

# AI Programming

## [Week 10] Practice

2024. 11. 07.



부산대학교  
PUSAN NATIONAL UNIVERSITY

- 실습 준비
- 실습 목표
- OOP
- problem.py 실습
- Gradient Descent
- gradient descent.py 실습
- 과제 안내

## HW05 (using modules) 본인 제출 파일 준비

- first-choice (n).py
- first-choice (tsp).py
- steepest ascent (n).py
- steepest ascent (tsp).py
- tsp.py
- numeric.py
- problem (문제 .txt 파일 정의된 폴더)

## HW06 파일 준비

- problem.py
- gradient descent.py

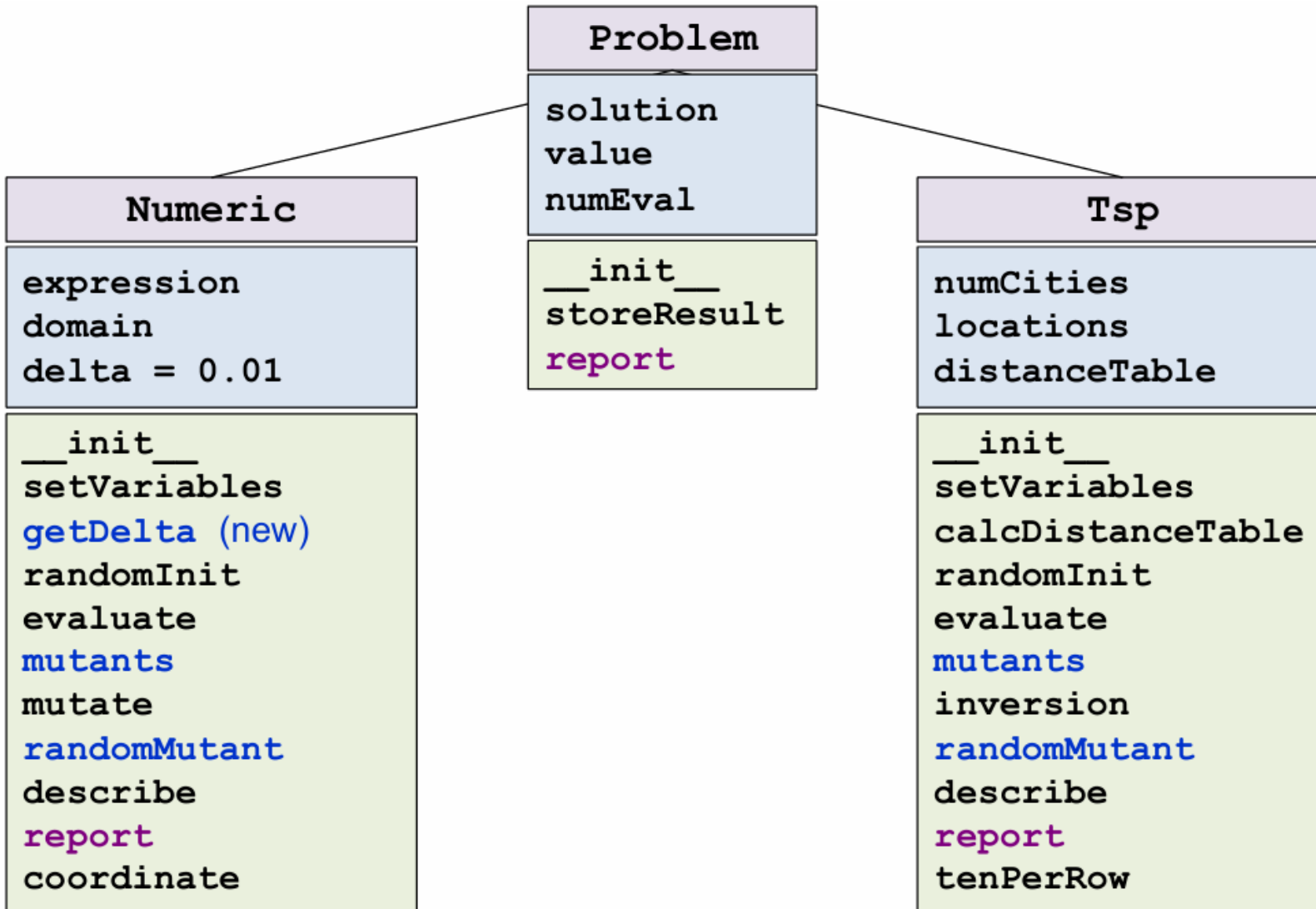
## 1. HW05 코드를 **Object-Oriented Programming (OOP)** 방식으로 리팩토링

