human:
...     pass
...
>>> class Cat:
...     pass
...
>>> class Student(Human):
...     pass
...
>>> h, s = Human(), Student()
```



클래스와 인스턴스의 관계를 확인하기 : isinstance()

- 클래스간의 상속관계가 있는 경우에도 자식 클래스의 인스턴스는 , 부모 클래스의 인스턴스로 평가된다 .

```
>>> print("h is instance of Human {0}".format(isinstance(h, Human))) # 상속관계 판단
h is instance of Human True
>>> print("s is instance of Human {0}".format(isinstance(s, Human)))
s is instance of Human True
>>> print("h is instance of object {0}".format(isinstance(h, object)))
h is instance of object True
>>> print("h is instance of Human {0}".format(isinstance(h, Human))) # 상속관계 판단
h is instance of Human True
>>> print("s is instance of Human {0}".format(isinstance(s, Human)))
>>> print("int is instance of object {0}".format(isinstance(int, object))) #파이썬3이>
>>> print("int is instance of object {0}".format(isinstance(int, object))) #파이썬3이
후 기본자료형도 jobject에서 파생
```

생성자 (constructor) 와 소멸자 (deconstructor)
(예제파일 Class_1_2 참조)

파이썬에서도

- 인스턴스를 생성할때 , 초기화 작업을 위해 생성자 메소드 (__init__()) 를
- 메모리 해제등의 종료 작업을 위해 소멸자 메소드 (__del__()) 를

지원한다 .

- 생성자 메소드는 인스턴스가 생성될 때 자동으로 호출
- 소멸자 메소드는 인스턴스의 레퍼런스 카운터가 0 이 될 때 호출

```
class myClass:
    def __init__(self, value):  # <-- 생성자 메소드
        self.Value = value
        print("myClass is created, Value = ",value)

    def __del__(self):  # <-- 소멸자 메소드
        print("myClass is deleted")

def foo():
    a = myClass(10)  # <-- 함수 foo안에서만 인스턴스 a 존재

foo()
```

```
myClass is created, Value =  10
myClass is deleted
```

생성자 (constructor) 와 소멸자 (deconstructor)
(예제파일 Class_1_2 참조)

- 변수나 함수 앞뒤로 '__' 가 있는 경우, 이러한 이름은 특별한 용도로 미리 정의한 것.
- 생성자 메소드는, 함수를 호출할 때, 인자를 전달하는 것과 동일하게, 인스턴스 생성 시 초기화할 멤버변수의 값을 전달할 수 있다.
- myClass 클래스는 생성 시 초기값으로 1 개의 인자를 받는다. 클래스의 인스턴스는 foo() 함수 내부에 생성되어, 함수블럭을 벗어나면 자동으로 소멸. 따라서 foo() 만 호출해도 인스턴스의 생성자와 소멸자가 호출됨.

- 여분의 인스턴스 레퍼런스들이 있을 때, 이들이 모두 다 지워진 후에야, 소멸자가 호출된다.

```
>>> c = myClass(10)
myClass is created, Value = 10

>>> c_copy = c  # <-- 인스턴스 레퍼런스 하나 증가(2)
>>> del c      # <-- 레퍼런스 하나 감소(1)
>>> del c_copy  # <-- 레퍼런스 하나 감소(0), 소멸자 호출
myClass is deleted
```

파이썬의 수학 관련 내장함수와 math & random 모듈 사용하기

내장함수 1 (builtin functions → check 'dir(__builtins__)')

- Math module 을 import 하지 않고도 쓸 수 있다 .

