L10: Object-Oriented Programming (1) – Class

Hyeryung Jang (장혜령)
(hyeryung.jang@dgu.ac.kr)
School of AI Convergence, Dongguk University

May 5, 2022, Thursday

Syllabus: Today's Topic

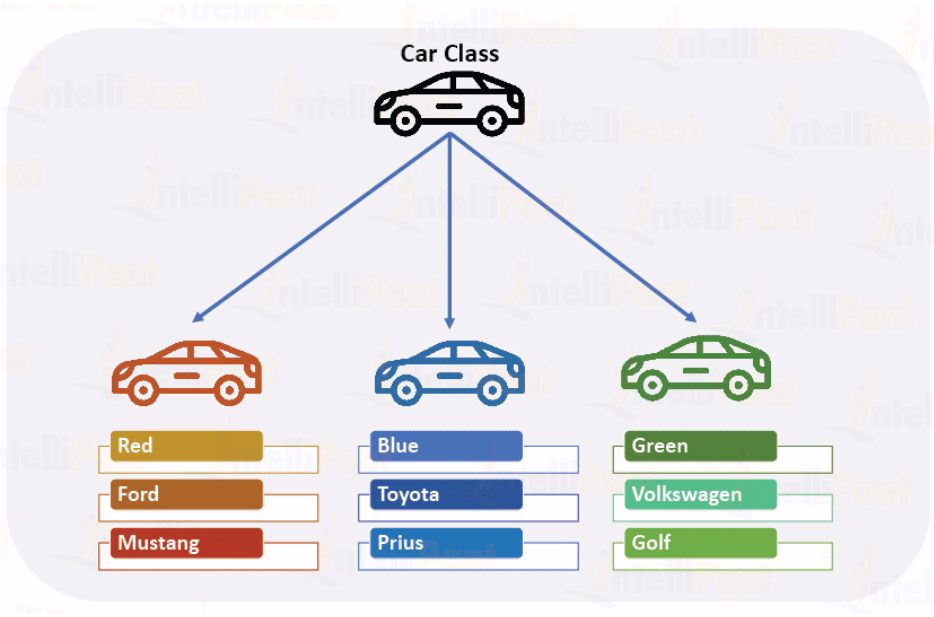
Week	Topics
1	파이썬 소개, 파이썬 개발환경 구축
2	자료형과 기본연산, 자료형 변환, 변수
3	문자열 자료형과 관련함수
4	리스트, 딕셔너리, 튜플, 집합 자료형
5	제어문 - 조건문과 반복문 - 및 예외처리
6	예외처리 및 함수: 함수의 구조와 매개변수
7	함수: 변수의 효력범위, 재귀함수, 람다함수
8	중간고사
9	사용자 입/출력, 복사
10	**객체지향 프로그래밍: 클래스, 상속 및 변수 (1)**
11	객체지향 프로그래밍: 클래스, 상속 및 변수 (2)
12	파일 입/출력 및 파일 다루기
13	정규표현식
14	모듈, 패키지, 파이썬 라이브러리
15	기말고사

What you will learn in this lecture:

- ✓ Learn about **class**: (maybe) the last major data structure in Python
- ✓ Learn about class constructor, instance/class variables and instance/class methods
- ✓ Learn how to use instance/class variables, and instance/class methods

Class and Object

Example:

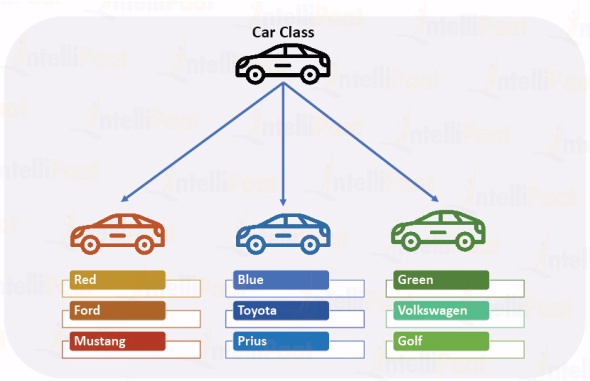


- Car class
  - Define the information and behavior that characterize a ‘car’
  - (state): Color, Price, Number of doors, Brand, Name, FuelType, MaxSpeed, …
  - (behavior): getFuel(), setSpeed(), …
  - What happens if we want to create 100+ cars in a program?

