12

Class

Python built-in objects

Object type	Example literals/creation
Numbers	1234, 3.1415, 0b111, 1_234, 3+4j, Decimal, Fraction
Strings	'code', "app's", b'a\x01c', 'h\u00c4ck', 'hÄck'
Lists	[1, [2, 'three'], 4.5], list(range(10))
Tuples	(1, 'app', 4, 'U'), tuple('hack'), namedtuple
Dictionaries	{'job': 'dev', 'years': 40}, dict(hours= 10)
Sets	set('abc'), {'a', 'b', 'c'}
Files	open('docs.txt'), open(r'C:\data.bin', 'wb')
Other core objects	Booleans, types, None
Program-unit objects	Functions, modules, classes
Implementation objects	Compiled code, stack tracebacks

Mark Lutz. (2025). Learning Python, 6th Edition. n.p.: O'Reilly Media, Inc.

Methods

- 클래스 내부에서 정의된 함수
- 객체.메서드() 형식으로 사용

구분	함수(function_name())	메서드(object.method())
호출 주체	독립적	특정 자료형의 객체
예시	print(), len()	‘hi’.upper(), [1].append()

Methods

예) str 클래스의 upper(), capitalize(), list 클래스의 append(), sort(), etc.

```
s = "hello"
s.upper()
```

```
'HELLO'
```

```
s.capitalize()
```

```
'Hello'
```

```
l = [1, 2, 3]
l.append(4)
print(l)
```

```
[1, 2, 3, 4]
```

Methods

- 여러 개의 메서드를 하나의 줄에서 연속해서 호출

```
s = " hello "
s = s.strip()
s = s.upper()
s = s.replace("H", "J")
print(s)
```

```
JELLO
```

```
s = " hello "
print(s.strip().upper().replace("H", "J"))
```

```
JELLO
```

self

- 메서드의 첫 번째 매개변수는 항상 self
- 호출 시, 해당 객체 자신이 자동 전달됨

```
class Calculator:
    def __init__(self):
        self.result = 0

    def adder(self, num):
        self.result += num
        return self.result

cal1 = Calculator()
cal2 = Calculator()
```

self

- self는 인스턴스 자신을 가리키는 이름
- 메서드는 항상 객체를 참조해야 하므로 self가 필수

1.클래스 이름으로 직접 호출할 때

```
Calculator.adder(cal1, 3)
```

1.인스턴스로 호출할 때

```
cal1.adder(3)
```

__init__

- 객체가 생성될 때 자동으로 호출되는 메서드
- 주로 초기값 설정에 사용
- init 없이도 객체는 생성할 수 있지만 속성 초기화 불가능

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

adam = User('Adam')
```

```
class Book:
    def __init__(self, title, pages):
        self.title = title
        self.pages = pages

reference = Book('Do it Python', [210, 215]
```

__init__

- 변수 생성 및 초기화

```
class Counter:
    def __init__(self):
        self.count = 0

    def add(self):
        self.count += 1
```

```
a = Counter()
a.add()
a.add()
a.count
```

```
2
```

- 문자열 반환 메서드

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

    def speak(self):
        return f"{self.name} Barks"
```

```
d = Dog('Jake')
d.speak()
```

```
'Jake Barks'
```

return of methods

- return 이 없는 경우, 주로 값을 출력, 변경. print 시 None 출력
- return 이 있는 경우, 내부 동작에 따른 값 반환, print로 출력 가능

```
class NextClass:
    def printer(self, text):
        self.message = text
        print(self.message)
```

```
x = NextClass()
x.printer('instance call')
print('------------------')
print(x.printer('instance call'))
```

```
instance call
------------------
instance call
None
```

```
class NextClass:
    def printer(self, text):
        self.message = text
        return self.message
```

```
x = NextClass()
x.printer('instance call')
print('------------------')
print(x.printer('instance call'))
```

