OOP & Numpy

객체 지향 프로그래밍 (Object-Oriented Programming; OOP) 이란?

- 객체를 중심으로 프로그래밍 하는 기법
  - 객체와 그 객체들간의 상호작용 관점

- 장점
  - 어떤 하나의 '객체'에 '객체'가 필요로 하는 데이터와 기능들을 모두 넣어 **코드의 재사용성, 확장성, 가독성 개선**

클래스 (Class)

- 객체 정의 -> 클래스 설계
  - 클래스: '틀'
- 클래스가 가져야 할 속성과 기능을 정의하여야 함

```
class Account:
    # 계좌의 속성(Attribute)
    number = '0000-000-000000'
    balance = 0
    rate = 1.0

    # 계좌의 기능(Method)
    def deposit(self, money):  #입금
        self.balance += money
    def withdraw(self, money): #인출
        self.balance -= money
    def obtain_interest(self):  #이자 획득
       self.balance += self.balance*(self.rate/100)
```

인스턴스 (Instance)

- 클래스로부터 여러 개의 객체 생성 가능
  - 생성된 객체: 인스턴스
  - 각 인스턴스는 서로 다른 속성을 가질 수 있음
    - 서로 “다른” 객체이므로

```
acc1 = Account()
acc2 = Account()
acc3 = Account()

acc1.deposit(500)
acc2.deposit(1000)

print(acc1.balance)
print(acc2.balance)
print(acc3.balance)
```

메소드 (Method)

- 클래스가 가지는 기능을 클래스 내 메소드로 정의하여 사용할 수 있음

- 메소드로 정의하는 경우
  - 클래스에 변화를 일으킬 때
  - 클래스가 능동적으로 하는 행동
  - 기타

```
acc1.deposit(500)
acc2.obtain_interest()
```

생성자 (Constructor)

- 인스턴스 생성 시, 자동으로 호출되는 특수 메소드
  - 정의하지 않아도 문제되지 않았던 이유: Default 생성자 존재

- “__init__”이라는 이름을 지님

```
class Account:
    …
    # def __init__(self): Default 생성자

    def __init__(self, num='000-000-00000', amnt=0, rate=1.0):
        self.number = num
        self.balance = amnt
        self.rate = rate
```

정보 은닉 (Information Hiding)

- 사용자에게 알 필요가 없는 정보를 숨기고, 필요한 정보만을 제공하는 것
- 목적
  - 수정되어서는 안되는 속성 등을 지키기 위함

```
class Account:
    …
    def get_balance(self):
        return self.balance
    def set_balance(self, amnt):
        self.balance = amnt
```

정적 변수 (Static Variable)

- 모든 인스턴스가 공유하는 변수 (클래스 변수)

- 개설된 계좌의 총 개수를 알고 싶다면:

```
class Account:
    …
    num_acc = 0
    def __init__(…):
        …
        Account.num_acc += 1 #Account의 클래스 변수 값 변경
```

```
acc1 = Account()
acc2 = Account()
acc3 = Account()
print(acc1.num_acc)      # 3
print(Account.num_acc)   # 3
```

상속 (Inheritance)

- 기존 클래스의 필드와 메소드를 모두 물려받아 새로운 클래스를 만드는 것
  - 특수한 기능의 계좌가 있다면, 기존 계좌의 속성과 메소드를 물려받아 만드는 것이 효율적

상속 (Inheritance)

- 최소잔액계좌

```
class MinBalanceAccount(Account): #Account 클래스를 상속받음
    def __init__(self, min_balance, num='0000-000-00000', amnt=0, rate=1.0):
        Account.__init__(self, num=num, amnt=amnt, rate=rate)
        self.minimum_balance = min_balance
        self.bonus_rate = 1.0
```

다형성 (Polymorphism)

- 상속 관계 내의 다른 클래스들의 인스턴스들이 서로 다른 동작을 할 수 있도록 하는 것
  - Method Overriding
  - Operator Overloading