Object-Oriented Programming Language: Python

- Python is an **object-oriented programming (OOP)** language
  - 객체 지향 프로그래밍

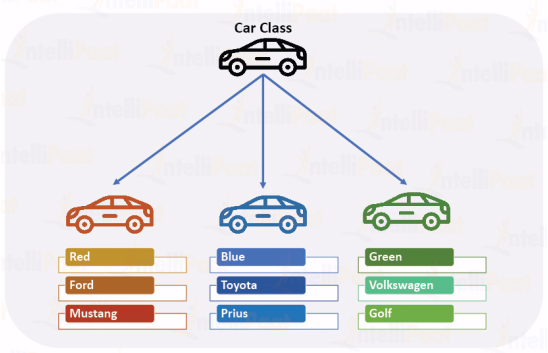
- OOP is a programming paradigm based on the concept of **"objects"**
  - The object contains both data and code
  - Data/State/Variable, known as ***attributes***
  - Code/Function/Behavior, known as ***methods***

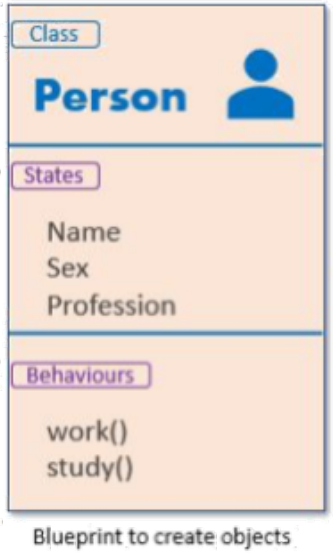


- One important aspect of OOP in Python is to create **reusable code**
  - Almost all the code is implemented using a special construct called **"class"**
  - A class is a code template for creating objects

Class and Object

- A **class (클래스)** is a blueprint, or template, for the object
  - A user-defined data structure
  - A code template for object creation
  - Can create as many objects as you want
  - 'int', 'float', … are pre-defined, built-in classes



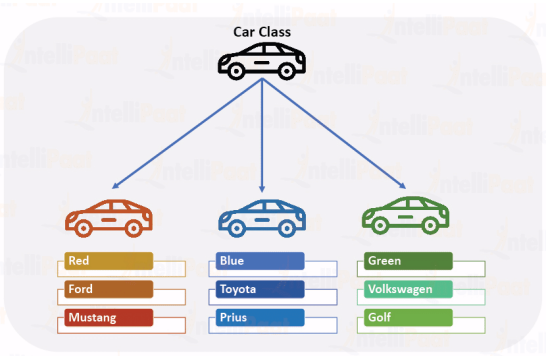


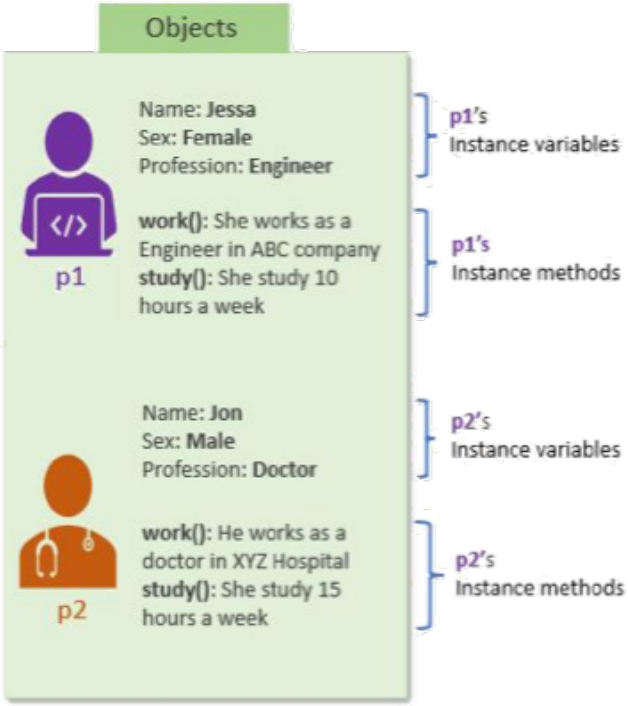
Blueprint to create objects

- ✓ A class based on the states and behavior of a Person
  - ✓ **State**: 이름, 성별, 직업
  - ✓ **Behavior**: work, study
- ✓ Using this class, we can create multiple objects (=people!) with different states and behavior

