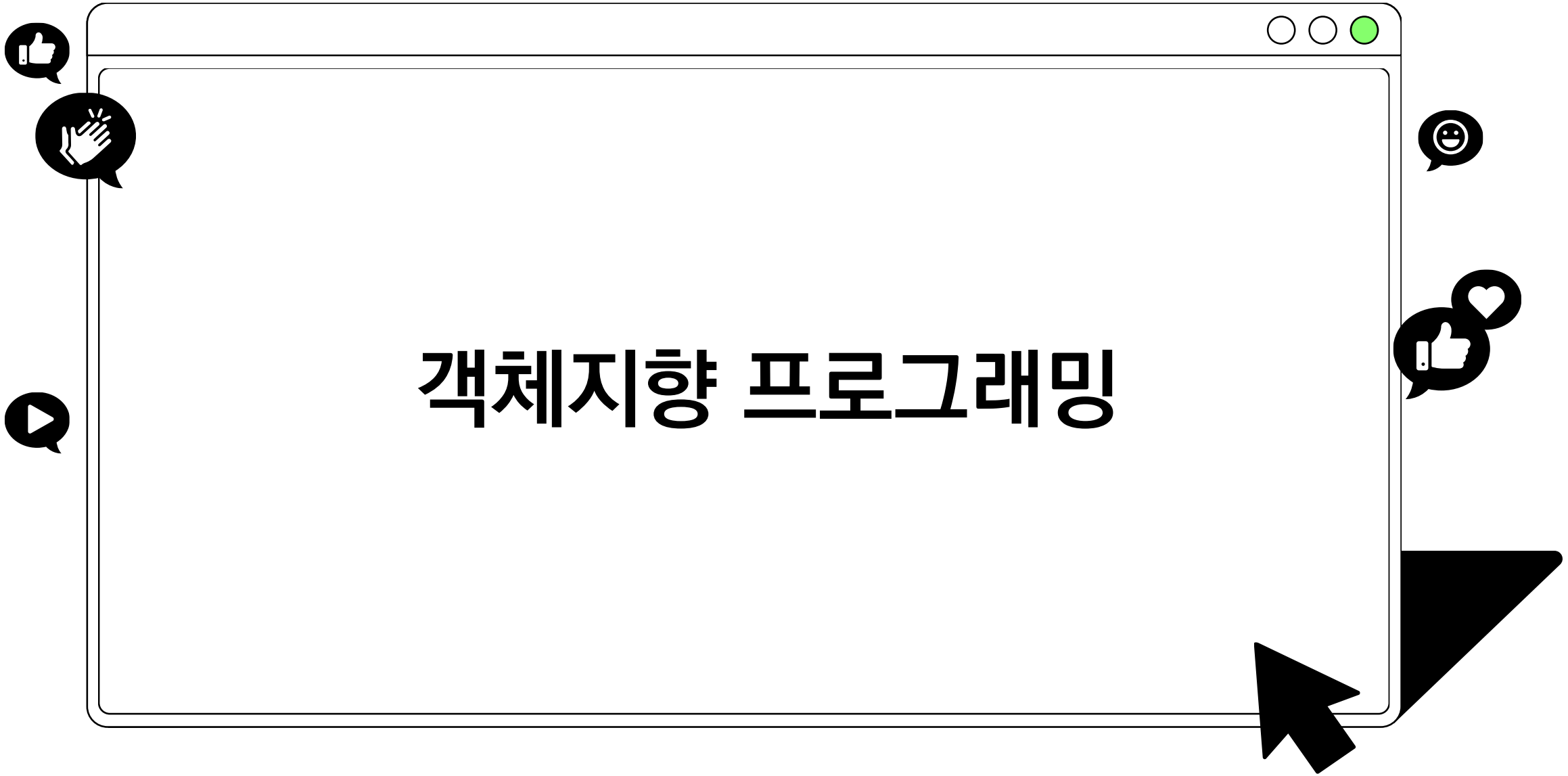
