테크 지식 & 파이썬 프로그래밍 기초

5주차
객체지향: 함수와 클래스

20 . .

지은이: 이은재

Ⅰ. 함수

: 함수는 여러 단계에 걸쳐 특정한 작업을 실행하는 실행문들을 하나로 모으고 반복적으로 사용할 수 있도록 만들어주는 도구입니다. 프로그램을 만들다보면 똑같은 내용을 여러 번 작성하게 되는 경우가 많습니다. 이때 함수를 정의해 사용하면 불필요하게 중복되는 내용을 줄이고 프로그램 작동의 흐름도 더 명확하게 파악할 수 있습니다. 함수를 사용할 때와 사용하지 않을 때를 비교해보고 함수를 사용하는 이유와 사용방식을 체험해보겠습니다.

1) 함수 선언하기

함수를 만드는 일을 '선언'이라고 부릅니다. 일반적으로 아래와 같이 선언해서 사용할 준비를 할 수 있습니다. 만들 수 있는 함수의 종류나 개수는 무한히 많습니다. 그렇기 때문에 함수의 선언에는 함수의 이름과 매개변수, 함수의 내용, 반환할 결과가 프로그래머의 의도에 따라서 정확하게 정의되어야합니다. 반환 값이 없이 실행할 문장만 모여 있는 함수, 매개변수가 필요 없는 함수나 if문과 for문이 들어가는 함수, 심지어는 아무런 기능도 없는 함수도 만들 수 있습니다.

```
def 함수명(매개변수1, 매개변수2, ...):
        함수 호출시 실행할 문장1
        함수 호출시 실행할 문장2
        ...
```

ex1) 덧셈 함수 만들기

```
def my_add(number1, number2):
        result = number1 + number2
        return result

number3 = my_add(3, 14)
```

ex2) 원하는 횟수만큼 출력하는 함수 만들기

```
def want_to_print(content, times):
        for time in range(times):
                print(content)


want_to_print('안녕', 4)
```

- 전역 변수와 지역 변수 (global variables / local variables)

: 함수를 정의하는 코드 블록 내에서 사용되는 변수는 함수 외부의 변수와 구분됩니다. 대소문자 표기와 이름이 완전히 같더라도 둘은 서로 전혀 다른 변수입니다. 함수 내부의 변수는 지역변수로, 그 변수가 선언된 코드 블록 내에서만 그 성질이 유지됩니다. 함수 외부에서 선언되는 변수가 함수들 내부에까지 그대로 영향을 미치는 경우는 전역 변수라고 부르고, 프로그램 전체에서 계속 동일한 성질을 유지합니다. 따라서 다양한 함수가 선언되고 호출되는 프로그램일수록 전역변수와 지역변수의 구분을 제대로 해두지 않으면 많은 혼란이 발생합니다. 꼭 필요한 경우가 아니라면 전역변수의 사용은 피하는 것도 중요합니다.

ex1) 지역변수 사용하기

```
def world_richest():
    the_richest = 'Musk'
    print(the_richest)

world_richest()
print(the_richest)
```

ex2) 전역변수 사용하기

```
the_richest = 'Seo'

def world_richest():
    the_richest = 'Musk'
    print(the_richest)

world_richest()
print(the_richest)
```

ex3) 전역변수와 지역변수 구분해 사용하기

```
def world_richest():
    global the_richest
    print(the_richest)
    the_richest = 'Bill'

def richest_korean():
    the_richest = 'Seo'
    print(the_richest)
```

```
the_richest = 'Musk'
world_richest()
richest_korean()
print(the_richest)
```

2) 함수 호출

: 선언된 함수를 원할 때 불러와 사용하는 것을 함수를 '호출'한다고 합니다. 직접 만든 함수라면 정의된 내용을 잘 기억해 사용하면 되고 누군가가 미리 만들어 둔 함수라면 검색을 통해 그 정의와 기능을 확인할 수 있습니다.

ex) print 함수 documentation (https://docs.python.org/3/library/functions.html#print)

print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
Print objects to the text stream file, separated by sep and followed by end. sep, end, file, and flush, if present, must be given as keyword arguments.

All non-keyword arguments are converted to strings like str() does and written to the stream, separated by sep and followed by end. Both sep and end must be strings; they can also be None, which means to use the default values. If no objects are given, print() will just write end.

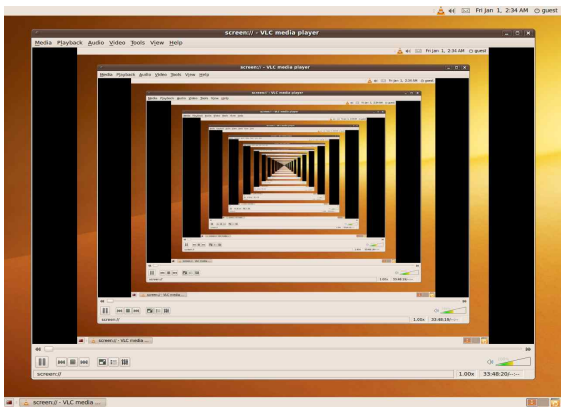
The file argument must be an object with a write(string) method; if it is not present or None, sys.stdout will be used. Since printed arguments are converted to text strings, print() cannot be used with binary mode file objects. For these, use file.write(...) instead.

Whether the output is buffered is usually determined by file, but if the flush keyword argument is true, the stream is forcibly flushed.

Changed in version 3.3: Added the flush keyword argument.

- 재귀함수란? (recursive function)

: 함수 중에는 정의 단계에서부터 자기 자신을 포함하는 것들이 있습니다. 자기 자신이 실행문에 들어있기 때문에 정확한 종료조건이 주어지지 않는 한 무한히 반복 실행됩니다. 이러한 함수를 재귀함수라고 하며, 이론적으로 모든 반복문 실행은 재귀적 방법으로 대체될 수 있습니다. 재귀는 처음 프로그래밍을 배울 때일수록 일상에서 접할 일이 많지 않은 독특한 개념이기 때문에 무한루프에 빠지는 경우가 많으니 주의해서 사용해야 합니다. 파이썬은 재귀실행횟수를 1000번 까지 기본 지원하는 언어입니다.



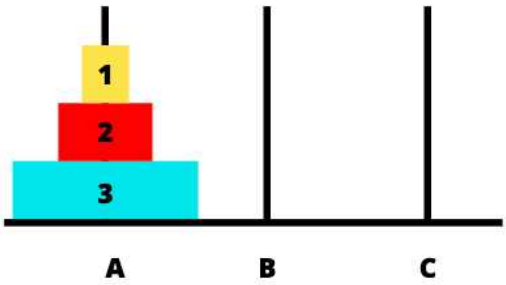
▲ 재귀적 실행의 예

ex1) 카운트다운

```
def count_down(n):
    if n == 0:
        print('Blast off!')
    else:
        print(n)
        count_down(n-1)

count_down(10)
```

ex2) 재귀적 문제해결 – 하노이의 탑




```
# n: 옮기려는 원반의 개수
# start_position: 옮기려는 원반의 초기 위치
# end_position: 원반을 옮기려는 최종 도착 위치
# via_position: 원반을 옮기는 도중에 거쳐가는 기둥
# 출력: 원반을 옮길 순서

def tower_of_hanoi(n, start_position, end_position, via_position):
    if n == 1:
        print(start_position, '->', end_position)
        return 0

    # 원반 n-1 개를 via_position으로 이동
    tower_of_hanoi(n-1, start_position, via_position, end_position)

    # 가장 큰 원반을 목적지로 이동
    print(start_position, '->', end_position)

    # via_position에 있는 원반 n-1개를 목적지로 이동
    tower_of_hanoi(n-1, via_position, end_position, start_position)

print('n = 1')
tower_of_hanoi(1, 1, 3, 2)

print('n = 2')
tower_of_hanoi(2, 1, 3, 2)

print('n = 3')
tower_of_hanoi(3, 1, 3, 2)
```

Ⅱ. 객체지향 - object oriented programming, OOP

- 객체지향이란?

: 제가 '프로그램이란 무엇일까?'라는 질문에 '명령어들의 집합'이라고 대답했던 것을 기억하시나요? 지금까지 우리는 프로그래밍을 배우면서 여러 개의 명령이 우리가 작성한 순서 그대로 실행되는 방식으로 프로그램을 작성해왔습니다. 이러한 프로그래밍 방법론을 '절차지향 프로그래밍'이라고 부릅니다. 물론 절차지향적인 방식만을 사용해도 프로그램 동작에는 아무런 문제가 없습니다. 실제로도 과거에는 프로그램들 대부분이 순차적인 명령어 실행만을 사용해서 개발되었습니다. 프로그램의 규모가 크지 않았고 사용자와의 상호작용을 크게 염두에 둘 필요도 없었을 시절이었기 때문입니다.

그러나 우리가 살고 있는 지금은 조작이 쉽고 직관적이며 예쁘기까지 한 대규모의 프로그램이 사용자 맞춤형으로, 그것도 온라인 환경에서 실행되는 것이 당연한 일처럼 되었습니다. 그런데 절차지향 프로그래밍은 몇 가지 아주 치명적인 단점을 가지고 있기 때문에 이처럼 현대적인 요구를 만족시키기가 어렵습니다.

