codingOn X posco

K-Digital Training 스마트팩토리 개발자 입문

codingOn

Python 클래스

오늘 수업은?

- 지난시간에는 함수를 배우면서 코드의 중복을 줄이고 효과적으로 개발하는 방법에 대해서 배웠습니다.
- 오늘은 프로그래밍에서 중요한 개념인 클래스에 대해서 학습하겠습니다.
- 클래스 개념은 매우 중요합니다. 이번시간도 열심히 따라와 주시면 감사하겠습니다.

학습목표

- 객체지향 프로그래밍에 대해서 이해한다.
- 클래스에 대해서 이해한다.
- 캡슐화에 대해서 이해한다.

객체지향 프로그래밍

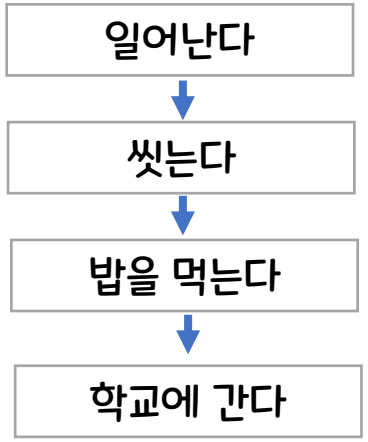
객체지향 프로그래밍

codingOn

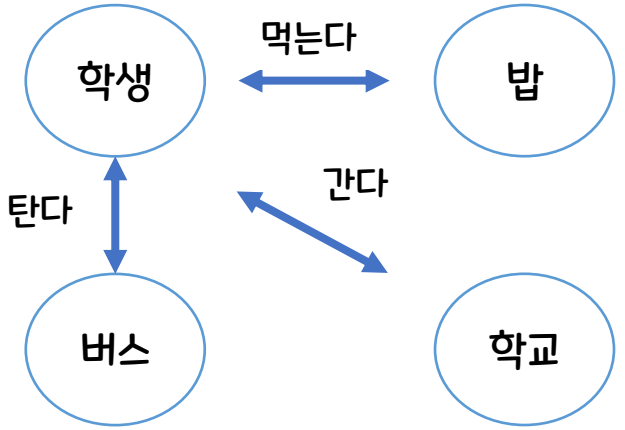
- 객체(Object)란?
  - “의사나 행위가 미치는 대상“ -> 사전적 의미
  - 구체적, 추상적 데이터 단위 (구체적- 책상, 추상적-회사)
- 객체지향 프로그래밍(Objected Oriented Programming, OOP)
  - 객체를 기반으로 하는 프로그래밍
  - 먼저 객체를 만들고 객체 사이에 일어나는 일을 구현함.

객체지향 프로그래밍

- 절차지향과 객체지향 프로그래밍 비교
- c 언어 이외 대부분의 언어는 객체지향 언어



〈절차지향 –C언어〉



〈객체지향 –Java〉

객체(복습)

- 파이썬에서 다루는 데이터는 모두 객체
- 앞으로 배울 숫자, 문자열, 리스트, 딕셔너리 등등 모든 것들
- 객체란 데이터와 그 데이터를 처리하는 메서드를 묶어서 하나로 만든 개념
- 특징
  - 가변성: 리스트, 딕셔너리, 셋과 같은 가변 객체는 값을 수정할 수 있음
  - 불변성: 숫자, 문자열, 튜플 같은 불변 객체는 값을 수정할 수 없고, 값을 변경하면 새로운 객체가 생성

클래스

클래스를 사용하는 이유

- 코드의 재사용성
- 구조화된 데이터 관리
- 즉, 클래스를 사용하면 연관된 데이터와 기능(메서드)을 하나의 단위로 묶어 관리할 수 있어 코드가 더욱 직관적이고 유지보수가 용이

클래스(class)

codingOn

- **클래스란?**

  객체에 대한 속성과 기능을 코드로 구현 한 것

  Classification(분류) 에서 유래함

  "클래스를 정의 한다"라고 함,

```
class 클래스 이름 :
    def __init__(self):
        멤버변수
    def 함수이름(self):
        return
```

- **객체의 속성과 기능**
  - 객체의 특성(property), 속성(attribute) -> **멤버 변수**
  - 객체가 하는 기능 -> **멤버 함수**

자동차 클래스

- 속성(멤버변수) : 모델명, 배기량, 색상 등..
- 기능(함수) : 시동걸기, 시동끄기, 주행하기 등..