- 이점
  - 코드의 길이를 줄여 가독성 향상

메소드 오버라이딩 (Method Overriding)

- 상위 클래스의 메소드를 하위 클래스에서 재정의

```
class MinBalanceAccount(Account):
    …
    def withdraw(self, amnt): # 인출 기능 수정
        if self.balance - amnt < self.minimum_balance:
            print('Sorry, minimum balance must be
maintained')
        else:
            Account.withdraw(self, amnt)

    def obtain_interest(self): #보너스 이율 반영
         self.balance +=
(self.balance)*((self.rate+self.bonus_rate)/100.0)
```

연산자 오버로딩 (Operator Overloading)

- 연산자를 클래스에 맞게 재정의하여 사용하는 것
  - 연산자: +, -, *, /

```
class Account:
    …
    def __add__(self, another): #두 계좌 통합 기능
        new_acc=Account(amnt=self.balance+another.balance,
rate=self.rate)
        return new_acc
```

```
…
acc3=acc1+acc2
print(acc3.get_balance())
```

연산자 오버로딩 (Operator Overloading)

Operator	Expression	Internally
Addition	p1 + p2	p1.__add__(p2)
Subtraction	p1 - p2	p1.__sub__(p2)
Multiplication	p1 * p2	p1.__mul__(p2)
Power	p1 ** p2	p1.__pow__(p2)
Division	p1 / p2	p1.__truediv__(p2)
Floor Division	p1 // p2	p1.__floordiv__(p2)
Remainder (modulo)	p1 % p2	p1.__mod__(p2)
Bitwise Left Shift	p1 << p2	p1.__lshift__(p2)
Bitwise Right Shift	p1 >> p2	p1.__rshift__(p2)
Bitwise AND	p1 & p2	p1.__and__(p2)
Bitwise OR	p1 | p2	p1.__or__(p2)
Bitwise XOR	p1 ^ p2	p1.__xor__(p2)
Bitwise NOT	~p1	p1.__invert__()

Operator	Expression	Internally
Less than	p1 < p2	p1.__lt__(p2)
Less than or equal to	p1 <= p2	p1.__le__(p2)
Equal to	p1 == p2	p1.__eq__(p2)
Not equal to	p1 != p2	p1.__ne__(p2)
Greater than	p1 > p2	p1.__gt__(p2)
Greater than or equal to	p1 >= p2	p1.__ge__(p2)

연습문제 1

- 계좌이체 메소드 구현
  - def transfer(self, another, amnt)

```
acc1= MinBalanceAccount(min_balance=0)
acc1.set_balance(500)
acc2= MinBalanceAccount(min_balance=0)
acc2.set_balance(1000)

acc1.transfer(acc2, 100) # acc1에서 acc2로 100원
print(acc1.get_balance()) # 400
print(acc2.get_balance()) # 1100
```

Numpy 란?

- 행렬(Matrix) 및 Tensor 연산을 쉽고 빠르게 해주는 Python의 라이브러리
- 앞으로 진행할 실습에서 많이 사용될 예정
  - Tensorflow, Theano 등의 주요 NN 라이브러리에서 데이터 관련 작업 시 직접 활용하게 됨

Numpy 설치 및 설치확인 방법

- 설치
  - sudo pip install numpy
- 설치확인 (Python 내에서)
  - >> import numpy
    - 문제없이 import 되면 설치가 된 것
  - Numpy 버전 확인
    - >> import numpy
    - >> numpy.__version__

Arrays

- 배열(Array) 값을 명시하여 생성하는 방법
  - array

```
import numpy as np

a = np.array([1,2,3]) # 1차원 array [1,2,3] 생성
print(type(a))
print(a.shape)
print(a[0], a[1], a[2])
a[0] = 5
print(a)

b = np.array([[1,2,3],[4,5,6]])
print(b.shape)
print(b[0, 0], b[0, 1], b[1, 0])
```

