객체지향 프로그래밍 (Object Oriented Programming)

프로그램을 구성하는 변수와 함수들에서 서로 연관성있는 것 끼리 묶어서 모듈화하는 개발하는 언어들을 객체지향프로그래밍 언어라고 한다.

Instance(객체)

- 연관성 있는 값들과 그 값들을 처리하는 함수(메소드)들을 묶어서 가지고 있는 것(값).
- 객체의 구성요소
  - 속성(Attribute)
    - 객체의 데이터/상태로 객체를 구성하는 값들.
  - 메소드(method)
    - 객체가 제공하는 기능으로 주로 Attribute들을 처리한다.

Class(클래스) 정의

- class란: 객체의 설계도
  - 동일한 형태의 객체들이 가져야 하는 Attribute와 Method들을 정의 한 것
    - 클래스를 정의할 때 어떤 속성과 메소드를 가지는지 먼저 설계해야 한다.
  - 클래스로 부터 객체(instance)를 생성한 뒤 사용한다.

```
class 클래스이름:  #선언부
    #클래스 구현
    #메소드들을 정의
```

- 클래스 이름의 관례: 파스칼 표기법-각 단어의 첫글자는 대문자 나머진 소문자로 정의한다.
  - ex) Person, Student, HighSchoolStudent

클래스로부터 객체(Instance) 생성

- 클래스는 데이터 타입 instance는 값이다.

```
변수 = 클래스이름()
```

In [ ]:

Attribute(속성)

- attribute는 객체의 데이터, 객체가 가지는 값, 객체의 상태

객체에 속성을 추가, 조회

- 객체의 속성 추가/값 변경

1. Initializer(생성자)를 통한 추가
2. 객체.속성명 = 값 (추가/변경)
3. 메소드를 통한 추가/변경
   - 1(Initializer)은 초기화할 때. 2, 3은 속성값을 변경할 때 적용.

- 속성 값 조회
  - 객체.속성명
- **객체.__dict__**
  - 객체가 가지고 있는 Attribute들을 dictionary로 반환한다.

In [ ]:

생성자(Initializer)

- 객체를 생성할 때 호출되는 특수메소드로 attribute들 초기화에 하는 코드를 구현한다.
  - Inializer를 이용해 초기화하는 Attribute들이 그 클래스의 객체들이 가져야 하는 공통 Attribute가 된다.
- 구문

```
def __init__(self [,매개변수들 선언]):  #[ ] 옵션.
    # 구현 -> attribute(instance변수) 초기화
    self.속성명 = 값
```

> 변수 초기화: 처음 변수 만들어서 처음 값 대입.

self parameter

- 메소드는 반드시 한개 이상의 parameter를 선언해야 하고 그 첫번째 parameter를 말한다.
- 메소드 호출시 그 메소드를 소유한 instance가 self parameter에 할당된다.
- Initializer의 self
  - 현재 만들어 지고 있는 객체를 받는다.
- 메소드의 self
  - 메소드를 소유한 객체를 받는다.
- Caller에서 생성자/메소드에 전달된 argument들을 받을 parameter는 두번째 변수부터 선언한다.

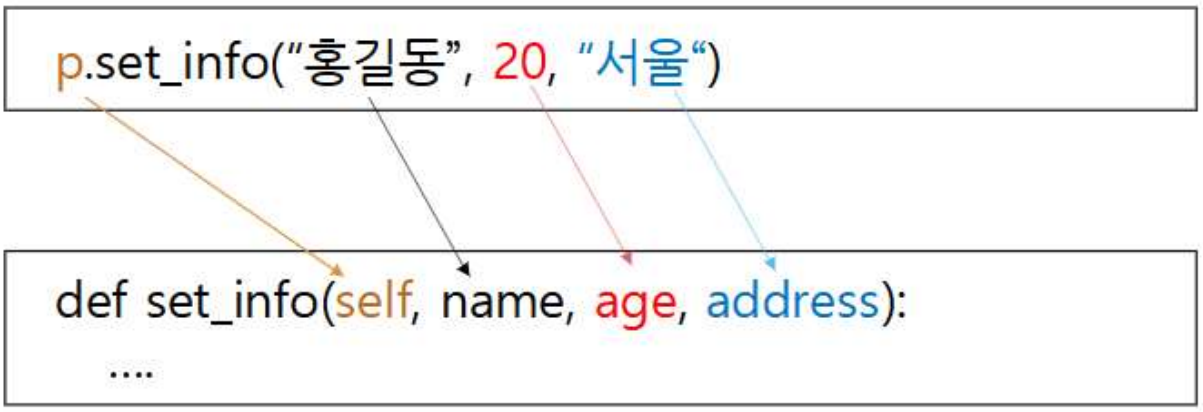
In [ ]:

