20240531 파이썬 3차 과제

201839350 방우혁

1. Page 544 Programming 09번 문제를 구현하고 설명하라.

[문제]

Map()을 사용하여, [1,2,3,4,5,6,7,8,9,10]의 요소들을 제곱하는 프로그램을 작성하라.

# 리스트 초기화

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

# map() 함수와 lambda 함수를 사용하여 각 요소를 제곱

double\_list = list(map(lambda x: x \* x, numbers))

# 결과 출력

print(double\_list)

2. Page 544 Programming 10번 문제를 구현하고 설명하라.

[문제]

Map()과 filter()를 사용하여, [1,2,3,4,5,6,7,8,9,10] 중에서 짝수값의 제곱으로 새로운 리스트를 작성하는 프로그램을 작성하라.

# 리스트 생성

List\_num = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# filter()와 map() 함수, 그리고 lambda 함수를 사용하여 짝수 값을 제곱

Even\_list = list(map(lambda x: x \* x, filter(lambda x: x % 2 == 0, List\_num)))

# 결과 출력

print(Even\_list)

3. Page 544 Programming 11번 문제를 구현하고 설명하라.

[문제]

원을 나타내는 클래스 circle +,>,< 연산자를 중복 정의하라.

class Circle:

def \_\_init\_\_(self, radius):

self.\_\_radius = radius

@property

def radius(self):

return self.\_\_radius

def \_\_add\_\_(self, other):

return Circle(self.radius + other.radius)

def \_\_gt\_\_(self, other):

return self.radius > other.radius

def \_\_lt\_\_(self, other):

return self.radius < other.radius

def \_\_repr\_\_(self):

return f"Circle(radius={self.radius})"

4. Page 565 shape2.py를 구현하고 메서드 오버라이딩 및 다형성과 동적 바인딩에 대하여 설명하라.

# shape2

# shape 클래스 정의

class Shape:

def \_\_init\_\_(self, name):

self.name = name

def getArea(self): # 추상메소드

raise NotImplementedError("이것은 추상메소드입니다. ")

# 각 도형의 클래스 정의 -> shape 클래스를 상속

class Circle(shape):

def \_\_init\_\_(self, name, radius):

super().\_\_init\_\_(name)

self.radius = radius # 지름

def getArea(self):

return 3.141592\*self.radius\*\*2

class Rectangle(shape):

def \_\_init\_\_(self, name, width, height):

super().\_\_init\_\_(name)

self.width = width #가로

self.height = height #높이

def getArea(self):

return self.width\*self.height

# 다형성, 동적 바인딩을 활용한 예제, 프로그램 실행 시점에 메소드 결정 및 호출

shapeList = [ Circle("c1", 10), Rectangle("r1", 10, 10) ]

for s in shapeList:

print(s.getArea())

5. Page 572-573 card.py를 구현하고 설명하라.

class Card:

suitNames = ['클럽', '다이아몬드', '하트', '스페이드']

rankNames = [None, '에이스', '2', '3', '4', '5', '6', '7',

'8', '9', '10', '잭', '퀸', '킹']

def \_\_init\_\_(self, suit, rank):

self.suit = suit

self.rank = rank

def \_\_str\_\_(self):

return Card.suitNames[self.suit]+" "+\

Card.rankNames[self.rank]

class Deck:

def \_\_init\_\_(self):

self.cards = []

for suit in range(4):

for rank in range(1, 14):

card = Card(suit, rank)

self.cards.append(card)

def \_\_str\_\_(self):

lst = [str(card) for card in self.cards]

return str(lst)

deck = Deck() # 덱 객체를 생성한다.

print(deck) # 덱 객체를 출력한다. \_\_str\_\_()이 호출된다.