Arrays

- 배열 값을 직접 주지 않고 생성하는 방법
  - zeros, ones, full, eye, random.random

```
import numpy as np

a = np.zeros((2,2)) # 모든 값이 0인 2x2 array 생성
print(a)

b = np.ones((1,2)) # 모든 값이 1인 1x2 array 생성
print(b)

c = np.full((2,2), 7) # 모든 값이 7인 2x2 array 생성
print(c)

d = np.eye(2) # 2x2 단위행렬(Identity matrix) 생성
print(d)

e = np.random.random((2,2)) # 0~1 랜덤 값을 가지는 2x2 array 생성
print(e)
```

Array Slicing

- 배열의 일부를 가져다 오는 방법
- 인덱스 영역에 콜론(:) 사용
  - [:2] -> 시작부터 2 앞까지 (0, 1)
  - [1:3] -> 1부터 3 앞까지 (1, 2)
  - [:] -> 모든 인덱스
  - [1:] -> 1부터 끝

```
import numpy as np

a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])

# 첫번째 행의 처음부터 2 앞까지(0,1), 두번째 행에서
# 1부터 3 앞까지(1,2)의 값들을 떠와 b에 할당.
# 따라서 b 는 다음과 같은 2x2 array:
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]

print(a[0, 1])
b[0, 0] = 77
print(a[0, 1])
# Slicing을 한다고 해서 값을 복사해가는 것이 아니므로
# 값의 변화가 동시에 일어남
```

Array Slicing

```
import numpy as np

# 2차원 배열(3x4) 생성
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])

# 두 번째 행의 값들을 모두 가져오는 두 가지 방법
row_r1 = a[1, :]   # 1차원 행렬 (4,)
row_r2 = a[1:2, :] # 2차원 행렬 (1x4) - 기존 차원 유지
print(row_r1, row_r1.shape)
print(row_r2, row_r2.shape)

# 두 번째 열의 값들을 모두 가져오는 두 가지 방법
col_r1 = a[:, 1]
col_r2 = a[:, 1:2]
print(col_r1, col_r1.shape)
print(col_r2, col_r2.shape)
```

Array Indexing

```
import numpy as np

# 2차원 배열(3x2) 생성
# [[ 1  2]
#  [ 3  4]
#  [ 5  6]]
a = np.array([[1, 2], [3, 4], [5, 6]])

# [1 4 5] 1차원 행렬(3,)을 얻는 두 가지 방법
print(a[[0, 1, 2], [0, 1, 0]])
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))

# [2 2] 1차원 행렬(2,)을 얻는 두 가지 방법
print(a[[0, 0], [1, 1]])
print(np.array([a[0, 1], a[0, 1]]))
```

Array Indexing

```
import numpy as np

# 2차원 배열(4x3) 생성
# [[ 1   2  3]
#  [ 4   5  6]
#  [ 7   8  9]
#  [ 10 11 12]]
a = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
print(a)

b = np.array([0, 2, 0, 1])

print(a[np.arange(4), b]) # [1 6 7 11]

a[np.arange(4), b] += 10

print(a) # array([[11,  2,  3],
         #        [ 4,  5, 16],
         #        [17,  8,  9],
         #        [10, 21, 12]])
```

데이터 타입

```
import numpy as np

x = np.array([1, 2])
print(x.dtype)  # prints “int64”

x = np.array([1.0, 2.0])
print(x.dtype)  # “float64”

x = np.array([1, 2], dtype=np.int64) # 데이터타입 명시
print(x.dtype)  # “int64”
```

Array 연산

```
import numpy as np

x = np.array([[1, 2], [3, 4]], dtype=np.float64)
y = np.array([[5, 6], [7, 8]], dtype=np.float64)

# Elementwise sum
# [[ 6.0  8.0]
#  [10.0 12.0]]
print(x + y)
print(np.add(x, y))

# Elementwise difference
# [[-4.0 -4.0]
#  [-4.0 -4.0]]
print(x - y)
print(np.subtract(x, y))
```

```
# Elementwise product
# [[ 5.0 12.0]
#  [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))

# Elementwise division
# [[0.2         0.33333333]
#  [0.42857143  0.5       ]]
print(x / y)
print(np.divide(x, y))

# Elementwise square root
# [[1.          1.41421356]
#  [1.73205081  2.        ]]
print(np.sqrt(x))
```