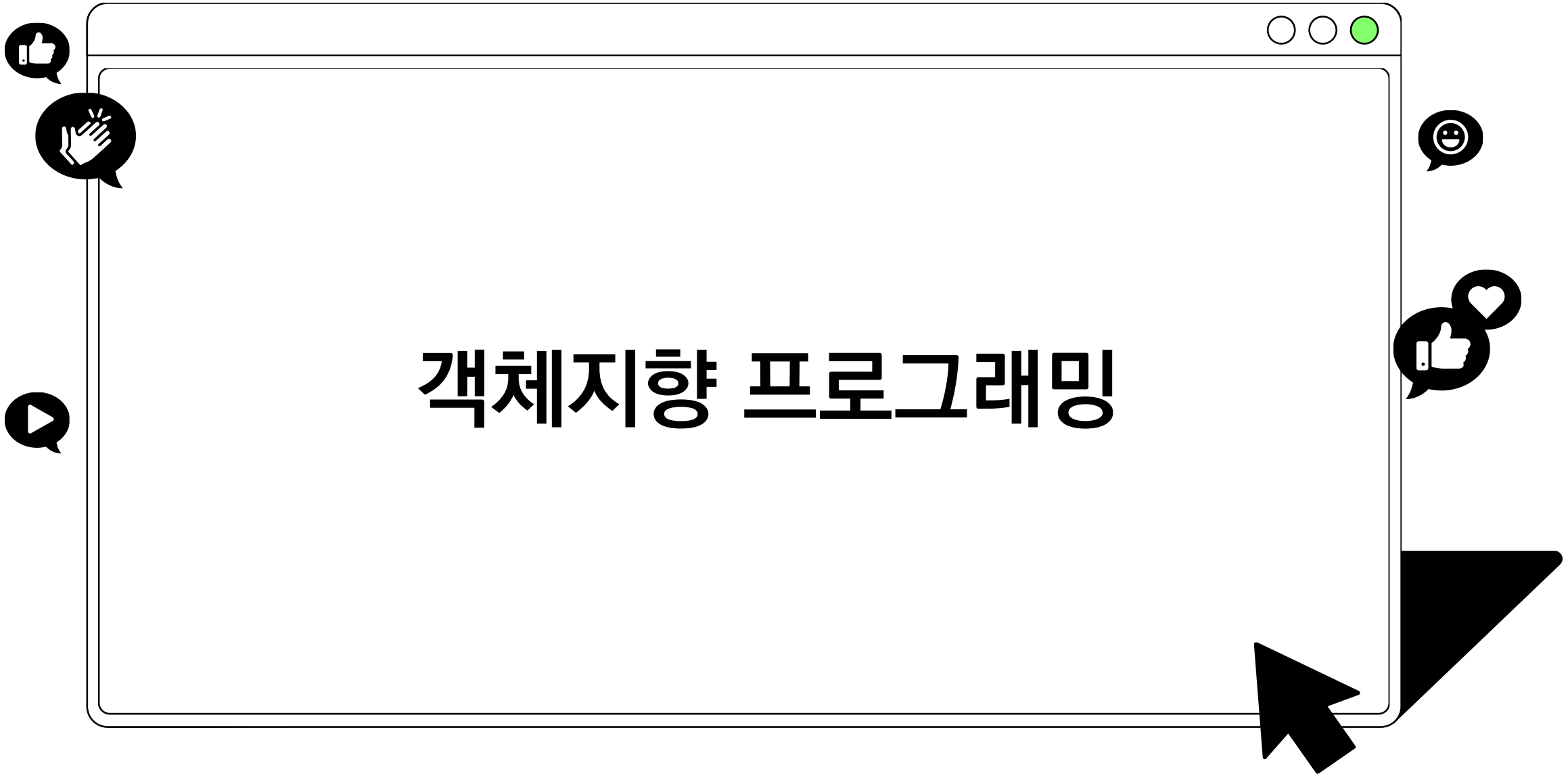
HAPPY HACKING



