제11강 Class

학습 목차

- Class와 Object ?
- C++과 파이썬 클래스의 비교
- 파이썬 클래스의 차별적 특성

객체(Object)와 클래스(Class)

- 컴퓨터 언어를 통해 사물을 다루는 도구

- 객체
  - 개별 사물 하나 하나 – 유일함(Unique)
  - 사물마다 특성(Property, Attribute)이 다름
  - 객체를 나타내려면? 어떤 특성들이 있는지, 각 특성의 값은?

속성, 행위

- 속성
  - attribute, property, member variable

- 행위
  - behavior, function, method

C++로 Person 클래스 만들기

- On line C++ compiler 활용
  - https://www.tutorialspoint.com/compile_cpp_online.php

- 멤버 변수
  - 이름 name, 나이 age

- 멤버 함수
  - Introduce
  - 출력: My name is daehyun, and I'm 18 years old

C++

```cpp
#include <iostream>
#include <string>
using namespace std;

class Person
{
    string name;
    int age;
public:
    Person(string given_name, int given_age) { name = given_name; age = given_age; }
    void introduce(void)
    { cout << "My name is " << name << ", and I\'m " << age << " years old."; }
};

int main()
{
    Person a_person("daehyun", 18);
    a_person.introduce();
}
```

파이썬 버전

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

    def introduce(self):
        print(f'My name is {self.name}, and I'm {self.age} years old.')



a_person = Person('daehyun', 18)
a_person.introduce()
```

클래스를 만드는 또 다른 접근법 – 파이썬 스타일.

```
class FamilyMember:
    pass

father = FamilyMember()
father

type(father) is FamilyMember
isinstance(father, FamilyMember)

father.name = 'Tom'
father.age = 60
father.address = 'Seoul'

father.name
```

동일 클래스로 만들어진 객체라도 다른 속성 가능

```
son = FamilyMember()
son.name = 'John'
son.age = 10
son.school = 'KPU'

son.school
son.address
father.school
```

클래스 변수, 클래스 속성

- 클래스 소속의 변수
- 동일한 이름은 인스턴스 변수가 없으면, 클래스 변수로 대체됨.

```
father.last_name
son.last_name

FamilyMember.last_name = 'Lee'

father.last_name
son.last_name

son.last_name = 'Kim'
son.last_name

del son.last_name
son.last_name
```

클래스 변수 초기화 및 액세스

```
class FamilyMember:
    last_name = 'Lee'

    def __init__(self, name='noname', age=0):
        self.name, self.age = name, age

    def introduce(self):
        print(f"I'm {self.name} {FamilyMember.last_name}.")
```

```
father = FamilyMember('Tom', 60)
father.introduce()
father.last_name = 'Kim'
father.introduce()
FamilyMember.last_name = 'Park'
father.last_name
del father.last_name
father.introduce()
```

클래스 변수 vs 인스턴스 변수

- 클래스 변수
  - 클래스이름.변수이름
  - 클래스에 속하는 변수
  - 동일한 이름의 인스턴스 변수가 없으면, 인스턴스 변수를 대신함.
  - 클래스 내에서 일종의 global 변수 역할을 할 수 있음.
  - C++ 의 static member 변수와 유사.

- 인스턴스 변수
  - self.변수이름
  - 동일한 이름의 클래스 변수를 덮어씀.

self 의 해석

```
FamilyMember.introduce() # error

FamilyMember.introduce(father)

father.introduce()
```

method의 실시간 추가

```
def identify(self):
    if self.age >= 18:
        print("I'm adult.")
    else:
        print("I'm child")

FamilyMember.identify = identify
father.identify()
```

method 의 실시간 변경

```
def introduce_in_korean(self):
    print(f"저는 {FamilyMember.last_name} {self.name} 입니다.")

FamilyMember.introduce = introduce_in_korean
father.introduce()
```

하지만…

```
def father_introduce():
    print("I'm father.")
father.introduce = father_introduce
father.introduce()
del father.introduce
father.introduce()
```

Instance method

- 가장 기본적인 method
- 인스턴스 변수를 주로 활용
- 클래스 변수도 액세스(Read & Write)할 수 있음.