Class and Object

- An **object (객체)** is an *instance* of a class
  - A collection of **attributes** (instance variables, data, state)
  - A collection of **methods** (functions, code, behavior)
  - Every object is uniquely identified





- ✓ Object 1: Jessa
- ✓ Object 2: Jon

- ✓ Both objects are created from the same class “Person”
- ✓ But they have different states and behaviors

Create a Class in Python

- A class is defined by using the **'class' keyword**

```
class <class_name>:
    '''This is a docstring: I have created a new class'''
    <statement 1>
    <statement 2>
    ...
    <statement N>
```

```
class Student:
    pass
```

✓ Create an empty Student class

  - class_name: name of the class
  - docstring: a brief description of the class, not mandatory
  - statements: attributes and methods → class member

- The **object is created** using the **class name**
  - s1, s2 are objects of Student class

```
<object_name> = <class_name>(<arguments>)
```

```
s1 = Student()
s2 = Student()
type(s1)
```

```
__main__.Student
```

Create Object of a Class

- A **constructor (생성자)** is a method used to create and initialize an object of a class
  - **__init__()**: is defined in the class
  - Declare and initialize instance variables
  - **"self"** keyword should be the first parameter
  - "self" refers to the current object

```
class Student:
    # constructor
    def __init__(self, name, age):
        # instance variables
        self.name = name
        self.age = age
```

```
s1 = Student('Peter', 25)
print(f'Student: {s1.name}, {s1.age}')
```

```
Student: Peter, 25
```

Create Object of a Class

- Create a object of a Student class

```
<object_name> = <class_name>(<arguments>)
```

```
s1 = Student('Peter', 25)
s2 = Student('Jessica', 26)

# instance variables
print(f'Student 1: {s1.name}, {s1.age}')
print(f'Student 2: {s2.name}, {s2.age}')
```

```
Student 1: Peter, 25
Student 2: Jessica, 26
```

- Arguments are passed to the __init__() method to initialize the instance variables
- For every object, the constructor will be executed only once
- Python provides a default constructor if no constructor is explicitly defined

Instance Variables

- **Instance variables** (attributes)
  - Bound to “object”: declared inside the __init__() method
  - Not shared by objects: every object has its own, separate copy
  - Can modify the value of instance variables and assign a new value

- Access instance variables
  - Inside the instance method: using the object reference **‘self’**
  - After the instance is created: using **instance name** `<object_name>.<attribute_name>`

```
class Student:
  # constructor
  def __init__(self, name, age):
    # instance variables
    self.name = name
    self.age = age
```

```
  # instance variables
  print(f'Student 1: {s1.name}, {s1.age}')
  print(f'Student 2: {s2.name}, {s2.age}')
```

```
Student 1: Peter, 25
Student 2: Jessica, 26
```

Instance Variables

- Dynamically add/delete instance variables to an object

```
s1 = Student('Peter', 25)
print(f'Student: {s1.name}, {s1.age}')
# add new instance variable
s1.major = 'EE'
print(f'Student: {s1.name}, {s1.age}, {s1.major}')
```

```
Student: Peter, 25
Student: Peter, 25, EE
```

```
del s1.name
print(f'Student: {s1.name}, {s1.age}, {s1.major}')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-35-9e4ea3edbd2f> in <module>()
      1 del s1.name
----> 2 print(f'Student: {s1.name}, {s1.age}, {s1.major}')

AttributeError: 'Student' object has no attribute 'name'
```

Instance Variables

- List all instance variables of an object: .__dict__

```
print(s1.__dict__)
```

```
{'age': 25, 'major': 'EE'}
```

- Modify the value of instance variables

```
s1.name = 'John'
s1.age = 27

print(f'Student: {s1.name}, {s1.age}, {s1.major}')
```

```
Student: John, 27, EE
```

Instance Methods

- **Instance methods** (functions)
  - Bound to the object: defined in a class (similar to defining a regular function)
  - Used to access or modify the object data
  - Performs a set of actions on the value provided by the instance variables
  - Must have a **'self' as the first parameter** to refer to the current object

```
class Student:
  # constructor
  def __init__(self, name, age):
    # instance variables
    self.name = name
    self.age = age

  # instance method
  def display_info(self):
    print(f'Name: {self.name}, Age: {self.age}')
```

```
s1 = Student('Peter', 25)
s1.display_info()
```

```
Name: Peter, Age: 25
```