객체지향 프로그래밍

HAPPY HACKING



객체지향 프로그래밍

파이썬은 모두 객체(object)로 이뤄져 있다.

객체(object)는 특정 타입의 인스턴스(instance) 이다.

123, 900, 5는 모두 int의 인스턴스

'hello', 'bye'는 모두 string의 인스턴스

[232, 89, 1], []은 모두 list의 인스턴스

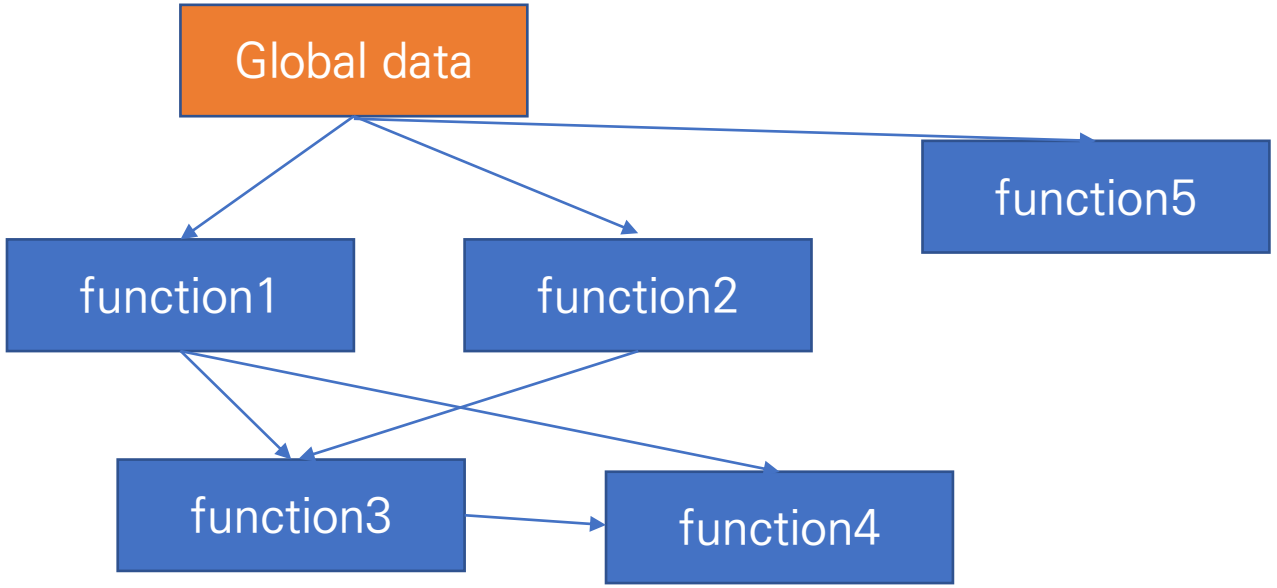
HAPPY HACKING

객체

- 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?
- 객체지향 프로그래밍이란?
  - 프로그램을 여러 개의 독립된 객체들과 그 객체들 간의 상호작용으로 파악하는 프로그래밍 방법