클래스 선언

codingOn

- class 클래스 이름으로 선언(이름은 PascalCase)
- 변수 = 클래스 이름() 로 호출, 인스턴스 생성
  - 인스턴스란? 클래스로부터 생성된 구체적인 객체

```
# 클래스생성
class 클래스이름:
  # 코드


# 호출 시
변수 = 클래스이름()

예)
class TestClass:
   pass


test_instance = TestClass()
```

클래스와 객체

- 클래스
  - 새로운 형식을 정의
  - 객체를 만드는 틀 (ex. 붕어빵 틀)
- 객체
  - 클래스로부터 만들어진 것
  - 인스턴스 라고도 함 (ex. 붕어빵)
- 하나의 클래스로 여러 개의 객체를 생성함
- 각각의 객체는 서로 영향을 주지 않음

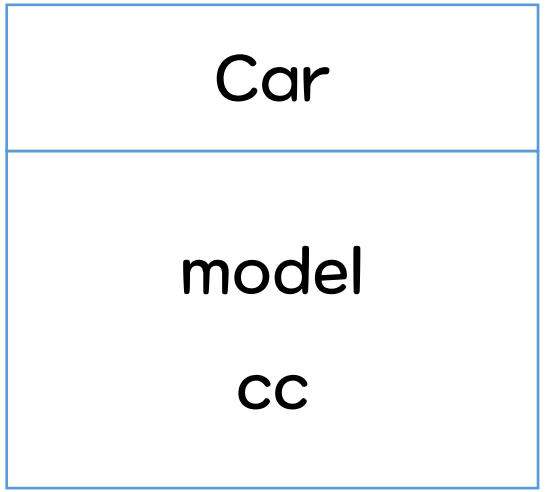
클래스(class)

codingOn

- 자동차 클래스 정의 및 사용

  객체(인스턴스) = 클래스 이름()

  객체이름.속성 -> 점연산자로 접근




```python
class Car:
    model = ""
    cc = 0

car1 = Car() # 인스턴스 생성
car1.model = '아반떼'
car1.cc = 1600

car2 = Car() # 인스턴스 생성
car2.model = 'K5'
car2.cc = 2000

print(f'모델명 : {car1.model}') # 모델명 : 아반떼
print(f'배기량 : {car1.cc}cc') # 배기량 : 1600cc
print(f'모델명 : {car2.model}') # 모델명 : K5
print(f'배기량 : {car2.cc}cc') # 배기량 : 2000cc
```

메서드

클래스 함수 (메서드, Method)

codingOn

- 메서드는 클래스 안에 정의된 함수
- 메서드 선언 시 첫 번째 매개변수로 self 필수 작성
  - self는 자기 자신을 뜻함
  - 클래스를 인스턴스화 하지 않고 호출해서 사용하는 경우에는 필요 없음
  - 호출할 때 self에 값을 직접 넘겨주지 않음 (자동 할당)
  - self는 키워드가 아닌 식별자명. 즉, 다른 이름을 사용할 수 있지만,
    현재 객체 자신을 의미하기 때문에 대부분의 개발자가 self 사용

메서드

codingOn

- 클래스에 함수 추가하기

  멤버 함수 정의 - def info()

Car
model cc
info()

```python
class Car:
    model = ""
    cc = 0

    def get_info(self):
        print(f"모델명 : {self.model}, 배기량 : {self.cc}cc")

car1 = Car() # 인스턴스 생성
car1.model = '아반떼'
car1.cc = 1600
car1.get_info()
# 모델명 : 아반떼, 배기량 : 1600cc
```

특수메서드

codingOn

▪ 생성자(constructor)

클래스를 생성할 때 호출되는 명령어 집합, 초기자라고도 한다.
생성자(초기화함수)는 def __init__(self, …)의 형태로 작성하고, 리턴값이 없다.

```
class Car:

    def __init__(self, model, year):
        self.model = model
        self.year = year

    def get_info(self):
        print(f"모델명 : {self.model}, 연식 : {self.year}년형")


car1 = Car("BMW", 2024) # 인스턴스 생성
car1.get_info()
# 모델명 : BMW, 연식 : 2024년형

car2 = Car("BENZ", 2025) # 인스턴스 생성
car2.get_info()
# 모델명 : BENZ, 연식 : 2025년형
```

생성자는 있어도 되고 없어도 됨

- 객체에 초기화가 필요 없는 경우
- 클래스를 단순히 구조화된 데이터 저장소로 사용하려는 경우

특수메서드

codingOn

- def __str__(self)

  문자열을 return하는 함수이다. 객체의 정보를 담고 있다.

```
class Car:

    def __init__(self, model, year):
        self.model = model
        self.year = year

    def __str__(self):
        return f"모델명 : {self.model}, 연식 : {self.year}년형"

car1 = Car("BMW", 2024) # 인스턴스 생성
print(car1)
# 모델명 : BMW, 연식 : 2024년형

car2 = Car("BENZ", 2025) # 인스턴스 생성
print(car2)
# 모델명 : BENZ, 연식 : 2025년형
```