Instance 메소드(method)

- 객체가 제공하는 기능
- 객체의 attribute 값을 처리하는 기능을 구현한다.
- 구문

```
def 이름(self [, 매개변수들 선언]):
    # 구현
    # attribute 사용(조회/대입)
    self.attribute
```

- self (첫번째 매개변수)
  - 메소드를 소유한 객체를 받는 변수
  - 호출할 때 전달하는 argument를 받는 매개변수는 두번째 부터 선언한다.



- **메소드 호출**

In [ ]:

정보 은닉 (Information Hiding)

- Attribute의 값을 caller(객체 외부)가 마음대로 바꾸지 못하게 하기 위해 직접 호출을 막고 setter/getter 메소드를 통해 값을 변경/조회 하도록 한다.
  - 데이터 보호가 주목적이다.
  - 변경 메소드에 Attribube 변경 조건을 넣어 업무 규칙에 맞는 값들만 변경되도록 처리한다.
  - **setter**
    - Attribute의 값을 변경하는 메소드. 관례상 set 으로 시작
  - **getter**
    - Attribute의 값을 조회하는 메소드. 관례상 get 으로 시작
- Attribute 직접 호출 막기
  - Attribute의 이름을 __(double underscore)로 시작한다. (__로 끝나면 안된다.)
  - 같은 클래스에서는 선언한 이름으로 사용가능하지만 외부에서는 그 이름으로 호출할 수 없게 된다.

In [ ]:

property함수를 사용

- 은닉된 instance 변수의 값을 사용할 때 getter/setter대신 변수를 사용하는 방식으로 호출할 수 있도록 한다.
- 구현
  1. getter/setter 메소드를 만든다.
  2. 변수 = property(getter, setter) 를 등록한다.
  3. 호출

    - 값조회: 변수를 사용 => getter가 호출 된다.
    - 값변경: 변수 = 변경할 값 => setter가 호출 된다.

In [ ]:

데코레이터(decorator)를 이용해 property 지정.

- setter/getter 구현 + property()를 이용해 변수 등록 하는 것을 더 간단하게 구현하는 방식
- setter/getter 메소드이름을 변수처럼 지정. (보통은 같은 이름으로 지정)
- getter메소드: @property 데코레이터를 선언
- setter메소드: @getter메소드이름.setter 데코레이터를 선언.
    - 반드시 getter 메소드를 먼저 정의한다.
    - setter메소드 이름은 getter와 동일해야 한다.
- getter/setter의 이름을 Attribute 변수처럼 사용한다.
- 주의: getter/setter 메소드를 직접 호출 할 수 없다. 변수형식으로만 호출가능하다.

In [ ]:

TODO

- 제품 클래스 구현
- 속성 : 제품ID:str 제품이름: str, 제품가격:int, 제조사이름:str
- 정보은닉에 맞춰서 작성. 값을 대입/조회 하는 것은 변수처리 방식을 할 수 있도록 구현.
- 메소드: 전체 정보를 출력하는 메소드

메소드 : setter-4개, getter-4개. 전체정보 출력하는 메소드-1개

In [ ]:

상속 (Inheritance)