Array 연산 – Dot product

```
import numpy as np

x = np.array([[1, 2], [3, 4]])
y = np.array([[5, 6], [7, 8]])

v = np.array([9, 10])
w = np.array([11, 12])

# v · w = 219
print(v.dot(w))
print(np.dot(v, w))

# 매트릭스 / 벡터 곱; [29 67]
print(x.dot(v))
print(np.dot(x, v))

# 매트릭스 / 매트릭스 곱;
# [[19 22]
#  [43 50]]
print(x.dot(y))
print(np.dot(x, y))
```

Array 연산 - Sum

```
import numpy as np

x = np.array([[1, 2], [3, 4]])

print(np.sum(x))          # 모든 원소의 합
print(np.sum(x, axis=0))  # 각 열의 합
print(np.sum(x, axis=1))  # 각 행의 합
```

Array 연산 - Transpose

```
import numpy as np

x = np.array([[1, 2], [3, 4]])

print(x)
print(x.T)     # 전치행렬 (Transposed mat.)

v = np.array([1,2,3])
print(v)           # prints “[1 2 3]”
print(v.T)     # prints “[1 2 3]”
```

Broadcasting

```
import numpy as np

x = np.array([[1,2,3], [4,5,6], [7,8,9], [10,11,12]])
v = np.array([1,0,1])
y = np.empty_like(x) # x와 같은 shape의 빈 매트릭스 만들기

# x의 각 행에 v와 elementwise 합을 하여 y의 해당 행에 할당
for i in range(4):
    y[i, :] = x[i, :] + v

# [[ 2  2  4]
# [[ 5  5  7]
# [[ 8  8 10]
# [ 11 11 13]]
print(y)
```

Broadcasting

```
import numpy as np

x = np.array([[1,2,3], [4,5,6], [7,8,9], [10,11,12]])
v = np.array([1,0,1])
vv = np.tile(v, (4, 1)) # v 내용을 4x1의 각 행에 반복함

print(vv)

# [[ 2  2  4]
# [[ 5  5  7]
# [[ 8  8 10]
# [ 11 11 13]]
y = x + vv
print(y)
```

Broadcasting

```
import numpy as np

x = np.array([[1,2,3], [4,5,6], [7,8,9], [10,11,12]])
v = np.array([1,0,1])

# 브로드캐스팅을 사용하여 x의 각 행에 v값을 더함
y = x + v

# [[ 2  2  4]
# [[ 5  5  7]
# [[ 8  8 10]
# [ 11 11 13]]
print(y)
```

Broadcasting 활용 예

- Covariance Matrix 구할 때
  - 어떤 행렬의 각 열의 값들과 해당 열의 평균의 차를 계산하여 구함

주요 함수

- numpy.linalg.inv – 역행렬 계산 함수
- numpy.matlib.repmat – 확장 함수
- numpy.linalg.norm – Euclidean Distance 계산 함수
- numpy.conv – Covariance Matrix 계산 함수

연습문제 2

- 연립 2차 방정식 풀기
  - $\begin{cases} a+b=15 \\ 2a+b=25 \end{cases}$ 는 다음과 같이 표현가능하다.
  - $\begin{pmatrix} 1 & 1 \\ 2 & 1 \end{pmatrix}\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 15 \\ 25 \end{pmatrix}$ 그러므로 $\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 2 & 1 \end{pmatrix}^{-1}\begin{pmatrix} 15 \\ 25 \end{pmatrix}$로 계산할 수 있음.
  - 이를 계산하는 프로그램 작성하기

pickle

- 데이터를 타입 변경 없이, 파일로 저장 및 불러오기가 가능하도록 하는 모듈
  - list, dict, numpy.array, …
  - cf.
    - read, write 함수는 string으로만 입출력 가능

pickle

- pickle.dump(npair, file)
  - npair를 file로 저장하는 함수
- pickle.load(file)
  - file을 읽고 로드하는 함수

pickle

- 예)
  - 딕셔너리를 파일로 저장 & 불러오기

```
import pickle
favorite_color = { "lion": "yellow", "kitty": "red" }
pickle.dump( favorite_color, open( "save.p", "wb" ) )
```

```
import pickle
favorite_color = pickle.load( open( "save.p", "rb" ) )
```