HAPPY HACKING

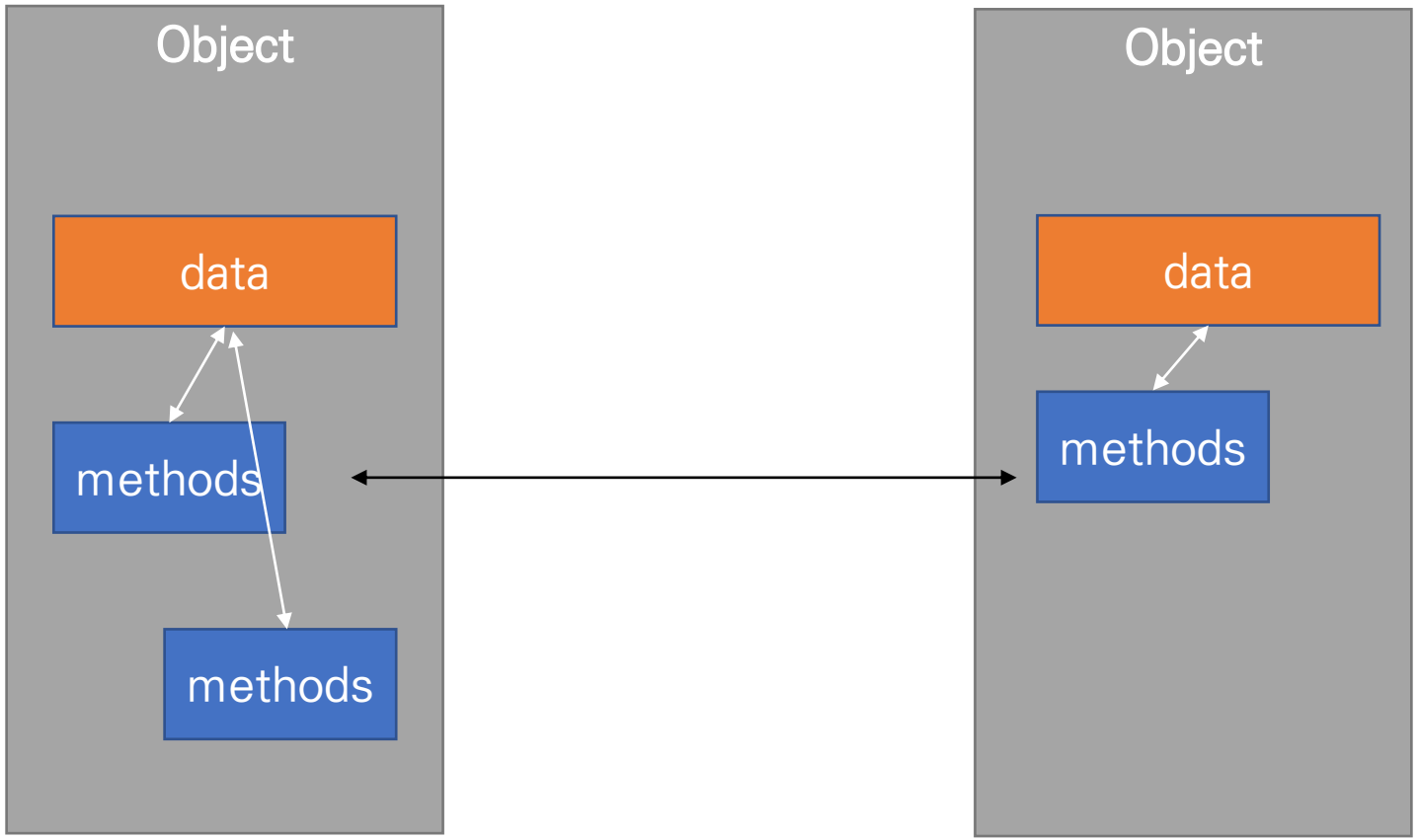
절차지향 프로그래밍



데이터와 함수로 인한 변화

HAPPY HACKING

객체지향 프로그래밍

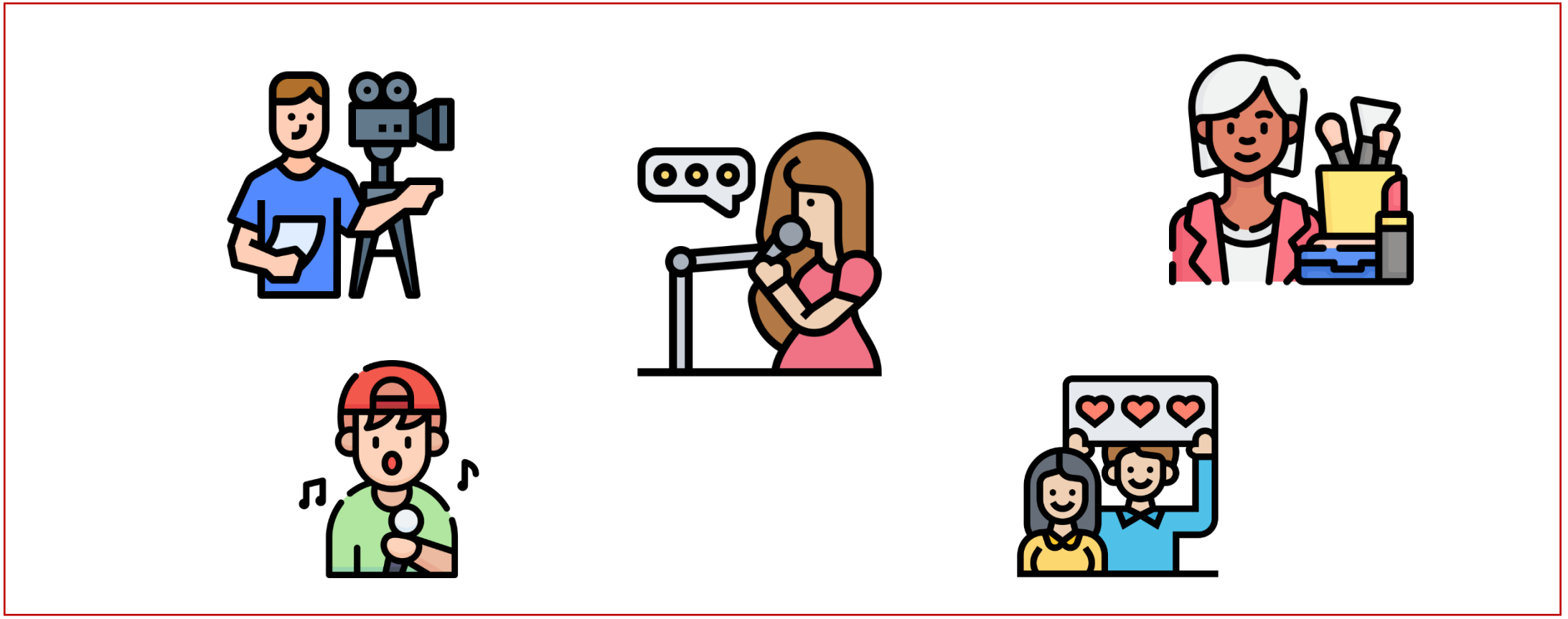


데이터와 기능(메소드) 분리, 추상화된 구조(인터페이스)

객체 지향 프로그래밍

- 현실 세계를 프로그램 설계에 반영(**추상화**)

Person



HAPPY HACKING

객체 지향 프로그래밍

- 현실 세계를 프로그램 설계에 반영(**추상화**)

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

    def greeting_message(self):
        return f'안녕하세요, {self.name}입니다.'
```

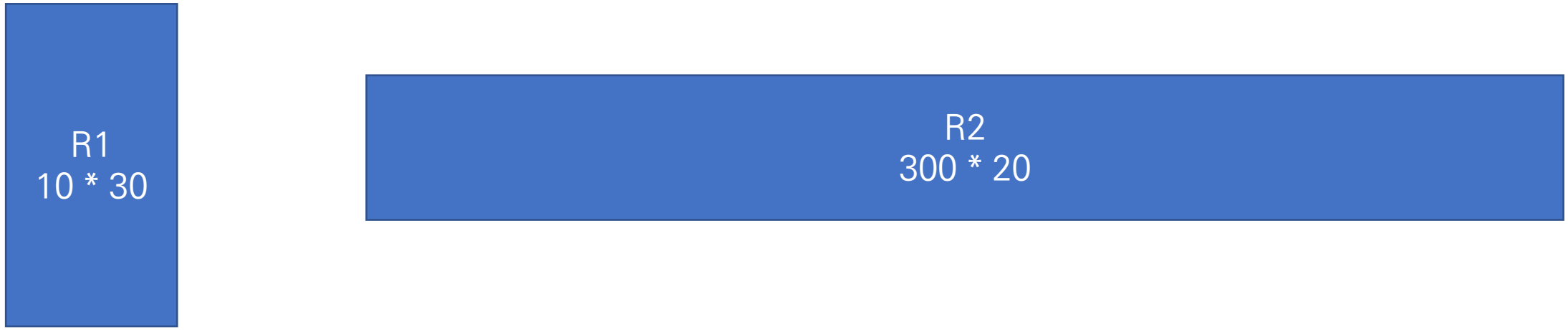
```
jimin = Person('지민', '남')
print(jimin.greeting_message())
# 안녕하세요, 지민입니다.