- 기존 클래스를 확장하여 새로운 클래스를 구현한다.
    - 생성된 객체(instance)가 기존 클래스에 정의된 Attribute나 method를 사용할 수있고 그 외의 추가적인 member들을 가질 수 있는 클래스를 구현하는 방법.
- **기반(Base) 클래스, 상위(Super) 클래스, 부모(Parent) 클래스**
    - 물려 주는 클래스.
    - 상속하는 클래스에 비해 더 추상적인 클래스가 된다.
    - 상속하는 클래스의 데이터 타입이 된다.
- **파생(Derived) 클래스, 하위(Sub) 클래스, 자식(Child) 클래스**
    - 상속하는 클래스.
    - 상속을 해준 클래스 보다 좀더 구체적인 클래스가 된다.

- 상위 클래스와 하위 클래스는 계층관계를 이룬다.
  - 상위 클래스는 하위 클래스 객체의 타입이 된다.

In [ ]:

다중상속과 단일 상속

- 다중상속
  - 여러 클래스로부터 상속할 수 있다
- 단일상속
  - 하나의 클래스로 부터만 상속할 수 있다.
- 파이썬은 다중상속을 지원한다.
- MRO (Method Resolution Order)
  - 다중상속시 메소드 호출할 때 그 메소드를 찾는 순서.

  1. 자기자신
  2. 상위클래스(하위에서 상위로 올라간다)
     - 다중상속의 경우 먼저 선언한 클래스 부터 찾는다. (왼쪽->오른쪽)
- MRO 순서 조회
  - Class이름.mro()

In [ ]:

Method Overriding (메소드 재정의)

상위 클래스의 메소드의 구현부를 하위 클래스에서 다시 구현하는 것을 말한다.
상위 클래스는 모든 하위 클래스들에 적용할 수 있는 추상적인 구현밖에는 못한다.
이 경우 하위 클래스에서 그 내용을 자신에 맞게 좀더 구체적으로 재구현할 수 있게 해주는 것을 Method Overriding이라고 한다.
방법은 하위 클래스에서 overriding할 메소드의 선언문은 그래로 사용하고 그 구현부는 재구현하면 된다.

super() 내장함수

- 하위 클래스에서 상위 클래스의 instance를 반환(return) 해주는 함수
- 구문

```
super().메소드명()
```

- 상위 클래스의 Instance 메소드를 호출할 때 – super().메소드()
  - 특히 method overriding을 한 클래스에서 상위 클래스의 overriding한 메소드를 호출 할 경우 반드시 `super().메소드()` 형식으로 호출해야 한다.
- 같은 클래스의 Instance 메소드를 호출할 때 – self.메소드()

In [ ]:

객체 관련 유용한 내장 함수, 특수 변수

- `isinstance(객체, 클래스이름-datatype)` : bool
  - 객체가 두번째 매개변수로 지정한 클래스의 타입이면 True, 아니면 False 반환
  - 여러개의 타입여부를 확인할 경우 class이름(type)들을 리스트로 묶어 준다.
  - 상위 클래스는 하위 클래스객체의 타입이 되므로 객체와 그 객체의 상위 클래스 비교시 True가 나온다.
- `객체.__dict__`
  - 객체가 가지고 있는 Attribute 변수들과 대입된 값을 dictionary에 넣어 반환
- `객체.__class__`
  - 객체의 타입을 반환

특수 메소드

특수 메소드란

- 특정한 상황에서 사용될 때 자동으로 호출되도록 파이썬 실행환경에 정의된 약속된 메소드들이다. 객체에 특정 기능들을 추가할 때 사용한다.
  - 정의한 메소드와 그것을 호출하는 함수가 다르다.
    - ex) `__init__()` => 객체 생성할 때 호출 된다.
- 메소드 명이 더블 언더스코어로 시작하고 끝난다.
  - ex) `__init__()`, `__str__()`
- 매직 메소드(Magic Method), 던더(DUNDER) 메소드라고도 한다.
- 특수메소드 종류
  - https://docs.python.org/ko/3/reference/datamodel.html#special-method-names (https://docs.python.org/ko/3/reference/datamodel.html#special-method-names)

