OOP I

- 객체(Object)
- 객체지향프로그래밍(Object Oriented Programming)
- 클래스(Class)와 객체(Object)

주어.동사 이때 동사는 행동, 메서드

내용 간단 정리

- 클래스 - person
- 인스턴스 - iu / jimin
- 메서드 : 행동 ~ 함수
- 속성 : 이름

객체(Object)

> Python에서 **모든 것은 객체(object)**이다.
>
> 모든 객체는 **타입(type), 속성(attribute), 조작법(method)**을 가진다.

객체(Object)의 특징

- **타입(type)**: 어떤 연산자(operator)와 조작(method)이 가능한가?
- **속성(attribute)**: 어떤 상태(데이터)를 가지는가?
- **조작법(method)**: 어떤 행위(함수)를 할 수 있는가?

타입(Type)과 인스턴스(Instance)

type	instance
int	`0`, `1`, `2`
str	`''`, `'hello'`, `'123'`
list	`[]`, `['a', 'b']`
dict	`{}`, `{'key': 'value'}`

타입(Type)

- 공통된 속성(attribute)과 조작법(method)을 가진 객체들의 분류

인스턴스(Instance)

- 특정 타입(type)의 실제 데이터 예시(instance)이다.
- 파이썬에서 모든 것은 객체이고, **모든 객체는 특정 타입의 인스턴스**이다.

```
a = 10
b = 20
```

```
# a, b 는 객체
# a, b 는 int 타입(type)의 인스턴스
```

In [ ]:
```
# a 가 int 의 인스턴스인지 확인해봅시다.
a = 10  # 10은 인스턴스이다.
```

In [ ]:
```
type(a)
```

- 타입검사

In [ ]:
```
type(a) is int
```

In [ ]:
```
type(a) == int
```

In [ ]:
```
isinstance(a,int)
```

속성(Attribute)과 메서드(Method)

속성은 사람 이름, 메서드는 사람 행동

객체의 속성(상태, 데이터)과 조작법(함수)을 명확히 구분해 봅시다.

type	attributes	methods
complex	.real , .imag	
str	_	.capitalize() , .join() , .split()
list	_	.append() , .reverse() , .sort()
dict	_	.keys() , .values() , .items()

속성(Attribute)

- 속성(attribute)은 객체(object)의 상태/데이터를 뜻한다.

활용법

<객체>.<속성>

예시

```
3+4j.real
```

- `complex` 타입 인스턴스가 가진 속성을 확인해봅시다.

In [ ]:
```
# 복소수를 만들어보고, 타입을 출력해봅시다.
```

In [ ]:
```
complex_number = 3 + 4j
```

In [ ]:
```
type(complex_number)
```

In [ ]:
```
complex_number.real
```

In [ ]:
```
complex_number.imag
```

In [ ]:
```
# 허수부랑 실수부를 각각 출력해봅시다. complex 객체의 실수 속성과 허수 속성이라고도 표현 가능
```

In [ ]:
```

```

메서드(Method)

- 특정 객체에 적용할 수 있는 행위(behavior)를 뜻 한다.

활용법

<객체>.<조작법>()

예시

```
[3, 2, 1].sort()
```

- `list` type의 인스턴스에 적용 가능한 조작법(method)을 확인해 봅시다.

In [ ]:
```
# 리스트를 하나 만들고 정렬해봅시다. list 타입 객체의 sort() 메서드로 정렬 가능합니다.
```

In [ ]:
```
a = [3, 2, 1]
```

In [ ]:
```
type(a)
```

In [ ]:
```
a.sort()
# 리스트(주어).정렬(동사)
```

In [ ]:
```
a
```

In [ ]:
```
# list 타입의 객체들이 할 수 있는 것들을 알아봅시다. (list 타입 객체가 가지고 있는 모든 속성
```

In [ ]:
```
print(dir(a))
```

In [ ]:
```
print(dir(3+4j))
```

In [ ]:
```
a = [3, 2, 1]
a.pop()
```

In [ ]:
```
sorted(a).pop()
```

In [ ]:
```
sorted_a = sorted(a)
pop_number = sorted_a.pop()
# 이게 위에 코드와 같은 의미인것
```

