Python 프로그래밍

#4. 파이썬 심화

목표

- 함수: Lambda Operator
- 클래스 (class)

익명 함수: Lambda

- 이름이 없는 함수
- `increment_lambda = lambda x: x+1`
- `(lambda x: x+1)(1)`
- Python lambda 의 경우에도 함수형 프로그래밍 언어들에서 제공하는 Higher-Order Function (고차 함수) 들을 제공하고 있다. (Lambda Operator)

익명 함수: Lambda Operator

- map
- filter
- reduce

익명 함수: Lambda - map

- map은 각각의 element에 함수를 동일하게 적용한다.

```
list(map(lambda x: x*2, [1,2,3,4,5]))
```

```
>>> [2, 4, 6, 8, 10]
```

익명 함수: Lambda - filter

- filter는 각각의 element에 함수를 적용해 True인 element만 가져온다.

```
list(filter(lambda x: x>3, [1,2,3,4,5]))
```

```
>>> [4, 5]
```

익명 함수: Lambda - reduce

- reduce는 element에다 순차적으로 함수를 적용해 최종적으로 하나의 데이터를 추출한다.

```
from functools import reduce
reduce((lambda x,y: x+y), [1,2,3,4,5])
```

```
>>> 15
```

객체 지향 프로그래밍

- Object Oriented Programming (OOP)
- 객체를 중심으로 추상화시키고 확장하기 쉽게 만들 수 있다.
- 절차 지향 프로그래밍
  - 우리가 짜왔던 방식

객체 지향 프로그래밍

- 객체
- 인스턴스
- 정보 은닉 (Information Hiding)
- 추상화 (Abstraction)
- 다형성 (Polymorphism)

객체

- 상태(변수)와 행동(메소드)을 가지는 하나의 묶음. 꾸러미.
- 우리는 이 객체에 상태를 변화시키고 일을 시킨다.
- 클래스로 만든다.
- 파이썬은 모든 것을 객체로 다룬다.
- 변수, 클래스는 물론 함수도 객체로 취급한다.

클래스

- 객체를 찍어내는 틀, 객체를 어떻게 만들까가 적힌 설계도
- 클래스를 통해 클래스의 인스턴스를 만든다!
- 클래스 안의 함수는 보통 메소드

객체와 인스턴스?

- 객체와 인스턴스는 엄밀히는 다르지만.. 같다.

```
// Java code
Person p;  // p는 객체
p = Person();  // p는 Person의 인스턴스

# python
p = Person() # p객체는 Person의 인스턴스
```

정보은닉 (Information Hiding)

- 프로그램을 사용하는 사람이 알아야 하는건 프로그램의 사용법이지 내부의 동작이나 구조가 아니다!
- 따라서 알려질 필요 없는 메소드나 함수를 숨긴다.
- 파이썬에서는 언더스코어 (_)로 정보은닉을 한다고 약속한다.

추상화 (Abstraction)

- 공통되는 특징이나 속성을 뽑아내서 재사용하기 쉽게 만든다.
- 예를 들어 삼각형 클래스, 사각형 클래스, 원 클래스가 있다.
- 이 클래스에 담긴 공통적인 속성
  - color, position, move(), rotate()
- 이들을 매번 작성해 보는 것보다 더 큰 틀을 만들자!
  - class Shape(): ...

다형성 (Polymorphism)

- 공통된 추상화 클래스에서 만들어진 인스턴스들은 같은 함수를 가진다
- 예를들어 Shape 안에 있는 move()함수는 상속받은 클래스에서 각자의 행동을 구성할 수 있다.
- Keyword: 상속 (Inheritance), 덕-타이핑 (Duck typing)

클래스 만들기

- 클래스를 만드는 여러 방법

```
class Person:
  pass

class MyClass():
  pass

class MyClass2(object):
  pass
```

클래스와 인스턴스의 namespace

- 클래스의 namespace가 인스턴스 namespace에 영향을 미친다

```
class Person():
  pass

p = Person()
Person.title = "hello~"

>>> p.title
"hello~"
```

isinstance?

- 인스턴스 객체가 어떤 클래스로부터 생성되었는지 확인

```
>>> isinstance(10, int)
True

>>> isinstance(p, Person)
True
```

생성자 (Constructor)

- 클래스로 부터 인스턴스가 생성 될 때 초기 값을 입력 받을 수 있다.
- 인스턴스가 생성 될 때 동작을 정의할 수 있다.

```
class Person():
  def __init__(self, name):
    self.name = name

p = Person("임원균")
```

연산자 중복 정의

- (Operator Overloading)
- 연산자를 재정의해서 객체를 조금 더 쉽게 다룰 수 있도록 한다
- 미리 규약된 특별한 메소드를 재정의
- 책 108p에 어떤 메소드들을 정의 할 수 있는지 나와있다.

상속 (Inheritance)

- 클래스가 가진 속성과 메소드들을 기반으로 새로운 클래스를 만들 수 있다.
- => 클래스를 재활용 함으로써 재사용성을 높인다

MRO

- Method Resolution Order
- 어떤 순서대로 메소드가 실행 될지 결정하는 순서
- Depth-first, Lift to Right