주요 특수메소드

- `__init__(self [, …])`
  - Initializer
  - 객체 생성시 호출 된다.
  - 객체 생성시 Attribute의 값들을 초기화하는 것을 구현한다.
  - self 변수로 받은 instance에 Attribute를 설정한다.
- `__call__(self [, …])`
- 객체를 함수처럼 호출 하면 실행되는 메소드
  - Argument를 받을 Parameter 변수는 self 변수 다음에 필요한대로 선언한다.
  - 처리결과를 반환하도록 구현할 경우 `return value` 구문을 넣는다. (필수는 아니다.)

In [ ]:

```

```

- `__repr__(self)`
  - Instance(객체) 자체를 표현할 수 있는 문자열을 반환한다.
    - 보통 객체 생성하는 구문을 문자열로 반환한다.
    - 반환된 문자열을 eval() 에 넣으면 동일한 attribute값들을 가진 객체를 생성할 수 있도록 정의한다.
  - 내장함수 **repr(객체)** 호출할 때 이 메소드가 호출 된다.
  - 대화형 IDE(REPL) 에서 객체를 참조하는 변수 출력할 때도 호출된다.

    - eval(문자열)
      - 실행 가능한 구문의 문자열을 받아서 실행한다.

- `__str__(self)`
  - Instance(객체)의 Attribute들을 묶어서 문자열로 반환한다.
  - 내장 함수 **str(객체)** 호출할 때 이 메소드가 호출 된다.
    - str() 호출할 때 객체에 `__str__()` 의 정의 안되 있으면 `__repr__()` 을 호출한다. `__repr__()` 도 없으면 상위클래스에 정의된 `__str__()` 을 호출한다.
    - print() 함수는 값을 문자열로 변환해서 출력한다. 이때 그 값을 str() 에 넣어 문자열로 변환한다.

In [ ]:

```

```

연산자 재정의(Operator overriding) 관련 특수 메소드

- 연산자의 피연산자로 객체를 사용하면 호출되는 메소드들
- 다항연산자일 경우 가장 왼쪽의 객체에 정의된 메소드가 호출된다.
  - `a + b` 일경우 a의 `__add__()` 가 호출된다.
- **비교 연산자**
  - `__eq__(self, other)` : self == other
    - == 로 객체의 내용을 비교할 때 정의 한다.
  - `__lt__(self, other)` : self < other, __gt__(self, other): self > other
    - min()이나 max()에서 인수로 사용할 경우 정의해야 한다.
  - `__le__(self, other)` : self <= other
  - `__ge__(self, other)` : self >= other
  - `__ne__(self, other)` : self != other

- **산술 연산자**
  - `__add__(self, other)` : self + other
  - `__sub__(self, other)` : self - other
  - `__mul__(self, other)` : self * other
  - `__truediv__(self, other)` : self / other
  - `__floordiv__(self, other)` : self // other
  - `__mod__(self, other)` : self % other

In [ ]:

class변수, class 메소드

- **class변수**
  - (Intance가 아닌) 클래스 자체의 데이터
  - Attribute가 객체별로 생성된다면, class변수는 클래스당 하나가 생성된다.
  - 구현
    - class 블럭에 변수 선언.
- **class 메소드**
  - 클래스 변수를 처리하는 메소드
  - 구현
    - @classmethod 데코레이터를 붙인다.
    - 첫번째 매개변수로 클래스를 받는 변수를 선언한다. 이 변수를 이용해 클래스 변수나 다른 클래스 메소드를 호출 한다.

static 메소드

- 클래스의 메소드로 클래스 변수와 상관없는 단순기능을 정의한다.
  - Caller 에서 받은 argument만 가지고 일하는 메소드를 구현한다.
- 구현
  - @staticmethod 데코레이터를 붙인다.
  - Parameter에 대한 규칙은 없이 필요한 변수들만 선언한다.

class 메소드/변수, static 메소드 호출

- 클래스이름.변수
- 클래스이름.메소드()

In [ ]: