[1-5. 함수와 클래스]

파이썬에서 함수(Function)는 코드를 조직화하고 재사용할 수 있게 하는 기본적인 구성 요소입니다. 함수는 특정 작업을 수행하는 코드 블록으로 정의되며, 필요할 때마다 호출되어 사용됩니다.

함수를 사용하면 코드의 중복을 줄이고 프로그램의 가독성과 유지 보수성을 향상시킬 수 있습니다.

1. 함수(Function)

1-1. 함수의 기본 구조

함수의 선언은 'def' 키워드로 시작하여, 콜론(:)으로 끝냅니다. 함수의 시작과 끝은 코드 본문의 들여쓰기로 구분합니다.

In [ ]:
```
def function_name(parameters):
    """docstring"""
    statement(s)
    return expression
```

- function_name : 함수의 이름입니다.
- parameters : 함수에 전달되는 인자를 정의하는 부분입니다. 인자는 선택적일 수 있습니다.
- docstring : 함수의 동작을 설명하는 문서화 문자열입니다. 선택적으로 사용됩니다.
- statement(s) : 함수에서 실행될 명령들입니다.
- return : 함수가 처리한 결과를 반환하는 부분입니다. 'return' 문은 선택적으로 사용됩니다.

1-2. 함수 사용 예시

In [ ]:
```
def greet(name):
    """인사말을 출력하는 함수"""
    return f"Hello, {name}!"

print(greet("Alice"))
```

```
Hello, Alice!
```

In [ ]:
```python
def greet(name, message="Good morning"):
    """사용자 정의 메시지와 함께 인사말을 출력"""
    return f"{message}, {name}!"

print(greet("Alice"))
print(greet("Bob", "Good evening"))
```

```
Good morning, Alice!
Good evening, Bob!
```

In [ ]:
```python
def get_min_max(numbers):
    """리스트의 최솟값과 최댓값을 반환"""
    return min(numbers), max(numbers)

min_val, max_val = get_min_max([1, 2, 3, 4, 5])
print(f"Minimum: {min_val}, Maximum: {max_val}")
```

```
Minimum: 1, Maximum: 5
```

1-3. 함수의 인자

1-3-1. 위치 인자(Positional Arguments)

- 함수에 인자를 순서대로 전달하는 방식입니다. 인자의 순서가 함수 정의에 명시된 매개변수의 순서와 일치해야 합니다.

In [ ]:
```python
def add(a, b):
    return a + b
result = add(2, 3)  # 5
print(result)
```

```
5
```

1-3-2. 키워드 인자(Keyword Arguments)

- 인자를 이름으로 전달하는 방식으로, 인자의 순서가 바뀌어도 됩니다.

In [ ]:
```python
def describe_pet(animal_type, pet_name):
    print(f"I have a {animal_type} named {pet_name}.")
describe_pet(animal_type="hamster", pet_name="Harry")
```

```
I have a hamster named Harry.
```

1-3-3. 기본 매개변수(Default Parameters)

- 함수 정의 시 매개변수에 기본값을 지정할 수 있습니다. 호출 시 해당 인자를 생략하면 기본값이 사용됩니다.

In [ ]:
```python
def describe_pet(pet_name, animal_type="dog"):
    print(f"I have a {animal_type} named {pet_name}.")
describe_pet(pet_name="Willie")  # animal_type은 "dog"으로 자동 설정
```

```
I have a dog named Willie.
```

1-3-4. 가변 위치 인자(Variable Positional Arguments)

- '*args'를 사용하여 함수가 임의의 개수의 위치 인자를 받을 수 있게 합니다.

In [ ]:
```python
def make_pizza(*toppings):
    print("Making a pizza with the following toppings:")
    for topping in toppings:
        print(f"- {topping}")
make_pizza('pepperoni')
make_pizza('mushrooms', 'green peppers', 'extra cheese')
```

```
Making a pizza with the following toppings:
- pepperoni
Making a pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese
```

1-3-5. 가변 키워드 인자(Variable Keyword Arguments)

- '**kwargs'를 사용하여 함수가 임의의 개수의 키워드 인자를 받을 수 있게 합니다.

In [ ]:
```python
def build_profile(first, last, **user_info):
    user_info['first_name'] = first
    user_info['last_name'] = last
    return user_info
user_profile = build_profile('albert', 'einstein', location='princeton',
print(user_profile)
```

```
{'location': 'princeton', 'field': 'physics', 'first_name': 'albert', 'la
st_name': 'einstein'}
```

1-4. 함수 인자의 전달

파이썬에서 함수에 인자를 전달할 때, 이는 "레퍼런스에 의한 전달(pass by reference)" 방식으로 이루어집니다.