6. Page 574-575 person.py를 구현하고 설명하라.

class Person:

def \_\_init\_\_(self, name, number):

self.name = name

self.number = number

class Student(Person):

UNDERGRADUATE=0

POSTGRADUATE = 1

def \_\_init\_\_(self, name, number, studentType ):

super().\_\_init\_\_(name, number)

self.studentType = studentType

self.gpa=0

self.classes = []

def enrollCourse(self, course):

self.classes.append(course)

def \_\_str\_\_(self):

return "\n이름="+self.name+ "\n주민번호="+self.number+\

"\n수강과목="+ str(self.classes)+ "\n평점="+str(self.gpa)

class Teacher(Person):

def \_\_init\_\_(self, name, number):

super().\_\_init\_\_(name, number)

self.courses = []

self.salary=3000000

def assignTeaching(self, course):

self.courses.append(course)

def \_\_str\_\_(self):

return "\n이름="+self.name+ "\n주민번호="+self.number+\

"\n강의과목="+str(self.courses)+ "\n월급="+str(self.salary)

hong = Student("홍길동", "12345678", Student.UNDERGRADUATE )

hong.enrollCourse("자료구조")

print(hong)

kim = Teacher("김철수", "123456790")

kim.assignTeaching("Python")

print(kim)

7. Page 581 Programming 01번 문제를 구현하고 설명허라.

[문제]

2. 2차원 공간의 한 점 (x, y)를 나타내는 클래스 Point를 정의한다.

Point 클래스의 \_\_init\_\_() 메소드는 self, x, y를 받아서 멤버 변수에 할당한다.

\_\_str\_\_()을 정의하여 “(x,y)” 형태의 문자열을 반환한다. Point를 상속받아서 3차원 공간의

한 점(x,y,z)을 나타내는 Point3D 클래스를 정의해보자.

# point 클래스 선언

class Point:

def \_\_init\_\_(self, x, y):

self.x = x # x 좌표

self.y = y # y 좌표

def \_\_str\_\_(self):

return f"({self.x}, {self.y})" # (x, y) 형태의 문자열을 반환

class Point3D(Point):

def \_\_init\_\_(self, x, y, z): # 3d에 사용할 z좌표 선언

super().\_\_init\_\_(x, y) # 부모 클래스를 초기화한뒤, 메서드를 호출

self.z = z # z 좌표 입력

def \_\_str\_\_(self):

return f"({self.x}, {self.y}, {self.z})" # (x, y, z) 형태의 문자열 반환

8. Page 582 Programming 05번 문제를 구현하고 설명하라.

[문제]

본문에서는 animal 클래스를 상속받아서 Dog 클래스를 정의하였다. Animal 클래스를 상속받아서

Cat클래스를 정의해보자. 고양이의 속성 중에서 Name, age, breed를 저장한다. Cat 클래스로 3마리의 고양이 객체를 생성하고, 각기 다른 이름과 나이를 설정한다. Get\_oldest\_cat(\*args)함수를 작성하여, 가장 나이가 많은 고양이의 나이를 출력해보자.

class Animal:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

class Cat(Animal):

def \_\_init\_\_(self, name, age, breed):

super().\_\_init\_\_(name, age) # 부모 클래스 초기화, 메서드 호출

self.breed = breed # 품종 선언 및 초기화

def \_\_str\_\_(self):

return f"{self.name}, {self.age} years old, Breed: {self.breed}"

def get\_oldest\_cat(\*args):

# 각 고양이의 나이를 비교하여 가장 큰 나이의 개체로 교환

return max(args, key=lambda cat: cat.age)

# 세 마리의 고양이 객체 생성

cat1 = Cat("alice", 3, "1")

cat2 = Cat("bob", 4, "2")

cat3 = Cat("Con", 2, "3")

# 가장 나이가 많은 고양이를 찾음

oldest\_cat = get\_oldest\_cat(cat1, cat2, cat3)

# 가장 나이가 많은 고양이의 나이 출력

print(f"가장 나이가 많은 고양이는 {oldest\_cat.age} 살입니다.")

9. Page 582 Programming 06번 문제를 구현하고 설명하라.

[문제]

대학교에는 학과(Department), 교과목(Course), 학생(Student)가 존재한다.

학과에는 많은 교과목들이 개설될 수 있고, 하나의 교과목에는 여러 학생들이 등록할 수 있다.

Has-a 관계를 염두에 두고 이것을 다음과 같이 프로그래밍해보자.

