BIG DATA ANALYSIS PROGRAMMING

WEEK-03 | Python Class

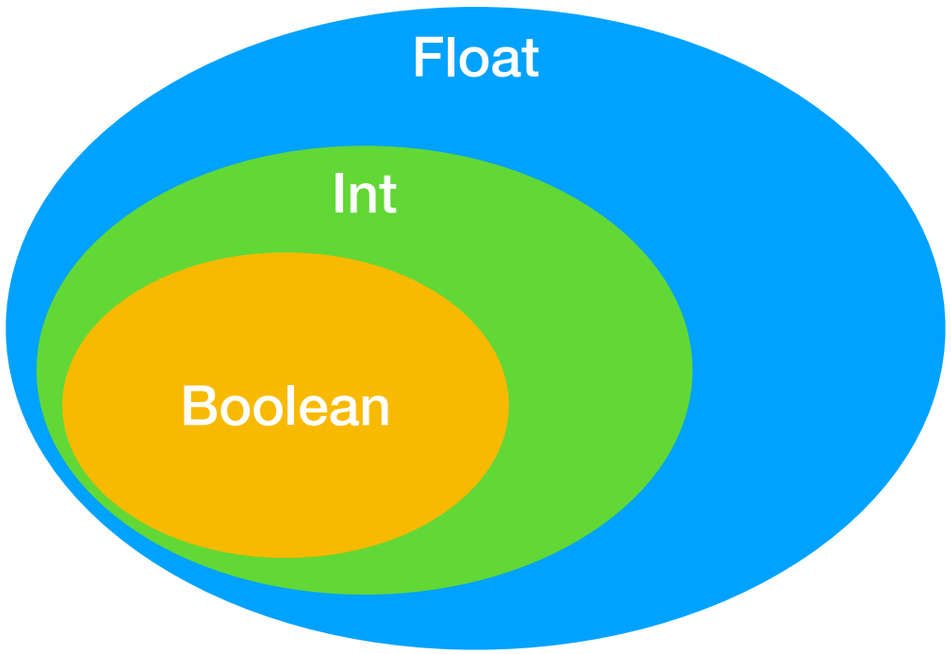
Yonsei University
Jungwon Seo

Python Recap

- Jupyter Notebook
  - 통합 개발 환경 (IDE)
  - **. + tab**: 사용가능한 method 및 variable 출력
  - **shift + tab**: method가 받을 수 있는 매개변수 확인

- Variable 변수
  - numeric
    - float, int ..
  - boolean
  - string: character의 list

h	e	l	l	o



Python Recap

- 그룹형 변수
  - List
  - Dictionary
  - Set
  - Tuple

List	Dictionary	Set	Tuple
번호기반	Key-value	수학 집합	수정 불가
순서 중요	순서 X	Unique한 값	함수 복수개의 리턴

Python Recap

- 함수
  - def 로 시작
  - 매개변수를 받을 수도 있고 안받을 수도 있다.
  - 리턴을 0개 부터 n개 까지 할 수 있다.
  - 2개 이상 부터 return 값들은 tuple로 감싸진다.
  - 매개변수를 key-value형태로 가져가면 변수의 순서를 무시 할 수 있다.
  - default값을 넣어주면 매개변수를 무시할 수도 있다.

```
def fun1(x=10):
    print(x)

def fun1(x):
    return x*2, x+2
```

```
def function1(x) :
    print(x)

function1(100)
```

Parameter

Argument

Python Class (클래스)

- Python은 multiple programming paradigms 언어
  - Procedural (절차지향)
  - Object-Oriented (객체지향)
  - Functional (함수형)

	절차지향	객체지향
장점	높은 가독성 쉬운 작성법 실행속도	코드의 재사용성 디버깅
단점	유지보수 디버깅	처리속도 설계시간
종류	C	c++, java

객체지향 프로그래밍

- Object-Oriented Programming

- 컴퓨터 프로그램을 명령어의 목록으로 보는 것이 아닌, 여러개의 독립된 단위(객체)들의 모임으로 파악하고자 함

- OOP concepts
  - 상속 : 다른 클래스(부모)의 Method 및 변수들을 상속 받을 수 있다.
  - 캡슐화 : 객체의 속성과 행위를 하나로 묶고, 그 중 일부를 외부에 감출 수 있다.
  - 추상화 : 복잡한 자료, 모듈, 시스템등으로 부터 핵심적인 개념 및 기능을 간추려 나타낼 수 있다.
  - 다형성 : 한 요소에 여러 개념을 넣을 수 있다.