이 말은, 함수에 인자로 전달되는 것이 실제 값의 복사본이 아니라, 그 값을 가리키는 레퍼런스(참조, 혹은 주소)라는 뜻입니다.

이로 인해 함수 내에서 인자로 받은 변수를 변경하면, 원본 데이터도 영향을 받을 수 있습니다. 다만, 전달되는 변수가 변경 불가능한 객체일 때와 변경 가능한 객체일 때는 각각 내부에서 처리하는 방식은 다릅니다.

1-4-1. 변경 불가능한 객체(Immutable Objects)

변경 불가능한 객체(예: 숫자, 문자열, 튜플)에 대해서는 이러한 참조 방식이 값의 복사처럼 보일 수 있습니다.

이는 함수 내부에서 이러한 인자의 값을 변경하려고 할 때, 변경 불가능한 객체의 특성 상 새로운 객체가 생성되고, 원본 객체는 변경되지 않기 때문입니다.

In [ ]:
```python
def modify(x):
    x = 10
    print("x inside function:", x)

x = 5
modify(x)
print("x outside function:", x)  # x는 여전히 5
```

```
x inside function: 10
x outside function: 5
```

1-4-2. 변경 가능한 객체(Mutable Objects)

리스트, 딕셔너리와 같은 변경 가능한 객체의 경우, 함수 내에서 객체를 변경하면 원본 객체에도 영향을 줍니다.

이는 함수에 전달된 레퍼런스를 통해 원본 객체 자체를 변경하기 때문입니다.

In [ ]:
```python
def modify(lst):
    lst.append(4)
    print("lst inside function:", lst)

lst = [1, 2, 3]
modify(lst)
print("lst outside function:", lst)  # lst에 4가 추가됨
```

```
lst inside function: [1, 2, 3, 4]
lst outside function: [1, 2, 3, 4]
```

1-5. pass

파이썬에서 'pass'문은 미래의 코드 구현을 위한 자리 표시자로 사용됩니다. 코드의 구조를 유지하면서 아직 구현하지 않은 기능의 자리를 마련해 둘 때 유용합니다.

사실상, 파이썬 인터프리터가 'pass' 문을 만나면 아무 것도 하지 않지만, 구문적으로 코드가 필요한 곳에서 구문 오류를 방지합니다.

이는 함수 정의, 반복문, 클래스 또는 조건문에서 실제 코드를 작성할 준비가 되지 않았지만 구조를 개요하고 싶을 때 특히 유용합니다.

In [ ]:
```python
def 나중에_구현할_함수():
    pass

if 나중에_구현할_함수() == True:
    pass  # 여기에 미래 구현 코드가 들어갑니다.
else:
    pass
```

2. 클래스(Class)

2-1. 클래스의 기본 구조

파이썬 클래스(class)는 데이터와 해당 데이터를 처리하는 메서드(method)들을 포함하는 틀이라고 생각할 수 있습니다. 클래스라는 틀을 이용하여 필요할 때마다 인스턴스(instance) 객체를 생성해서 사용할 수 있습니다.

즉 클래스는 일종의 도장과 같은 역할을 합니다. 공통적으로 사용되는 데이터와 함수(메서드) 들을 클래스에 모아 놓고, 필요할 때마다 도장을 찍듯이 인스턴스 객체를 생성해서 사용합니다.

이를 통해 코드를 논리적으로 구성하고 재사용성을 높일 수 있습니다.

클래스는 일반적으로 다음과 같은 구조를 가집니다.

In [ ]:
```python
class ClassName:
    # 속성(attribute) 정의
    attribute1 = value1
    attribute2 = value2

    # 메서드(method) 정의
    def method1(self, parameter1, parameter2):
        # 메서드 내용
        return something
```

- ClassName : 클래스 이름은 보통 첫 글자를 대문자로 작성하는 CamelCase 규칙을 따릅니다.
- 속성(attribute) : 클래스에 속한 변수입니다. 클래스의 상태를 나타내는 데이터로, 객체의 각 인스턴스에서 공유됩니다.
- 메서드(method) : 클래스에 속한 함수로, 클래스의 동작을 나타냅니다. 첫 번째 매개변수로 'self'를 갖고, 해당 클래스의 인스턴스를 참조합니다.

2-2. 클래스의 사용

클래스를 사용하여 객체를 생성하고 이를 활용할 수 있습니다. 이를 인스턴스화(instantiate)라고 합니다.

In [ ]:
```python
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def bark(self):
        return "Woof!"

# Dog 클래스의 인스턴스 생성
my_dog = Dog("Buddy", 3)

# 속성 접근
print(my_dog.name)  # 출력: Buddy
print(my_dog.age)   # 출력: 3

# 메서드 호출
print(my_dog.bark())  # 출력: Woof!
```

```
Buddy
3
Woof!
```

위 예시에서 'Dog' 클래스는 개를 나타내며, '**init**' 메서드는 개의 이름과 나이를 초기화 합니다.

'bark' 메서드는 개의 짖음을 나타냅니다.

이 클래스를 사용하여 'my_dog'이라는 인스턴스를 생성하고 속성에 접근하며 메서드를 호출합니다.

2-3. 클래스의 self 키워드

'self'는 파이썬에서 클래스의 인스턴스를 가리키는 키워드입니다. 클래스 내의 메서드에서 사용되며, 해당 메서드를 호출한 인스턴스 객체를 참조할 때 사용됩니다.

'self'를 통해 인스턴스 객체의 속성에 접근하고 수정할 수 있습니다.

일반적으로 파이썬에서 메서드의 첫 번째 매개변수로 'self'를 사용합니다. 다음은 'self'의 주요 특징과 사용 방법에 대한 설명입니다.

- 인스턴스 객체를 참조하는 용도 : 'self'를 사용하여 메서드가 속한 인스턴스 객체를 참조할 수 있습니다. 이를 통해 해당 인스턴스 객체의 속성에 접근하거나 수정할 수 있습니다.
- 메서드 내에서의 사용 : 클래스 내의 메서드에서는 항상 첫 번째 매개변수로 'self'를 사용해야 합니다. 이는 파이썬의 규칙이며, 메서드가 속한 인스턴스 객체를 가리키는 것을 의미합니다.
- 인스턴스 변수와 메서드 호출 : 인스턴스 변수(속성)와 메서드는 모두 'self'를 사용하여 접근됩니다. 인스턴스 변수는 'self'를 통해 접근하고 메서드는 'self'를 통해 호출됩니다.

예를 들어, 다음은 'Dog'클래스에서 '**init**' 메서드와 'bark' 메서드에서 'self'를 사용하는 예시입니다.

In [ ]:
```python
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def bark(self):
        return f"{self.name} says Woof!"
```

- '**init**' 메서드에서 'self.name'과 'self.age'는 각각 인스턴스 객체의 'name'과 'age' 속성을 나타냅니다. 이들은 해당 메서드를 호출한 인스턴스 객체의 속성으로 설정됩니다.
- 'bark' 메서드에서 'self.name'은 해당 메서드를 호출한 인스턴스 객체의 'name' 속성을 참조합니다.

2-3. 클래스와 인스턴스의 이름공간

클래스 객체(위 예제에서 Dog 클래스)와 인스턴스 객체(위 예제에서 my_dog)는 각자의 이름 공간(namespace)를 갖고 있으며, 객체의 속성과 메서드에 접근하는 데 사용됩니다.

2-3-1. 클래스 객체(Class Object)의 이름 공간

클래스 객체는 클래스를 정의할 때 생성되며, 클래스의 속성(attribute)과 메서드(method)를 포함하는 이름 공간을 갖습니다. 클래스 객체는 해당 클래스의 모든 인스턴스들이 공유하는 템플릿 역할을 합니다.

In [ ]:
```python
class Dog:
    species = 'Canis familiaris'

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

    def bark(self):
        return f"{self.name} says Woof!"
```

위 코드에서 'Dog' 클래스는 클래스 객체입니다. 'species', '**init**', 'bark'와 같은 속성과 메서드가 클래스 객체의 이름 공간에 저장됩니다. 이들은 모두 'Dog' 클래스의 인스턴스에서 공유됩니다.

한번 더 언급하면, 'species' 속성과 '**init**', 'bark' 메서드는 'Dog' 클래스의 이름 공간에 존재합니다.

2-3-2. 인스턴스 객체(Instance Object)의 이름 공간

인스턴스 객체는 클래스를 기반으로 생성되며, 해당 클래스의 속성과 메서드에 대한 복사본을 포함하는 이름 공간을 갖습니다.

인스턴스 객체는 클래스의 특정한 인스턴스(instance)를 나타냅니다.

In [ ]:
```
my_dog = Dog("Buddy", 3)
your_dog = Dog("Milk", 4)
```

In [ ]:
```
print(my_dog.name)
print(my_dog.age)
print(my_dog.bark())
print(my_dog.species)
```

```
Buddy
3
Buddy says Woof!
Canis familiaris
```

- 위 코드에서 'Buddy' 값을 가지고 있는 'name' 속성과 '3' 값을 가지고 있는 'age' 속성은 'my_dog' 인스턴스 객체의 이름 공간에 저장되어 있습니다.
- 'bark' 메서드의 경우 'Dog' 클래스의 이름 공간에 존재하지만, 'my_dog' 인스턴스 객체에서 해당 메서드에 접근이 가능합니다. 파이썬은 메서드를 호출할 때 해당 메서드를 찾기 위해 먼저 인스턴스 객체의 이름 공간을 검색한 후, 클래스 객체의 이름 공간을 검색합니다.
- 'my_dog' 인스턴스 객체에서 'species' 속성에 접근을 시도하는 경우 먼저 인스턴스 객체의 이름 공간을 검색한 후 클래스 객체의 이름 공간을 검색해서 해당 속성이 존재하는 경우 그 속성에 접근하게 됩니다.
- 즉, 위 예시에서 'name', 'age' 속성은 'my_dog' 인스턴스 객체의 이름 공간에 있는 속성의 값을 출력하였으며, 'bark' 메서드와 'species' 속성은 'Dog' 클래스의 이름 공간에 존재하는 메서드와 속성에 접근하여 해당 결과 값을 출력하였습니다.

In [ ]:
```
print(your_dog.name)
print(your_dog.age)
print(your_dog.bark())
print(my_dog.species)
```

```
Milk
4
Milk says Woof!
Canis familiaris
```

- 위 코드에서 'Milk' 값을 가지고 있는 'name' 속성과 '4' 값을 가지고 있는 'age' 속성은 'your_dog' 인스턴스 객체의 이름 공간에 저장되어 있습니다.
- 'bark' 메서드와 'species'속성은 'Dog' 클래스의 이름 공간에 존재하며, 'your_dog' 인스턴스 객체가 이에 접근해서 해당 결과값을 출력한 것입니다.

2-4. 생성자와 소멸자

생성자(Constructor)와 소멸자(Destructor) 메서드는 클래스의 인스턴스가 생성되고 소멸될 때 호출되는 특수한 메서드입니다.

2-4-1. 생성자(Constructor) 메서드

생성자 메서드는 객체가 생성될 때 자동으로 호출되는 메서드입니다. 주로 객체의 초기화(initialization)를 위해 사용됩니다.

생성자 메서드의 이름은 '**init**'으로 정의되며, 클래스를 초기화하고 속성을 설정하는 데 사용됩니다.

생성자 메서드의 일반적인 구문은 다음과 같습니다.

In [ ]:
```
class ClassName:
    def __init__(self, parameter1, parameter2, ...):
        # 객체 초기화 코드
        self.parameter1 = parameter1
        self.parameter2 = parameter2
        # ...
```

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

위 예시에서 '**init**' 메서드는 'Dog' 클래스의 생성자 메서드입니다. 이 메서드는 'name'과 'age' 매개변수를 받아들여 인스턴스의 'name'과 'age' 속성을 초기화합니다.

2-4-2. 소멸자(Destructor) 메서드

소멸자 메서드는 객체가 소멸될 때 자동으로 호출되는 메서드입니다. 주로 객체가 더 이상 필요하지 않을 때 할당된 자원을 해제하거나 정리하는 데 사용됩니다. 소멸자 메서드의 이름은 '**del**'로 정의되며, 객체의 소멸을 처리하는 코드를 포함합니다.

소멸자 메서드의 일반적인 구문은 다음과 같습니다.

In [ ]:
```python
class ClassName:
    def __del__(self):
        # 객체 소멸 처리 코드
```

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

    def __del__(self):
        print(f"{self.name} 객체가 소멸되었습니다.")
```

In [ ]:
```python
my_dog = Dog('Milk')
del(my_dog)
```

```
Milk 객체가 소멸되었습니다.
```

위 예시에서 '**del**' 메서드는 'Dog' 클래스의 소멸자 메서드입니다. 이 메서드는 객체가 소멸될 때 해당 객체의 이름을 출력하여 객체가 소멸되었음을 알려줍니다.