[샘플]

Dept = Department(“컴퓨터”) # 학과 이름

Math1 = dept.add\_course(“공업수학”, 3) # 교과목 이름, 학점

Math2 = dept.add\_course(“이산수학”, 2)

Kim = Student(“Kim”, 20200001)

Kim.enroll(math1)

[실제코드]

class Course:

def \_\_init\_\_(self, name, credits):

self.name = name # 교과목 이름

self.credits = credits # 학점

self.students = [] # 교과목에 등록된 학생 리스트

def enroll(self, student):

self.students.append(student) # 학생을 교과목에 등록

print(f"{student.name} has enrolled in {self.name}.")

class Department:

def \_\_init\_\_(self, name):

self.name = name # 학과 이름

self.courses = [] # 학과에 개설된 교과목 리스트

def add\_course(self, course\_name, credits): # 새로운 교과목 함수

course = Course(course\_name, credits) # 새로운 교과목 생성

self.courses.append(course) # 교과목을 학과에 추가

return course # 생성된 교과목 객체 반환

class Student: # 학생 클래스

def \_\_init\_\_(self, name, student\_id):

self.name = name # 학생 이름

self.student\_id = student\_id # 학생 ID

self.enrolled\_courses = [] # 학생이 등록한 교과목 리스트

def enroll(self, course):

self.enrolled\_courses.append(course) # 교과목을 학생의 등록 교과목 리스트에 추가

course.enroll(self) # 교과목에 학생 등록

# 학과 생성

dept = Department("컴퓨터")

# 교과목 생성 및 학과에 추가

math1 = dept.add\_course("공업수학", 3)

math2 = dept.add\_course("이산수학", 2)

# 학생 생성

kim = Student("Kim", 20200001)

# 학생을 교과목에 등록

kim.enroll(math1)

10. Page 582 Programming 07번 문제를 구현하고 설명하라.

[문제]

하나의 앨범에는 많은 노래를 담길 수 있다. 노래를 부른 가수는 여러 노래를 소유할 수 있다.

재생 목록(Playlist)을 여러 노래를 가질 수 있다. Has-a 관계를 염두에 두고 프로그래밍해보자.

[예시]

Lee = artist(“lee’s band”)

Album = Album(“첫 번째 앨범”, lee, 2020)

Album.add\_track(“첫 번째 노래”)

Album.add\_track(“두 번째 노래”)

Playlist = Playlist(“애창곡”)

For song in album.tracks:

Playlist.add\_song(song)

[코드]

class Artist:

def \_\_init\_\_(self, name):

self.name = name # 가수 이름

self.songs = [] # 가수가 소유한 노래 리스트

def add\_song(self, song):

self.songs.append(song) # 노래를 가수의 노래 리스트에 추가

class Album:

def \_\_init\_\_(self, title, artist, year):

self.title = title # 앨범 제목

self.artist = artist # 앨범의 가수 객체

self.year = year # 앨범 발매 연도

self.tracks = [] # 앨범에 포함된 노래 리스트

def add\_track(self, title):

song = Song(title, self.artist) # 새로운 노래 객체 생성

self.tracks.append(song) # 노래를 앨범의 트랙 리스트에 추가

self.artist.add\_song(song) # 노래를 가수의 노래 리스트에 추가

class Song:

def \_\_init\_\_(self, title, artist):

self.title = title # 노래 제목

self.artist = artist # 노래의 가수 객체

def \_\_str\_\_(self):

return f"{self.title} by {self.artist.name}" # 노래의 제목과 가수 이름 반환

class Playlist:

def \_\_init\_\_(self, name):

self.name = name # 재생 목록 이름

self.songs = [] # 재생 목록에 포함된 노래 리스트

def add\_song(self, song):

self.songs.append(song) # 노래를 재생 목록에 추가

# 가수 객체 생성

lee = Artist("lee's band")

# 앨범 객체 생성 및 노래 추가

album = Album("첫 번째 앨범", lee, 2020)

album.add\_track("첫 번째 노래")

album.add\_track("두 번째 노래")

# 재생 목록 객체 생성

playlist = Playlist("애창곡")

# 앨범의 모든 노래를 재생 목록에 추가

for song in album.tracks:

playlist.add\_song(song)