객체 지향 프로그래밍(Object-Oriented Programming)

Object가 중심(oriented)이 되는 프로그래밍

<wikipedia - 객체지향 프로그래밍> >

> 객체 지향 프로그래밍(영어: Object-Oriented Programming, OOP)은 컴퓨터 프로그래밍의 패러다임의 하나이다.
>
> 객체 지향 프로그래밍은 컴퓨터 프로그램을 명령어의 목록으로 보는 시각에서 벗어나 여러 개의 독립된 단위, 즉 "객체"들의 모임으로 파악하고자 하는 것이다.

절차 중심 vs. Object 중심

> 프로그래밍 패러다임: 어떻게 프로그램을 작성할 것인가

Object 중심의 장점

<wikipedia - 객체지향 프로그래밍>

> 객체 지향 프로그래밍은 프로그램을 유연하고 변경이 용이하게 만들기 때문에 대규모 소프트웨어 개발에 많이 사용된다.
>
> 또한 프로그래밍을 더 배우기 쉽게 하고 소프트웨어 개발과 보수를 간편하게 하며,
>
> 보다 직관적인 코드 분석을 가능하게 하는 장점을 갖고 있다.

- 코드의 **직관성**
- 활용의 **용이성**
- 변경의 **유연성**

클래스(Class)와 객체(Object)

> `type` : 공통 속성을 가진 객체들의 분류(class)
>
> `class` : 객체들의 분류(class)를 정의할 때 쓰이는 키워드

클래스(Class) 생성

- 클래스 생성은 `class` 키워드와 정의하고자 하는 `<클래스의 이름>` 으로 가능하다.
- `<클래스의 이름>` 은 `PascalCase` 로 정의한다.
- 클래스 내부에는 데이터와 함수를 정의할 수 있고, 이때 데이터는 **속성(attribute)** 정의된 함수는 **메서드(method)**로 불린다.

활용법

```
class <클래스이름>:
    <statement>
class ClassName:
    statement
```

In [ ]:
```
# class를 만들고 type을 출력해 봅시다.
```

In [ ]:
```
class Person:
    pass
```

In [ ]:
```
print(type(Person))
```

In [ ]:
```
type(int)
```

인스턴스(Instance) 생성

- 정의된 클래스(`class`)에 속하는 객체를 해당 클래스의 인스턴스(instance)라고 한다.
- `Person` 클래스의 인스턴스는 `Person()` 을 호출함으로써 생성된다.
- `type()` 함수를 통해 생성된 객체의 클래스를 확인할 수 있다.

활용법

```
# 인스턴스 = 클래스()
person1 = Person()
```

- `person1` 은 사용자가 정의한(user-defined) `Person` 이라는 데이터 타입(data type)의 인스턴스이다.

In [ ]:
```
# Person 클래스의 인스턴스를 만들어 봅시다.
# 인스턴스의 type과 우리가 위에서 정의한 클래스의 doc의 출력해 봅시다.
```

In [ ]:
```
# Person 클래스의 인스턴스는 Person()을 호출함으로써 생성된다.

person1 = Person()   # person1은 인스턴스, Person은 클래스
# 사람이 클래스, 지민, 나 등등이 인스턴스
```

In [ ]:
```
type(person1)
```

메서드(Method) 정의

특정 데이터 타입(또는 클래스)의 객체에 공통적으로 적용 가능한 행위(behavior)들을 의미한다.
하나하나 모든 것들이 객체라고 생각!

