Python 4

Python Object-Oriented Programming

OOP introduction

- object: attribute(속성), action(행동)을 가짐.
  - attribute: variable
  - action: method
- OOP
  - class: 설계도 (붕어빵틀)
  - instance: 구현체 (붕어빵)
- attribute 추가
  - init(객체 초기화 예약 함수), self
  - __(dunder)는 특수한 예약 함수, 변수 그리고 맨글링(함수명 변경)으로 사용
- method 구현하기
  - 기존 함수와 같으나 self를 추가해야만 class 함수로 인정됨

```
class SoccerPlayer(object):
  def __init__(self, name, position, back_number : int):
    self.name = name
    self.position = position
    self.back_number = back_number

  def change_back_number(self, new_number):
    self.back_number = new_number

  def __str__(self): # print() return
    return "Hello, my name is %s and my back number is %d" % (self.name,
self.back_number)

player1 = SoccerPlayer(name1, position1, back_number1)
player2 = SoccerPlayer(name2, position2, back_number2)
```

OOP characteristics

- 모델링에 필요한 것들
  1. inheritance (상속)
     - 부모클래스로부터 attribute, method를 물려받는 자식클래스 생성
     - class ClassChild(ClassParent)
     - super() 를 사용해 부모클래스의 속성들을 불러올 수 있음
  2. polymorphism (다형성)
     - 같은 이름의 메소드의 내부 로직을 다르게 작성
     - 파이썬 dynamic typing 특성으로, 같은 부모클래스의 상속에서 주로 발생

3. visibility (가시성)
    - 객체의 정보를 볼 수 있는 레벨 조절
    - 누구나 객체 안의 모든 변수를 볼 필요 없음, 소스 보호
    - Encapsulation , information hiding
    - “__variable”: private variable

```
class Inventory(object):
  def __init__(self):
    self.__items = []

@property    # property decorator: 숨겨진 private variable 반환
def item(self):
  return(self, __items)
```

- decorate
    - first-class objects (일급 객체): 변수나 데이터 구조에 할당 가능한 객체. parameter로 전달 가능, 리턴 값으로 사용

```
def formula(method, argument_list):
  return [method(value) for value in argument_list]
# method 에 여러 가지 function() 이 들어올 수 있음
```

    - inner function

```
def print_msg(msg):
  def printer():
    print(msg)
  printer()

print_msg("Hello, python")

def print_mss(msg):
  def printer():
    print(msg)
  return printer

function = print_msg("Hello, python")
function()
```

    - decorator function ex.1

```
def star(func):
  def inner(*args, **kwargs):
    print(args[1] * 30)
    func(*args, **kwargs)
    print(args[1] * 30)
  return inner
```

```
@star
def printer(msg):
  print(msg)

printer("Hello, python", "*")

# printer -> func
# "Hello, python", "*" -> msg
```

- decorator function ex.2

```
def generator_power(exponent):
  def wrapper(f):
    def inner(*args):
      result = f(*args)
      return exponent ** result
    return inner
  return wrapper

@generate_power(2)
def raise_two(n):
  return n ** 2

print(raise_two(7)) # 2 ** (7 ** 2)
```

Module and project

- module: 작은 프로그램 조각 (.py file)
- modules -> package

```
import {같은 디렉토리의 파일명} # 모듈 내 모든 코드가 메모리에 로드 됨
# {파일명}.{함수명}()

from {모듈명} import {함수명} # 모듈 내 특정 함수 호출

from {모듈명} import * # 모든 함수와 클래스 호출

from {폴더명} import {모듈명}

from . import {폴더명}

from game.graphic.render import render_test # 절대 참조

from ..sound.echo import echo_test # 상대 참조 (부모 디렉토리 기준)
```