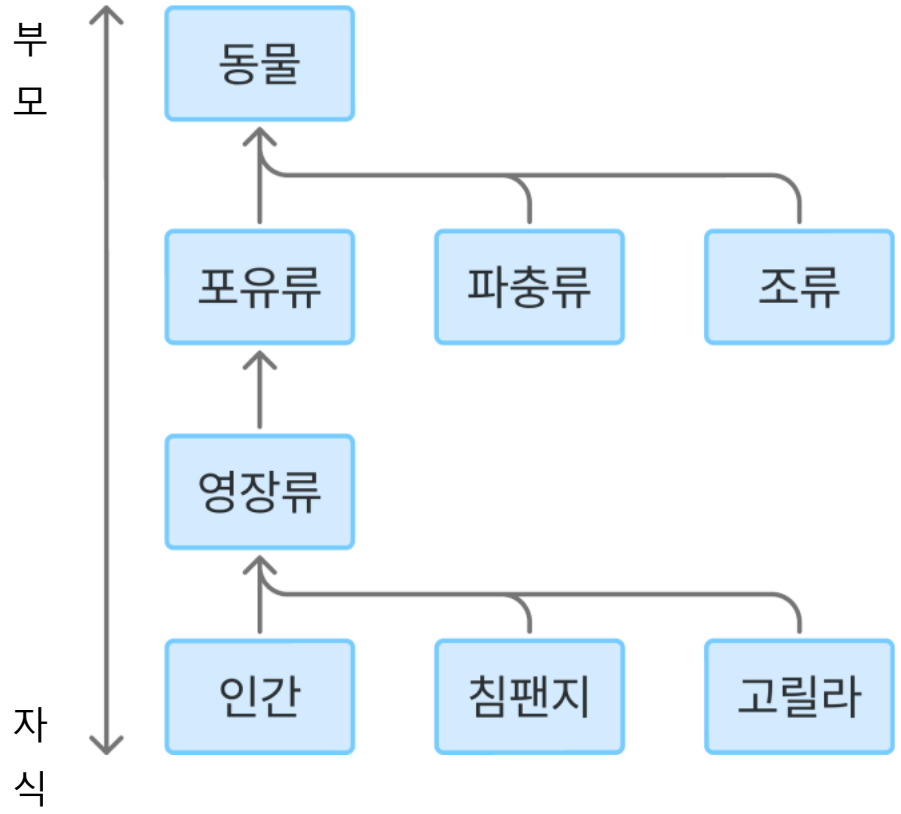
```
------------------
instance call
```

Inheritance

- 기존 클래스를 기반으로 새로운 클래스를 만드는 것
- 자식 클래스가 부모 클래스의 속성과 메서드를 물려받음

```
class 부모클래스:
    def __init__(self, 값1):
        self.속성1 = 값1

class 자식클래스(부모클래스):
    def __init__(self, 값1, 값2):
        super().__init__(값1)
        self.속성2 = 값2
```



Inheritance

- 이때 Dog는 Animal의 메서드를 그대로 사용 가능

```
class Animal:
    def speak(self):
        print("Hello")

class Dog(Animal):
    pass

d = Dog()
d.speak()  # Hello
```

Overriding

- 부모 클래스에서 정의한 메서드를 자식 클래스에서 재정의하는 것
- 동일한 이름으로 다른 동작 수행

```
class Animal:
    ...
class Dog(Animal):
    def speak(self):
        print("Woof!")

d = Dog()
d.speak()  # “Woof!”
```

Operator Oveloading

- 클래스가 일반적인 파이썬 연산 동작을 원하는 방식으로 정의
- 파이썬이 미리 정의한 특수 메서드 이름을 사용해 구현

연산자	메서드 이름
+	__add__
-	__sub__
*	__mul__
==	__eq__
len()	__len__
print()	__str__

Operator Overloading

```
class MyNumber:
    def __init__(self, value):
        self.value = value

    def __add__(self, other):
        return MyNumber(self.value + other.value)

    def __str__(self):
        return str(self.value)

a = MyNumber(3)
b = MyNumber(4)
print(a + b)  # 7
```

Example

- self: 객체 자신을 의미 (모든 메서드의 첫 인자)
- __init__: 인스턴스 초기화 메서드
- __str__: 객체 출력 시 사용자 지정 문자열 반환

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

    def __str__(self):
        return f'My name is {self.name}'

    def speak(self, message):
        print(f"{self.name}: {message}")
```

```
finn = Animal('Finn')
print(finn)
```

```
My name is Finn
```

```
finn.speak("Hello")
```

```
Finn: Hello
```

Example

- Dog 클래스는 Animal 클래스를 상속받음
- super().__init__(name)을 통해 부모 클래스의 초기화 메서드 호출
- self.breed = breed로 자식 클래스만의 속성 추가

```
class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name)
        self.breed = breed
```

```
jake = Dog('Jake', 'Bulldog')
print(jake)
```

```
My name is Jake
```

```
jake.speak("Hello")
```

```
Jake: Hello
```

Example

```
class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name)
        self.breed = breed

    def __str__(self):
        greet = super().__str__()
        greet += f" I'm a {self.breed}"
        return greet

    def speak(self, message):
        message += " Woof!"
        super().speak(message)

    def wag_tail(self):
        print(f"{self.name} is wagging his tail.")
```

```
jake = Dog('Jake', 'Bulldog')
print(jake)
```

```
My name is Jake I'm a Bulldog
```

```
jake.speak("Hello")
```

```
Jake: Hello Woof!
```

```
jake.wag_tail()
```

```
Jake is wagging his tail.
```

Summary

- 클래스와 객체
  - 생성자 __init__: 객체가 생성될 때 자동 호출
  - self: 인스턴스 자신을 가리키는 변수
  - Method: 클래스 안에 정의된 함수
  - Overriding: 부모 클래스의 메서드를 다시 정의
  - Operator overloading: +, ==, str() 등의 연산을 사용자 정의 클래스에 맞게 재정의
  - Inheritance: 다른 클래스로부터 속성과 메서드를 물려받음

12주차 과제

- 12_Class.ipynb