객체 리스트

```
print("===승용차 리스트===")
cars = [
    Car("소나타", 2020),
    Car("쏘렌토", 2024),
    Car("싼타페", 2018)
]
# print(cars[0]) # 모델명 : 소나타, 연식 : 2020년형

# 전체출력
for car in cars:
    print(car)

# ===승용차 리스트===
# 모델명 : 소나타, 연식 : 2020년형
# 모델명 : 쏘렌토, 연식 : 2024년형
# 모델명 : 싼타페, 연식 : 2018년형
```

실습1. 사칙연산 클래스 만들기

조건)

- 인스턴스를 생성할 때 2개의 숫자를 class에 보낸다.
- add 메서드는 두 수를 더한 결과값을 반환한다.
- sub 메서드는 두 수를 뺀 결과값을 반환한다.
- mul 메서드는 두 수를 곱한 결과값을 반환한다.
- div 메서드는 두 수를 나눈 결과값을 반환한다.

변수

인스턴스 변수와 클래스 변수

- 인스턴스변수
- 특정 인스턴스에만 영향을 미치며, 객체마다 독립적으로 유지
- 각 객체별로 고유한 데이터를 저장할 때 사용
- 객체 간에 서로 영향을 미치지 않음

인스턴스 변수와 클래스 변수

- 클래스변수
- 해당 클래스를 사용하는 모두에게 공용으로 사용되는 변수
- 값을 공유하고 유지하는 특성을 가짐
- 클래스 변수의 값을 변경하면, 해당 클래스의 모든 인스턴스에서 값이 변경

인스턴스 변수와 클래스 변수

codingOn

```
class Dog:
    kind = "진돗개"  # 클래스 변수

    def __init__(self, name):
        self.name = name  # 인스턴스 변수

# 객체(인스턴스) 생성
dog1 = Dog("백구")
dog2 = Dog("검둥이")

print(dog1.name) # 백구
print(dog2.name) # 검둥이

# 클래스 변수에 접근
print(dog1.kind) # 진돗개
print(dog2.kind) # 진돗개
print(Dog.kind)  # 진돗개
```

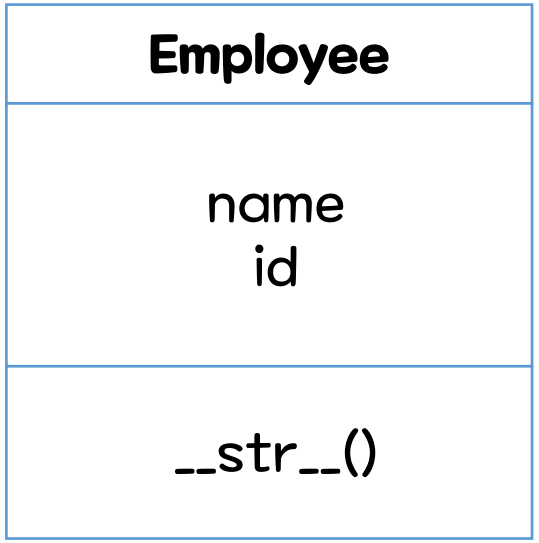
인스턴스 변수와 클래스 변수

codingOn

```python
class Example:
    shared = "공유 변수"  # 클래스 변수

    def __init__(self, name):
        self.name = name  # 인스턴스 변수

e1 = Example("A")
e2 = Example("B")

# 클래스 변수 변경
Example.shared = "변경된 공유 변수"

print(e1.shared)  # 변경된 공유 변수
print(e2.shared)  # 변경된 공유 변수

# 인스턴스 변수 변경
e1.name = "C"
print(e1.name)  # C
print(e2.name)  # B
```

사번 자동부여 클래스

- 사번 자동 발급하기
  - 멤버 변수 : 사번(id), 사원이름(name),
  - 클래스 변수 : serial_num



사번 자동부여 클래스

codingOn

- 사번 자동 발급

```
class Employee:
    serial_num = 1000  # 기본값 (클래스 변수)

    def __init__(self, name):
        Employee.serial_num += 1  # serial_num을 1 증가
        self.id = Employee.serial_num  # id에 serial_num 저장
        self.name = name

    def __str__(self):
        return f"사번 : {self.id}, 이름 : {self.name}"
```

사번 자동부여 클래스

codingOn

- serial_num이 인스턴스 변수인 경우

```
def __init__(self, name):
    self.serial_num = 1000  # 인스턴스 변수
    self.serial_num += 1
    self.id = self.serial_num  # id에 serial_num 저장
    self.name = name
```