> 클래스 내부에서 정의되어있는 함수 자체

활용법

```
class Person:
    # 메서드(method)
    def talk(self):    # 첫번째 인자로 self를 정의해봅시다.
        return '안녕'
```

In [ ]:
```
# Person 클래스에 talk() 메서드를 정의해봅시다.
```

In [ ]:
```
class Person:

    # 클래스 내부에서 정의된 함수 => 메서드
    def talk(self):
        print('안녕')
```

In [ ]:
```
# Person() 클래스의 iu인스턴스 생성
iu = Person()
```

In [ ]:
```
iu.talk()      # 실제 내부 호출은 Person.talk(iu) 와 같다
```

In [ ]:
```
# 메서드도 함수이기 때문에 추가적인 인자를 받을 수 있습니다.
```

In [ ]:
```
class Person:
    def talk(self):
        print( '안녕')

    def eat(self, food):
        print(f'냠냠 {food}')
```

In [ ]:
```
hyein = Person()  # Person이라는 클래스에 hyein이라는 인스턴스 생성
hyein.talk()
hyein.eat('카페')
```

In [ ]:
```
# 기본 인자, 가변 인자 리스트 등 함수의 인자와 동일하게 매개변수를 정의할 수 있습니다.
```

In [ ]:
```
class Person:
    def talk(self):
        print( '안녕')

    def eat(self, food='국밥'):
        print(f'냠냠 {food}')
```

In [ ]:
```
woojung = Person()
woojung.talk()
woojung.eat()
woojung.eat('피자')
```

self

> 인스턴스 자신(self)

- Python에서 인스턴스 메서드는 **호출 시 첫번째 인자로 인스턴스 자신이 전달**되게 설계되었다.
- 보통 매개변수명으로 `self` 를 첫번째 인자로 설정 (다른 이름도 가능하지만 추천하지는 않는다.)

In [ ]:
```
class Person:
    # _ underscore
    def __init__(self, name):  # 특별한 행동을 하는 method
        self.name = name       # 아 이거하면 아래에서도 쓸 수 있네 위에나온 예시
                               # 들과 다르게

    def talk(self):
        return f'안녕, 나는 {self.name}'
```

In [ ]:
```
# 자신의 이름으로 Person의 인스턴스를 생성해봅시다.
```

In [ ]:
```
john = Person('john')
```

In [ ]:
```
# talk() 메서드를 호출해 봅시다.
```

In [ ]:
```
john.talk()
```

In [ ]:
```
iu = Person('iu')
iu.talk()
```

In [ ]:
```
# talk 메서드의 첫번째 인자 self는 아래와 같은 뜻입니다.
```

In [ ]:
```

```

생성자(constructor) 메서드

인스턴스 객체가 생성될 때 호출되는 함수.
주민등록과 비슷한 느낌

활용법

```
def __init__(self):
    print('생성될 때 자동으로 호출되는 메서드입니다.')
```

- 생성자를 활용하면 인스턴스가 생성될 때 인스턴스의 속성을 정의할 수 있다.

소멸자(destructor) 메서드

- 인스턴스 객체가 소멸(파괴)되기 직전에 호출되는 함수.

활용법

```
def __del__(self):
    print('소멸될 때 자동으로 호출되는 메서드입니다.')
```

In [ ]:
```
# 생성자와 소멸자를 만들어봅시다.
```

In [ ]:
```
class Person:
    pass
```

In [ ]:
```
iu = Person()  # 생성자가 init이 정의되어 있지 않으면 Person()이렇게 해도 됨
```

In [ ]:
```
class Person:

    def __init__(self, name):
        self.name = name
        print(f'응애 앙 애기 {self.name}')
    def __del__(self):
        print(f'저는 갑니다...{self.name}')
```

In [ ]:
```
# 생성해 봅시다.
```

In [ ]:
```
# 위에서 self가 hong이 되는 것이고 instance값
# self.name 에 대응되는 name은 길동이 되는 것
# 그리고 print 할 때는 self.name이라고 쓰는게 관례적
```

In [ ]:
```
hong = Person('길동')    # 생성자가 정의되어 있기 때문에 이렇게 해도 됨
```

In [ ]:
```
del hong
```

In [ ]:
```
# 소멸시켜 봅시다.
```

In [ ]:
```

```

속성(Attribute) 정의

특정 데이터 타입(또는 클래스)의 객체들이 가지게 될 상태/데이터를 의미한다.