- 쉽게 말해 한 변수에 대응하는 변수 (Variable)와 함수 (Method)를 묶어 놓는 방식
  - 변수: 상태
  - 함수: 행동

기존 프로그래밍 방식의 한계

- 축구선수들을 변수에 담는다면?

```
player1 = {
    "name":"손흥민",
    "team":"토트넘",
    "nationality":"대한민국",
    "salary":1000
}
player2 = {
    "name":"호날두",
    "team":"유벤투스",
    "nationality":"포르투갈"
    "salary":2000
}
```

```
player3 = {
    "name":"메시",
    "team":"바르셀로나",
    "nationality":"아르헨티나",
    "salary":3000
}
```

기존 프로그래밍 방식의 한계

- List에 다 정리

```
players = [player1, player2, player3]
```

- 이때, player1이 이적과 동시에 연봉이 상승

```
player[0]['team'] = "레알마드리드"
player[0]['salary'] = player[0]['salary']*2
```

- 제 2, 3의 개발자는 위의 코드를 보고 무슨 상황인지 추론을 해야 함

- 만약 코드만 보고 바로 이해를 할 수 있다면?

OOP Style

```
class Player:
    def __init__(self,name,team,nat,salary):
        self.name = name
        self.team = team
        self.nat = nat
        self.salary = salary
    def update_team(self, team):
        self.team = team
    def update_salary(self, salary):
        self.salary = salary

player1 = Player("손흥민", "토트넘", "대한민국", 1000)
player1.update_team("레알 마드리드")
player1.update_salary(2000)
```

OOP Terms

```
class Mammal:
    def __init__(self):
        pass
```

클래스

초기화 매써드

```
class Human(Mammal):
```

부모 클래스

변수

```
    def __init__(self,date_of_birth, name, nationality):
        self.date_of_birth=date_of_birth
        self.name = name
        self.nationality = nationality
```

매써드

```
    def think(self):
        pass
    def move(self):
        pass
```

```
jungwon = Human("900302", "서중원", "대한민국")
```

인스턴스

인스턴스화

OOP Example-1

- 게임
- 캐릭터, 건물 등을 객체화


```
class Unit:
    def __init__(self,x,y):
        self.cur_x = x
        self.cur_y = y
        pass
    def move(self,new_x,new_y,speed):
        dir_x = 1 if new_x > self.cur_x else -1
        dir_y = 1 if new_y > self.cur_y else -1
        while (self.cur_x,self.cur_y) != (new_x, new_y):
            self.cur_x += dir_x*speed
            self.cur_y += dir_y*speed
    def attack(self, target):
        pass
    def hold(self):
        pass
    def patrol(self,new_x,new_y):
        pass
```

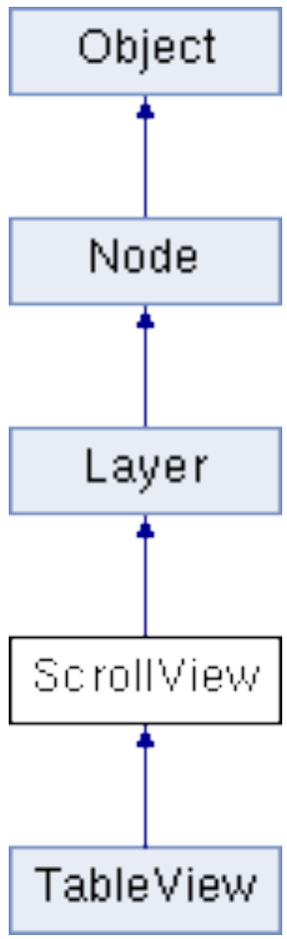
OOP Example-1

```
class Marine(Unit):
    def __init__(self,x,y):
        super().__init__(x,y)
        self.speed = 3
    def move(self, new_x, new_y):
        super().move(new_x, new_y, self.speed)
```

```
class Vulture(Unit):
    def __init__(self,x,y):
        super().__init__(x,y)
        self.speed = 10
    def move(self, new_x, new_y):
        super().move(new_x, new_y, self.speed)
```

OOP Example-2

- GUI
- 테이블, 메뉴, 버튼 등을 객체화
- onTouch, onScroll 등과 같은 사용자 액션에 대한 method 공유



* Source: https://cocos2d-x.org/reference/native-cpp/V3.0alpha0/d2/d84/classcocos2d_1_1extension_1_1_scroll_view.html

OOP Example-3

- ORM : Object-relational mapping
- 관계형 데이터베이스와 파이썬의 OOP성질을 연동
- 테이블 = 클래스, 컬럼 = 변수

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(120), unique=True, index = True)
    password = db.Column(db.String(120), nullable=False)
    created_time = db.Column(db.DateTime, default=datetime.datetime.utcnow)
    updated_time = db.Column(db.DateTime, default=datetime.datetime.utcnow)
```

클래스를 쓴다는 건

- 프로그램이 더 빠르다. (X)
- 일반적으로 안되는 알고리즘이 된다. (X)
- 설계를 잘 못 했을시, 오히려 가독성이 더 떨어질 수도 있다.
- 어떤 대상을 객체화하는 과정을 머리속으로 훈련!

E.O.D