```
>>> l = list(range(0, 10))
>>> l
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]


# sum(iterable, start) : 순회가능한 객체의 총 합계를 반환
>>> sum(l)   # 0~9까지 합
45
>>> sum(l,100)  # 0~9까지 합 + 100
145

# max(iterable) : 순회가능한 객체의 최대치
>>> max(l)
9

# min(iterable) : 순회가능한 객체의 최소치
>>> min(l)
0

# abs(x) : 절대값(x)
>>> abs(-100)
100
```

내장함수 2

```
# pow(x, y[, z]) : 제곱 연산
>>> pow(2, 10)  # 2의 10 제곱
1024
>>> pow(2, 10, 100)  # 제곱 결과를 100으로 나눈 나머지
24

# divmod(a, b) : a/b의 몫과 나머지를 튜플로 반환
>>> divmod(10, 7)
(1, 3)
>>> divmod(5.3, 2.1)
(2.0, 1.0999999999999996)


# round(x[, n]) : 인자 x의 반올림 결과. n=자리수(기본값 n=0:소수 첫째자리)
>>> round(123.456) # 소수 첫째 자리에서 반올림
123
>>> round(123.567)  # "
124
>>> round(123.456, 1) # 소수 둘째 자리에서 반올림
123.5
>>> round(123.456, -1) # 일의 자리에서 반올림
120.0
```

수학모듈을 이용한 연산 1

- **import math**
  - ceil, floor, trunc, copysign, fabs, factorial, fmod, fsum, modf, log, ...

```
import math

# ceil
>>> math.ceil(3.14)  # <-- 올림연산
4

# floor
>>> math.floor(3.14)  # <-- 내림연산
3

# trunc
>>> math.trunc(3.14)  # <-- 버림연산
3

# copysign
>>> math.copysign(6.5, -0.0)  # <-- 부호복사
-6.5

# fabs
>>> math.fabs(-6.5)  # <-- 절대값 연산
6.5
```

수학모듈을 이용한 연산 2

```
# factorial
>>> math.factorial(3.0)  # <-- factorial
6

# modf
>>> math.modf(-6.5)  # <-- 소수와 정수부분으로 분리
(-0.5, -6.0)
>>> l = [3.14, 1.23, 5.24]

# fsum
>>> math.fsum(l)  # <-- 합계
9.61

# fmod(a,b) != a%b  :
>>> math.fmod(5.5, 3)
2
>>> math.fmod(-5.5, 3)
2
>>> -5.5%3
0.5
```

수학모듈을 이용한 연산 3
지수 , 로그 , 삼각함수 연산

- math.pow(x,y)
- math.sqrt(x)
- math.exp(x)
- math.log(x[, base])

- math.sin(x), cos(x), tan(x), asin(x), acos(x), atan(x), sinh(x), cosh(x), tanh(x)
- math.degrees(x) : radian --> 60 분법
- math.radians(x) : 60 분법 --> radian

랜덤 모듈

import random

- random() : 0~1 사이
- uniform(a,b) : a 와 b 사이 float
- gauss(m, sb) : 정규분포 , m = mean, sb = standard deviation
- randrange(([start],stop[, step])) : range() 의 아이템중에서 임의로 선택반환
  - [random.randrange(20) for i in range(10)]
  - [ramdom.randrange(0, 20, 3) for i in range(5)]

- randint(a,b) : a 이상 , b 이하 정수 반환
- choice(sequence) : 시퀀스 임의의 요소 반환
- shuffle(x[, randome]) : 입력 시퀀스 x 를 임의로 섞음

3 차 (n03) 실습과제 (self-exercise)

실습목표 :

- 파이썬에서의 클래스와 객체 / 인스턴스의 개념을 익히고 클래스의 작성법을 익힌다.
- 파이썬에서 여러가지 builtins/math 모듈 /random 모듈에 정의된 함수들을 익혀보자.

실습과제 :

- 본 ppsml_note_04 (4 주차 강의) 슬라이드와 Jupyter notebook 의 여러가지 파이썬 스크립트 예제들을, 자신만의 Jupyter notebook 으로 정리하여 미리 연습 및 테스트 (변형 가능) 해본다. (제출 없음)

3 차 (n03) 정규과제 (homework)

정규과제 :

- 9/28(금) 강의시간 소개
- due 10/5(금), 파일이름형식 준수