```
class FamilyMember:
    last_name = 'Lee'

    def __init__(self, name='noname', age=0):
        self.name, self.age = name, age

    def introduce(self):
        print(f"I'm {self.name} {FamilyMember.last_name}.")
```

Class method

- 클래스 변수를 액세스(Read & Write) 할 수 있는 method
- 인스턴스 변수는 액세스 불가.

```
class FamilyMember:
    last_name = 'Lee'

    def __init__(self, name='noname', age=0):
        self.name, self.age = name, age

    def introduce(self):
        print(f"I'm {self.name} {FamilyMember.last_name}.")

    @classmethod
    def change_last_name(cls, new_last_name):
        cls.last_name = new_last_name
```

```
father = FamilyMember('Tom')
son = FamilyMember('John')
father.introduce()
son.introduce()
FamilyMember.change_last_name('Park')
father.introduce()
father.change_last_name('Kim')
father.introduce()
son.introduce()
```

Instance metho도 class 변수를 변경 가능한데….

```
class FamilyMember:
    last_name = 'Lee'

    def __init__(self, name='noname', age=0):
        self.name, self.age = name, age

    def introduce(self):
        print(f"I'm {self.name} {FamilyMember.last_name}.")

    def change_last_name(self, new_last_name):
        FamilyMember.last_name = new_last_name
```

```
father = FamilyMember('Tom')
son = FamilyMember('John')
father.introduce()
son.introduce()
father.change_last_name('Kim')
father.introduce()
son.introduce()
```

Constructor overloading

```
father = FamilyMember('Tom')
son = FamilyMember(10)
father.introduce()
son.introduce()
```

C++ 생성자 오버로딩

```
Person(string given_name, int given_age) { name = given_name; age = given_age; }
Person(string given_name) { name = given_name; age = 0; }
Person(int given_age) { name = "noname"; age = given_age; }
```

Python 생성자 오버로딩?

```
father = FamilyMember('John')
father.introduce()
I'm noname Lee and John years old.
son = FamilyMember(4)
son.introduce()
I'm noname Lee and 4 years old.
mother = FamilyMember('Suji', 50)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: __init__() takes 2 positional arguments but 3 were given
```

Class method 를 이용한 생성자 오버로딩

```
class FamilyMember:
    last_name = 'Lee'

    def __init__(self, name, age):
        self.name, self.age = name, age

    @classmethod
    def age(cls, age=0):
        return cls('noname', age)

    @classmethod
    def name(cls, name='noname'):
        return cls(name, 0)

    def introduce(self):
        print(f"I'm {self.name} {FamilyMember.last_name} and {self.age} years old.")
```

```
father = FamilyMember('John', 60)
father.introduce()
son = FamilyMember.age(10)
son.introduce()
mother = FamilyMember.name('Suji')
mother.introduce()
```

Static method

- 함수를 단순히 class 네임스페이스 안에 담아 둔 것.
- C++의 static member function과 유사.
- 유틸리티 function들을 네임스페이스 밑에서 관리하는 용도로 자주 사용.

```
class FamilyMember:
    last_name = 'Lee'

    def __init__(self, name, age):
        self.name, self.age = name, age

    @staticmethod
    def help():
        print(f'This is FamilyMember class with last name {FamilyMember.last_name}.')

    def introduce(self):
        print(f"I'm {self.name} {FamilyMember.last_name} and {self.age} years old.")
```

```
FamilyMember.help()
father = FamilyMember('john', 24)
father
father.introduce()
father.help()
```

상속(Inheritance)

```
class childclass(parentclass):
    ...
    ...
    ...
```

```python
class Figure:
    def info(self):
        print(self.name)

class Circle(Figure):
    name = 'CIRCLE'
    def __init__(self, radius=10):
        self.radius = radius
    def info(self):
        super().info()
        print(f'with radius {self.radius}.')

class Square(Figure):
    name = 'RECTANGLE'
    def __init__(self, length=10):
        self.length = length
    def info(self):
        super().info()
        print(f'with length {self.length}.')
```

```python
c = Circle(200)
c.info()
s = Square(10)
s.info()
```