실습2. Supermarket 클래스

codingOn

- 클래스를 선언할 때, 인자로 location, name, product, customer 받기

(location : 위치, name : 가게 이름, product : 파는 물건, customer : 고객의 수)

- print_location() : 위치를 출력하는 함수
- change_category() : 받은 인자로 파는 물건 바꾸는 함수
- show_list() : 현재 파는 물건 출력
- enter_customer() : 손님의 수를 1씩 늘리는 함수
- show_info() : 가게 이름, 위치, 파는 물건, 손님 수 모두 출력

```
위치: 마포구 염리동
상품: 음료
위치: 마포구 염리동, 이름: 마켓온, 상품: 음료, 고객수: 12
```

캡슐화

캡슐화

- 캡슐화(Encapsulation)
- 캡슐화는 데이터를 하나의 단위로 묶고, 외부로부터 직접 접근을 제한
- 필요한 경우 메서드를 통해 데이터를 제어하도록 하는 프로그래밍 기법
- 캡슐화는 데이터 보호와 모듈화를 목적
- 파이썬에서는 변수의 이름 앞에 밑줄(_ 또는 __)을 붙여 접근 제한을 표현

접근제어

codingOn

파이썬은 기본이 public임!

파이썬에서는 엄격한 정보 은닉이 없지만, 접근 제한자를 사용하여 이를 표현

Public: 어디서나 접근 가능

Private: 해당 클래스 내에서만 접근 가능

해당 멤버 앞에 __(double underscore)를 붙여서 표시

Protected: 보통은 해당 클래스 & 하위 클래스 내에서만 접근 가능 이라는 의미

파이썬에서는 해당 멤버의 앞에 _(single underscore) 를 붙여서 표시

즉, 파이썬은 실제 제약되지는 않고 일종의 경고 표시로 사용됨

정보은닉(Information Hiding)

codingOn

- 정보은닉 : 캡슐화로 인해 달성되는 결과
- **함수 생성 : get + 변수 이름(), set + 변수이름()**
- set + 변수이름() : 정보를 설정할때
- get + 변수이름() : 정보를 가져올때

정보은닉(Information Hiding)

codingOn

```
class Person:
    def __init__(self):
        self._name = ""  # 멤버 변수 초기화
        self._age = 0

    def setname(self, name):  # _name에 접근할 setname() 함수
        self._name = name

    def getname(self):  # _name에 접근할 getname() 함수
        return self._name

    def setage(self, age):  # _age에 접근할 setage() 함수
        self._age = age

    def getage(self):  # _age에 접근할 getage() 함수
        return self._age
```

```
p1 = Person()
p1.setname("흥부")
p1.setage(35)
print("이름 : ", p1.getname())
print("나이 : ", p1.getage())

p2 = Person()
p2.setname("놀부")
p2.setage(38)
print("이름 : ", p2.getname())
print("나이 : ", p2.getage())
```

실습3. 건강상태 클래스 만들기

- 운동을 하면 체력이 1증가하고, 술을 마시면 체력이 1감소함
- 건강 상태 : hp로 설정 , hp의 범위 : 1 ~ 100

출력예시

```
5시간 운동하다
술을 2잔 마시다
나몸짱 - hp: 98
======================
1시간 운동하다
술을 12잔 마시다
나약해 - hp: 0
```

나몸짱과 나약해는 사람이름

getter, setter

- 파이썬에서는 @property 라는 데코레이터를 활용하여 정의할 수 있음
  - 데코레이터? 기존 함수나 메서드의 동작을 수정하거나 확장할 때 사용
- getter함수에는 @property
- setter함수에는 @getternname.setter

getter, setter

codingOn

```python
class Person:
    def __init__(self, name, age):
        self.__name = name  # 비공개 속성
        self.__age = age    # 비공개 속성

    # Getter
    @property
    def name(self):
        return self.__name

    # Setter
    @name.setter
    def name(self, value):
        self.__name = value

    # Getter
    @property
    def age(self):
        return self.__age

    # Setter
    @age.setter
    def age(self, value):
        self.__age = value
```

```python
person = Person("홍길동", 30)
print(person.name)  # 홍길동
print(person.age)   # 30

person.name = "김철수"  # Setter
person.age = 35        # Setter
print(person.name)  # 김철수
print(person.age)   # 35
```

다음 수업은?

- 이번수업은 앞으로 파이썬 클래스 기본에 대해서 알아보았습니다.
- 다음시간에는 클래스의 상속, 다형성 등에 대해서 공부하겠습니다.

복.습.철.저

수고하셨습니다