- Calling an instance method: `<object_name>.<instance_method_name>(<arguments>)`

Instance Methods

- Modify instance variables inside an instance method
- Create the instance method ‘update()’ to modify the instance variables

```
class Student:
  # constructor
  def __init__(self, name, age):
    # instance variables
    self.name = name
    self.age = age

  # instance method displaying information
  def display_info(self):
    print(f'Name: {self.name}, Age: {self.age}')

  # instance method modifying instance variable
  def update(self, new_name, new_age):
    self.name = new_name
    self.age = new_age

  # instance method adding new instance variable
  def set_major(self, major):
    self.major = major
```

```
s1 = Student('Peter', 25)
s1.display_info()
s1.update('Alice', 29)
s1.display_info()
s1.set_major('EE')
print(f'Name: {s1.name}, Age: {s1.age}, Major: {s1.major}')
```

```
Name: Peter, Age: 25
Name: Alice, Age: 29
Name: Alice, Age: 29, Major: EE
```

Class Variables

- **Class variables**
  - Bound to “class”
  - Declared inside of class, but outside of any instance method
  - Shared by all objects of a class

```
class Student:
    # class variable
    school_name = 'ABC University'

    # constructor
    def __init__(self, name, age):
        # instance variables
        self.name = name
        self.age = age
```

```
s1 = Student('Peter', 25)
print(f'Student 1 - Name: {s1.name}, Age: {s1.age}, School: {Student.school_name}')
s2 = Student('Alice', 28)
print(f'Student 2 - Name: {s2.name}, Age: {s2.age}, School: {Student.school_name}')
```

```
Student 1 - Name: Peter, Age: 25, School: ABC University
Student 2 - Name: Alice, Age: 28, School: ABC University
```

Class Variables

- Access class variables
  - Inside the constructor or instance method: using **'self'** or **class name**
  - Outside of class: using object name or class name

```
class Student:
  # class variable
  school_name = 'ABC University'

  # constructor
  def __init__(self, name, age):
    # instance variables
    self.name = name
    self.age = age
    # access class variable
    print(self.school_name, Student.school_name)

  # instance method
  def display_info(self):
    print(f'Name: {self.name}, Age: {self.age}, School name: {self.school_name}')
```

```
s1 = Student('Peter', 25)
s1.display_info()
print(s1.school_name, Student.school_name)
```

```
ABC University ABC University
Name: Peter, Age: 25, School name: ABC University
ABC University ABC University
```

Class Variables

- Modify class variables
  - Can change the value of class variable either in the class or outside of class
  - By using a class name

```
Student.school_name = 'XYZ School'
s1.display_info()
```

```
Name: Peter, Age: 25, School name: XYZ School
```

```
s2 = Student('Alice', 28)
s2.display_info()
```

```
XYZ School XYZ School
Name: Alice, Age: 28, School name: XYZ School
```

```
s1.school_name = 'AAA School'
s1.display_info()
s2.display_info()
```

```
Name: Peter, Age: 25, School name: AAA School
Name: Alice, Age: 28, School name: XYZ School
```

  - If we use an object name, a new instance variable is created for that particular object, which shadows the class variables

Class Methods

- **Class methods**
  - Bound to the class, not the object
  - Used to access or modify the class state – would apply across all the objects
  - Must have a **'cls' as the first parameter** to refer to the class
  - Must explicitly tell it is a class method: **@classmethod**

```
class Student:
  # class variable
  school_name = 'ABC University'

  # constructor
  def __init__(self, name, age):
    # instance variables
    self.name = name
    self.age = age

  @classmethod
  def change_school(cls, new_school_name):
    print(f'Before: {Student.school_name}', end='\t')
    Student.school_name = new_school_name
    print(f'After: {Student.school_name}')

  # instance method
  def display_info(self):
    print(f'Name: {self.name}, Age: {self.age}, School name: {self.school_name}')
```

```
s1 = Student('Peter', 25)
s1.display_info()
Student.change_school('XYZ School')
s1.display_info()
```

```
Name: Peter, Age: 25, School name: ABC University
Before: ABC University  After: XYZ School
Name: Peter, Age: 25, School name: XYZ School
```

Thank you!

Any Questions?

hyeryung.jang@dgu.ac.kr