과제 1 – set.py

- Set 클래스 구현
  - list를 이용해 Set 구현
    - Set은 list와 다르게, 같은 원소를 중복하여 가질 수 없음
  - Union, Intersection, Difference 함수 구현

```
class Set:
    def __init__(self, member=[]):
        self.member=member
        # Should be implemented
    def append(self, a):
        # Should be implemented
    def delete(self, a):
        # Should be implemented
    def union(self, s2):
        # Should be implemented
    def intersection(self, s2):
        # Should be implemented
    def difference(self, s2):
        # Should be implemented
```

```
a = Set([1, 2, 3])
b = Set([2, 3, 4])

c = a.union(b)
print(c)

d = a.difference(b)
print(d)

e = a.intersection(b)
print(e)
```

과제 1 – set.py

- Set 클래스 구현
  - 연산자 오버로딩 구현
    - +: union
    - -: difference
    - /: intersection

```
a = Set([1, 2, 3])
b = Set([2, 3, 4])

c = a + b
print(c)

d = a - b
print(d)

e = a / b
print(e)
```

과제 2 – people.py

- Person, Student, Professor 클래스 구현
  - Student, Professor는 Person을 상속받음
- Person
  - 속성: name(str), age(int), department(str)
  - 메소드: __init__, get_name
- Student
  - 속성: id(int), GPA(float), advisor(Professor)
  - 메소드: __init__, print_info, reg_advisor
- Professor
  - 속성: position(str), laboratory(str), student(list of Student)
  - 메소드: __init__, print_info, reg_student

과제 2 – people.py

```
Stu1 = Student('Kim', 30, 'Computer',
20001234, 4.5)
Stu2 = Student('Lee', 22, 'Computer',
20101234, 0.5)
prof1 = Professor('Lee', 55, 'Computer',
'Full', 'KLE')

stu1.reg_advisor(prof1)
stu2.reg_advisor(prof1)
prof1.reg_student(stu1)
prof1.reg_student(stu2)

stu1.print_info()
stu2.print_info()
prof1.print_info()
```

'제 이름은 Kim, 나이는 30, 학과는 Computer, 지도교수님은 Lee 입니다'

'제 이름은 Lee, 나이는 22, 학과는 Computer, 지도교수님은 Lee 입니다'

'제 이름은 Lee, 나이는 55, 학과는 Computer, 지도학생은 Kim, Lee 입니다'

과제 3 – and_op.py

- AND 논리 연산 기능을 위한 W, b를 선언하고, $\boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b}$ 연산을 하여 그 결과 값을 출력하라
  - $\boldsymbol{x}$가 가지는 두 값에 대한 AND 연산을 $f(\boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b})$ 로 표현가능
    - $\boldsymbol{x}$가 (1,1) 일 때, $f(\boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b}) = 1$
    - $\boldsymbol{x}$가 (0,0), (0,1), (1,0) 일 때, $f(\boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b}) = 0$

```
import numpy as np
x1 = np.array([0,0])
x2 = np.array([0,1])
x3 = np.array([1,0])
x4 = np.array([1,1])

# To be implemented
…

print(SOMETHING FOR) # f(W · x1 + b) = 0
print(SOMETHING FOR) # f(W · x2 + b) = 0
print(SOMETHING FOR) # f(W · x3 + b) = 0
print(SOMETHING FOR) # f(W · x4 + b) = 1
```

W: (2x2) array,
b: (2,) array

f: Element 중에서 max값을 가지는 인덱스 찾기
-> *np.argmax()*