jieun = Person('지은', '여')
print(jieun.greeting_message())
# 안녕하세요, 지은입니다.
```

HAPPY HACKING

사각형 넓이 구하기 코드

- 예시



예시 : 절차지향 프로그래밍

```
a = 10
b = 30
square1_area = a * b
square1_circumference = 2 * (a + b)

c = 300
d = 20
square2_area = c * d
square2_circumference = 2 * (c + d)
```

예시 : 절차지향 프로그래밍

```
def area(x, y):
    return x * y


def circumference(x, y):
    return 2 * (x + y)


a = 10
b = 30
c = 300
d = 20
square1_area = area(a, b)
square1_circumference = circumference(a, b)
square2_area = area(c, d)
square2_circumference = circumference(c, d)
```

예시 : 객체지향 프로그래밍

```
class Rectangle:
    def __init__(self, x, y):
        self.x = y
        self.y = y

    def area(self):
        return self.x * self.y

    def circumference(self):
        return 2 * (self.x + self.y)


r1 = Rectangle(10, 30)
r1.area()
r1.circumference()

r2 = Rectangle(300, 20)
r2.area()
r2.circumference()
```

객체 지향 프로그래밍

- 사각형 – 클래스(class)
- 각 사각형 (R1,R2) – 인스턴스(instance)
- 사각형의 정보 – 속성(attribute)
  - 가로 길이, 세로 길이
- 사각형의 행동/기능 – 메소드(method)
  - 넓이를 구한다. 높이를 구한다.

HAPPY HACKING

객체지향의 장점 (위키피디아)

- 객체 지향 프로그래밍은 **프로그램을 유연하고 변경이 용이**하게 만들기 때문에 대규모 소프트웨어 개발에 많이 사용됩니다.
- 또한, 프로그래밍을 더 배우기 쉽게 하고 **소프트웨어 개발과 보수를 간편하게** 하며, 보다 **직관적인 코드 분석**을 가능하게 하는 장점을 가지고 있습니다.

HAPPY HACKING

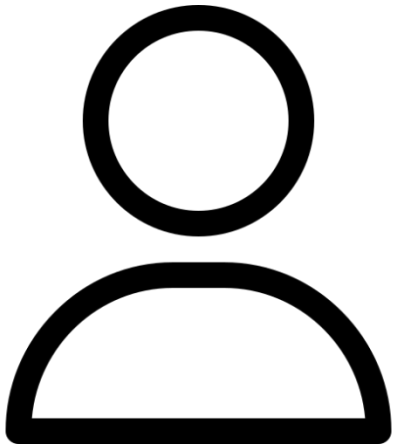
OOP 기초

기본 문법

```
# 클래스 정의
class MyClass:
    pass
# 인스턴스 생성
my_instance = MyClass()
# 메서드 호출
my_instance.my_method()
# 속성
my_instance.my_attribute
```

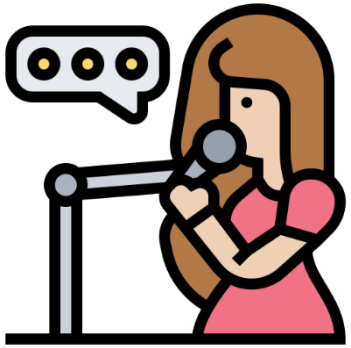
HAPPY HACKING

- 객체의 틀(클래스)을 가지고, 객체(인스턴스)를 생성한다.



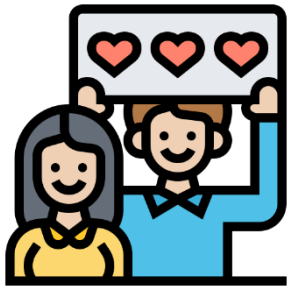
Person
(클래스)

가수 이지은
(인스턴스)



감독 강해피
(인스턴스)

팬 김해킹
(인스턴스)



HAPPY HACKING

- **클래스와 인스턴스**
  - 클래스 : 객체들의 분류(class)
  - 인스턴스 : 하나하나의 실체/예(instance)