활용법

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

    def talk(self):
        print(f'안녕, 나는 {self.name}')
```

In [ ]:
```
# 인스턴스의 속성, 즉 개별 인스턴스들이 사용할 데이터를 정의해봅시다.
```

In [ ]:
```
class Person:
    def __init__(self, name):
        self.name = name

    def talk(self):
        print(f'안녕, 나는 {self.name}')
```

In [ ]:
```
# 생성자 함수를 통해 생성과 동시에 인스턴스 속성에 값을 할당할 수 있습니다.
```

In [ ]:
```
hyein = Person('hyein')
hyein.talk()
```

In [ ]:
```
# 인스턴스 변수의 값을 변경할 수도 있습니다.
```

In [ ]:
```
hyein.name = 'hyeju'    # 속성의 값 변경 가능.
```

In [ ]:
```
hyein.talk()
```

In [ ]:
```
dir([])
```

매직메서드

- 더블언더스코어(__)가 있는 메서드는 특별한 일을 하기 위해 만들어진 메서드이기 때문에 스페셜 메서드 혹은 매직 메서드 라고 불립니다.
- 매직(스페셜) 메서드 형태: __someting__

```
'__new__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__rmod__',
'__rmul__',
'__setattr__',
'__sizeof__',
'__str__',
```

__str__(self)

```
class Person:
    def __str__(self):
        return '객체 출력(print)시 보여줄 내용'
```

- 특정 객체를 출력(`print()`) 할 때 보여줄 내용을 정의할 수 있음

In [ ]:
```
# dir() 함수를 통해 해당 객체가 활용 가능한 메서드를 확인해봅시다.
```

In [ ]:
```
print(dir(''))
```

In [ ]:
```
# Person의 인스턴스 person1을 생성 후 출력해봅시다.
```

In [ ]:
```
class Person:
    def __init__(self, name):
        self.name = name
```

In [ ]:
```
# Person의 인스턴스 person1을 생성 후 출력해봅시다.
```

In [ ]:
```
person1 = Person('john')
print(person1)   # map에서 봤던 것과 비슷하다.
```

In [ ]:
```
print(map(int,['1']))
```

In [ ]:
```
# __str__() 매직메서드를 정의해봅시다.
```

In [ ]:
```
class Person:
    def __init__(self, name):
        self.name = name

    def __str__(self):
        return f'<사람 : {self.name}>'
```

In [ ]:
```
# Person의 인스턴스 person1을 생성 후 출력해봅시다.
person1 = Person('john')
print(person1)
```

정리

객체(Object)

- 객체는 자신 고유의 **속성(attribute)**을 가지며 클래스에서 정의한 **행위(behavior)**를 수행할 수 있다.

클래스(Class)

- 공통된 속성(attribute)과 행위(behavior)를 정의한 것으로 객체지향 프로그램의 기본적인 **사용자 정의 데이터형(user-defined data type)**

인스턴스(Instance)

- 특정 `class` 로부터 생성된 해당 클래스의 실체/예시(instance)

속성(Attribute)

- 클래스/인스턴스가 가지는 속성(값/데이터)

메서드(Method)

- 클래스/인스턴스에 적용 가능한 조작법(method) & 클래스/인스턴스가 할 수 있는 행위(함수)

- 클래스보다 상위 개념이 타입이다.
- 공통된 속성을 갖고 있는 것이 타입이다.
- 타입이 큰 덩어리이고, int나 그런 것이 부분집합 같은 느낌이다.
- 클래스의 타입은 타입이다.