- tsp.py, numeric.py → problem.py

## 2. **Gradient Descent** 이해 및 구현

- gradient descent.py 파일 작성

# Object-Oriented Programming OOP



2가지 분류 문제 (TSP, Numeric)를  
문제 (Problem) 클래스로 정리

추상화, 캡슐화를 통해 프로그램의  
유지 보수를 간편하게 하고  
직관적인 코드로 수정

## 주의사항!

실습 및 과제 진행 시 강의자료 '[CH09] Search Algorithms (C) - preliminary'의 Define 'Problem'의 class 챕터 참조.  
**강의 자료**와 일치하도록 구현해야 합니다.

numeric.py (절차지향 코드)를 Numeric Class를 정의하여 OOP 스타일로 구성하기  
제공된 skeleton code (problem.py)를 활용하여 구현

### numeric.py

```
def createProblem(): .....
def randomInit(p): .....
def evaluate(current, p): .....
def mutate(current, i, d, p): .....
def describeProblem(p): .....
def displayResult(solution, minimum): .....
def coordinate(solution): .....
```

### problem.py

```
class Numeric(Problem):
    def setVariables(self):
    def randomInit(self):
    def evaluate(self, current):
    def mutate(self, current, i, d):
    def describe(self):
    def report(self):
    def coordinate(self):
```

## 1. setVariables 메소드 정의 (이전 구현에서 createProblem 함수) (5분)

```
class Numeric(Problem):  
  
    def setVariables(self):  
        # Create Problem에서 Problem이 정의되어 있는 파일을 입력 받고  
        # 파일로부터 varNames, low, up을 읽어와서  
        # self._domain, self._expression 변수 설정하기  
        self._domain = [varNames, low, up]
```

Enter the file name of a function: solution/problem/Convex.txt

hint) numeric.py의 createProblem 함수를 가져와서 class에 맞게 변형

## 2. describe 메소드 정의 (이전 구현에서 describeProblem 함수) (5분)

```
class Numeric(Problem):
    def describe(self):
        ## 아래와 같은 결과가 출력될 수 있도록
        ## self._domain 변수를 활용하여 출력
```

Objective function:

$$(x_1 - 2)^2 + 5 * (x_2 - 5)^2 + 8 * (x_3 + 8)^2 + 3 * (x_4 + 1)^2 + 6 * (x_5 - 7)^2$$

Search space:

x1: (-30.0, 30.0)

x2: (-30.0, 30.0)

x3: (-30.0, 30.0)

x4: (-30.0, 30.0)

x5: (-30.0, 30.0)

hint) numeric.py의 describeProblem 함수를 가져와서 class에 맞게 변형



## 3. report 메소드 정의 (이전 구현에서 displayResult 함수) (5분)

```
class Numeric(Problem):
    def report(self):
        ## 아래와 같은 결과가 출력될 수 있도록 작성
        ## self.coordinate()도 같이 구현 (coordinate 함수 옮기기)
        print()
        print("Solution found:")
        print(self.coordinate()) # Convert list to tuple
        print("Minimum value: {0:,.3f}".format(self._value))

        # 상위클래스(Problem)의 report 함수 활용하여 code recycling
        [ fill the blank ].report(self)
```

```
Solution found:
(2.0, 5.004, -7.999, -0.998, 7.003)
Minimum value: 0.000
```

```
Total number of evaluations: 56,241
```

hint) numeric.py의 displayResult 함수를 가져와서 class에 맞게 변형

## 4. randomInit 메소드 정의 (5분)

```
class Numeric(Problem):  
    def randomInit(self):  
        # self._domain를 활용하여 유효한 범위의 랜덤 값을  
        # 문제에 정의된 변수 수 만큼 가지는  
        # list 형 변수 init 반환  
        return init
```

hint) numeric.py의 randomInit 함수를 가져와서 class에 맞게 변형

## 5. evaluate 메소드 정의 (5분)

```
class Numeric(Problem):  
    def evaluate(current, p):  
        ## evaluate 함수 호출 counting은 self._numEval 변수를 통해 수행  
        ## 현재 상태(current)에서 p를 활용하여 함수 값 계산하고 반환  
        self._numEval += 1  
        return eval(expr)
```

hint) numeric.py의 evaluate 함수를 가져와서 class에 맞게 변형

## 6. mutate 메소드 정의 (5분)

```
class Numeric(Problem):  
    def mutate(self, current):  
        ## self._domain 이용하여 current의 mutate 계산하기  
        return mutant
```

hint) numeric.py의 mutate 함수를 가져와서 class에 맞게 변형

## 7. steepest ascent (n).py 코드의 Numeric problem 관련 함수를 Numeric class로 이동 (5분)

```
def mutants(current, p):
    neighbors = []
    for i in range(len(current)): # For each variable
        mutant = mutate(current, i, DELTA, p)
        neighbors.append(mutant)
        mutant = mutate(current, i, -DELTA, p)
        neighbors.append(mutant)
    return neighbors
```

```
class Numeric(Problem):
    def mutants(self, current):
        neighbors = []
        ...
    return neighbors
```

## 8. steepest ascent.py 내용 변경 (10분)

기존 `numeric.py`와 알고리즘 내(`steepest ascent.py`)에 정의된 함수 호출해서 사용하는 코드를 `Problem.py`의 `Numeric Class`의 메소드 호출해서 사용하는 코드로 변경

```
from numeric import *
```

```
def main():
    p = createProblem()

    solution, minimum = steepestAscent(p)

    describeProblem(p)
    displaySetting()

    displayResult(solution, minimum)
```



```
from problem import Numeric
```

```
def main():
    p = _____()
    p. _____()

    steepestAscent(p)

    p. _____()
    displaySetting(p)

    p. _____()
```

## 8. steepest ascent.py 내용 변경

```
def bestOf(neighbors, p):  
    best = neighbors[0]  
    bestValue = _____(best, p)  
    for i in range(1, len(neighbors)):  
        newValue = _____(neighbors[i], p)  
        if newValue < bestValue:  
            best = neighbors[i]  
            bestValue = newValue  
    return best, bestValue
```

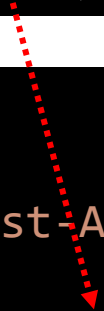
```
def steepestAscent(p):  
    current = _____(p)  
    valueC = _____(current, p)  
    while True:  
        neighbors = _____(current, p)  
        successor, valueS = bestOf(neighbors, p)  
        if valueS >= valueC:  
            break  
        else:  
            current = successor  
            valueC = valueS
```

Problem class의 storeResult 메소드를 사용해서 결과 저장하기

## 8. steepest ascent.py 내용 변경

```
def displaySetting():  
    print()  
    print("Search algorithm: Steepest-Ascent Hill Climbing")  
    print()  
    print("Mutation step size:", DELTA)
```

```
def displaySetting(p):  
    print()  
    print("Search algorithm: Steepest-Ascent Hill Climbing")  
    print()  
    print("Mutation step size:", p.getDelta())
```



Numeric Class의 self.\_delta 접근을 위한 Accessor(get) 구현



## 8. OOP style로 구현한 Steepest ascent (n).py 실행해보기

Enter the file name of a function: solution/problem/Convex.txt

Objective function:

$$(x_1 - 2)^2 + 5 * (x_2 - 5)^2 + 8 * (x_3 + 8)^2 + 3 * (x_4 + 1)^2 + 6 * (x_5 - 7)^2$$

Search space:

x1: (-30.0, 30.0)

x2: (-30.0, 30.0)

x3: (-30.0, 30.0)

x4: (-30.0, 30.0)

x5: (-30.0, 30.0)

Search algorithm: Steepest-Ascent Hill Climbing

Mutation step size: 0.01

Solution found:

(2.0, 5.004, -7.999, -0.998, 7.003)

Minimum value: 0.000

Total number of evaluations: 56,241

$$\frac{df(x)}{dx} = \lim_{dx \rightarrow 0} \frac{f(x + dx) - f(x)}{dx}$$

$$\frac{f(x + \varepsilon) - f(x)}{\varepsilon}$$

where  $\varepsilon$  is set to a small constant, say around  $10^{-4}$ .

Given a  $d$ -dimensional function  $f(\mathbf{x})$  where  $\mathbf{x} = (x_1, x_2, \dots, x_d)^T$ , its gradient  $\nabla f(\mathbf{x})$  is the following vector:

$$\left( \frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \dots, \frac{\partial f(\mathbf{x})}{\partial x_d} \right)^T$$

The  $i$ -th partial derivative in the above vector can be approximately calculated as

$$\frac{\partial f(\mathbf{x})}{\partial x_i} = \frac{f(\mathbf{x}') - f(\mathbf{x})}{\delta}$$

where  $\mathbf{x}' = (x_1, \dots, x_{i-1}, x_i + \delta, x_{i+1}, \dots, x_d)^T$ .

현재 상태에서 **gradient 반대 방향으로**  
'alpha' 비율 만큼 이동하면서 **최소값이 나올 때**  
까지 상태를 **update**

변수  $\mathbf{x}$ 에 대한 **Gradient**는  $\frac{f(\mathbf{x} + d\mathbf{x}) - f(\mathbf{x})}{d\mathbf{x}}$

## Gradient Descent를 수행하는 알고리즘 코드 (gradient descent.py 작성)

\* Gradient Descent는 Numeric Problem에서만 동작함

Gradient Descent가 동작할 수 있도록 Numeric Class에 메서드를 정의함

```
class Numeric(Problem):
    # self._alpha Accessor
    def getAlpha(self):

    # self._dx Accessor
    def getDx(self):

    # gradient를 통해 next step을 계산
    # next step이 domain 범위 이내 일 때만 next step을 반환
    def takeStep(self, x, v):

    # 주어진 변수 값들(x)이 도메인 범위 이내인지 확인하고 True, False를 반환
    def isLegal(self, x):

    # '각 변수'의 gradient를 list형으로 반환
    def gradient(self, x, v):
```

## Gradient Descent를 수행하는 알고리즘 코드 (gradient descent.py 작성)

\* Gradient Descent는 Numeric Problem에서만 동작함

Gradient Descent가 동작할 수 있도록 Numeric Class에 메서드를 정의함

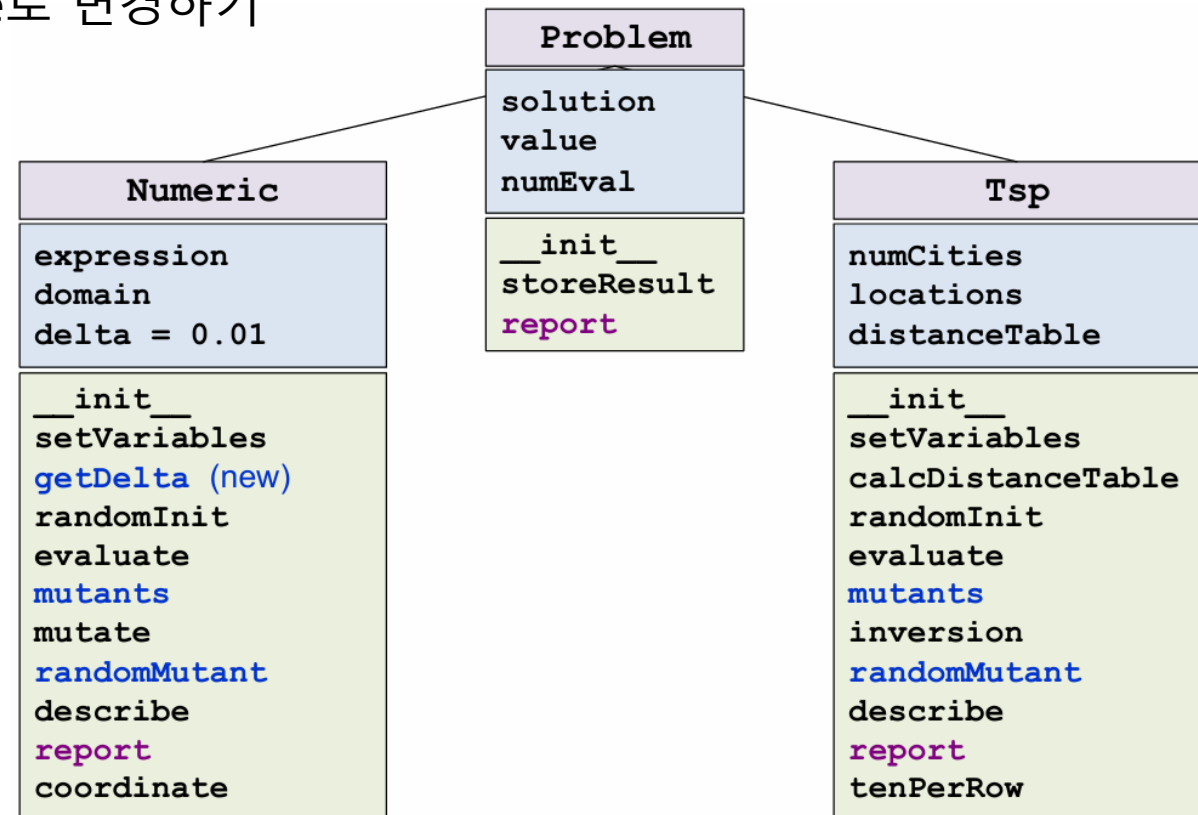
```
def gradient(self, x, v): # 'x' is a vector (list of values)
    grad = [] # Calculate partial derivatives and combine them
    for i in range(len(x)):
        xCopyH = x[:]
        xCopyH[i] += self._dx
        g = (self.evaluate(xCopyH) - v) / self._dx
        grad.append(g)
    return grad
```

1. First-choice (n).py를 OOP style로 변경하기 (Numeric Class도 같이 수정!)
2. tsp.py 코드를 Problem.py의 TSP class로 변경하기
3. Steepest Ascent (tsp).py, First-Choice (tsp).py OOP style로 변경하기
4. Gradient Descent 알고리즘을 OOP style로 구현하기

완성된 Problem.py 코드는 오른쪽 그림과 같은 변수와 메소드를 포함해야 함

강의자료 '[CH09] Search Algorithms (C) – preliminary'  
내용과 일치하게 구현해야 함

\*HW 관련 질문사항은 조교에게 쪽지



제출물:

파이썬 파일 총 6개를 **HW06\_NAME** 폴더로 묶어서 **압축**하여 제출 (.zip)

- first-choice (n).py

- first-choice (tsp).py

- steepest ascent (n).py

- steepest ascent (tsp).py

- problem.py

- gradient descent.py

리포트 제출 (.pdf)

- 본론에는 각 알고리즘을 문제 유형(tsp, numeric)마다 실행시킨 결과 총 5개의 terminal screen shot 포함

- \*gradient descent는 tsp 수행 불가능

- (문제는 자유롭게 선택하세요.)