```
class Person:
    pass

print(type(Person))
# type
p1 = Person()
type(p1)
# __main__.Person
isinstance(person1, Person)
# True
```

파이썬은 모든 것이 객체, 모든 객체는 특정 타입의 인스턴스

HAPPY HACKING

- **속성**
  - 특정 데이터 타입/클래스의 객체들이 가지게 될 상태/데이터를 의미

```
In [1]: class Person:

            def __init__(self, name):
                self.name = name

In [2]: person1 = Person('지민')

In [3]: person1.name
Out[3]: '지민'
```

HAPPY HACKING

- **메소드**
  - 특정 데이터 타입/클래스의 객체에 공통적으로 적용 가능한 행위(함수)

```
In [1]: class Person:

            def talk(self):
                print('안녕')

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

```
In [3]: person1 = Person()

In [4]: person1.talk()
안녕

In [5]: person1.eat('피자')
피자를 냠냠

In [6]: person1.eat('치킨')
치킨를 냠냠
```

HAPPY HACKING

- 객체 비교하기
  - ==
    - 동등한(equal)
    - 변수가 참조하는 객체가 동등한(내용이 같은) 경우 True
    - 두 객체가 같아 보이지만 실제로 동일한 대상을 가리키고 있다고 확인해 준 것은 아님
  - is
    - 동일한(identical)
    - 두 변수가 동일한 객체를 가리키는 경우 True

HAPPY HACKING

- 객체 비교하기

```
a = [1, 2, 3]
b = [1, 2, 3]

print(a == b, a is b)
```

```
True False
```

```
a = [1, 2, 3]
b = a

print(a == b, a is b)
```

```
True True
```

인스턴스

HAPPY HACKING

- **인스턴스 변수**
  - 인스턴스가 개인적으로 가지고 있는 속성(attribute)
  - 각 인스턴스들의 고유한 변수
- 생성자 메소드에서 `self.<name>`으로 정의
- 인스턴스가 생성된 이후 `<instance>.<name>`으로 접근 및 할당

인스턴스

HAPPY HACKING

```
In [1]: class Person:

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

인스턴스 변수 정의

```
In [2]: john = Person('john')
```

```
In [4]: print(john.name)
```

인스턴스 변수 접근 및 할당

```
john
```

```
In [5]: john.name = 'John Kim'
```

```
In [6]: print(john.name)
```

```
John Kim
```

HAPPY HACKING

- **인스턴스 메소드**
  - 인스턴스 변수를 사용하거나, 인스턴스 변수에 값을 설정하는 메소드
  - 클래스 내부에 정의되는 메소드의 기본
  - 호출 시, 첫번째 인자로 인스턴스 자기자신(self)이 전달됨

```
class MyClass

    def instance_method(self, arg1, …)
```

```
my_instance = MyClass()
my_instance.instance_method(…)
```

인스턴스

HAPPY HACKING

- `self`
  - 인스턴스 자기자신
  - 파이썬에서 인스턴스 메소드는 호출 시 첫번째 인자로 인스턴스 자신이 전달되게 설계
    - 매개변수 이름으로 `self`를 첫번째 인자로 정의
    - 다른 단어로 써도 작동하지만, 파이썬의 암묵적인 규칙

HAPPY HACKING

- 생성자(constructor) 메소드
  - 인스턴스 객체가 생성될 때 자동으로 호출되는 메소드
  - 인스턴스 변수들의 초기값을 설정
    - 인스턴스 생성
    - __init__메소드 자동 호출

```
class Person:

    def __init__(self):
        print('인스턴스가 생성되었습니다.')
```

```
person1 = Person()
```

```
인스턴스가 생성되었습니다.
```

```
class Person:

    def __init__(self, name):
        print(f'인스턴스가 생성되었습니다. {name}')
```

```
person1 = Person('지민')
```

```
인스턴스가 생성되었습니다. 지민
```

HAPPY HACKING

- 소멸자(destructor) 메소드
  - 인스턴스 객체가 소멸(파괴)되기 직전에 호출되는 메소드

In [3]:
```
class Person:

    def __del__(self):
        print('인스턴스가 사라졌습니다.')
```

In [4]:
```
person1 = Person()
del person1
```

```
인스턴스가 사라졌습니다.
```