우선, 프로그램에서 입력받고 저장하고 처리해야할 데이터의 종류와 특성은 이전에 비해 매우 다양하고 복잡해졌습니다. 그러나 데이터의 속성이 조금이라도 바뀌면 그에 맞는 코드를 처음부터 새로 설계해야하는 절차지향적 프로그램에서는 코드의 재사용이 매우 불편합니다. 또, 전체 프로그램의 규모가 커져 코드의 길이는 비교도 되지 않을 정도로 길어졌지만 일부만 따로 떼어낼 수 없다보니 개발자 여러 명이 동시에 협업하는 것도 불편했습니다. 게다가 프로그램 실행의 한 단계 한 단계가 모두 적혀있어야만 하는 절차지향적 프로그램에서는 프로그램 구동을 위한 데이터가 어떤 방식으로 어떻게 흘러가는지 사용자에게나 개발자에게 노출되지 않을 수 없습니다. 데이터 노출은 정보보안을 취약하게 만들기도 하지만 프로그램의 사용법을 복잡하게 만들고 인터페이스의 미관에도 악영향을 줍니다.

이외에도 기존 프로그램의 개발과 사용에서 일어났던 많은 문제들을 해결하기 위해 등장한 개념이 바로 "객체지향"입니다. 객체지향 프로그래밍 방식에서는 더 이상 프로그램을 '명령어들의 집합'으로 보지 않고, "객체들의 집합과 그 상호작용"으로 여깁니다. 어떻게 다르고 뭘 한다는 건지 아직 감이 잡히지 않으실 것 같습니다. 그래서 우선 여기에서는 객체지향의 가장 중요한 키워드들인 '추상화', '상속', '캡슐화', 그리고 '다형성'에 대해 알아보겠습니다. 이것들이 어떻게 작동하고 어떻게 도움이 될지 직접 생각해보고 그 과정에서 자연스럽게 '객체'를 이해하고 사용하는 연습을 해보겠습니다.

1) 추상화와 클래스 - abstraction & Class object

- 추상화

: 객체 간의 관계를 기반으로 하는 프로그램을 작성하다 보면 비슷한 기능과 역할을 하는 객체끼리는 서로가 담고 있는 데이터의 영역(domain)이나 함수의 실행과정이 겹치는 경우가 있습니다. 그때, 비슷한 객체들끼리 공통적으로 가지고 있는 특성만을 가지고 보다 일반화된 객체를 만들고 그 객체들에 적용될 함수들을 함께 모아놓을 수 있습니다. 이것을 '클래스' 라고 부릅니다. 이해를 돕기 위해서 예를 들어보겠습니다.

여기 동물들이 있습니다. 각 동물들의 특징을 데이터로 표현해본다면 어떨까요? 울음소리, 좋아하는 먹이, 귀와 눈, 다리의 개수 같은 것들을 말입니다. 이렇게요.

특징	강아지	고양이	햄스터	거북이	앵무새	금붕어
귀	2개	2개	2개	없음	2개	없음
눈	2개	2개	2개	2개	2개	2개
다리	4개	4개	4개	4개	2개	없음
꼬리	1개	1개	1개	1개	1개	1개
울음소리	멍멍	야옹	찍찍	없음	?	없음
먹이	고기	생선	견과류	잡식	잡식	사료
털	있음	있음	있음	없음	있음	없음

각각은 분명히 서로 다른 종이지만, 비슷한 부분도 아주 많고, 또 모두가 반려동물로 많이 기르는 동물이라는 것을 우리는 이미 잘 알고 있습니다. 그렇다면 이 동물들을 '반려동물'이라는 하나의 범주 아래에 묶어서 분류할 수도 있을 것 같습니다. 이때 귀, 눈, 다리, 꼬리, 울음소리, 먹이, 털 등등의 특징들은 각 동물의 특성을 파악할 수 있도록 해주는 공통의 속성 역할을 해줄 수 있습니다. 이 과정을 데이터의 추상화라고 하며, 코드로 표현해본다면 아마 아래와 같을 것입니다.

ex)
데이터의 ‘캡슐화’!

```
class Pet:
    def __init__(self, ears, eyes, legs, tail, cry, food, fur):
        self.ears = ears
        self.eyes = eyes
        self.legs = legs
        self.tail = tail
        self.cry = cry
        self.food = food
        self.fur = fur

    def cry_sound(self):
        print(self.cry)


dog = Pet(2, 2, 4, True, '멍멍', '고기', True)
dog.cry_sound()
```

- 클래스의 일반적인 형태와 인스턴스 생성방법

```
ex)
class 클래스명:
    def __init__(self, 속성1, 속성2, ...): #클래스의 속성(멤버) 선언 / 초기화
        self.속성1 = 속성1
        self.속성2 = 속성2
        ...

    def 메소드명(매개변수):
        메소드가 실행할 문장1
        메소드가 실행할 문장2
        ...


변수명 = 클래스명(속성1, 속성2, ...)
변수명.메소드명(매개변수)
```

2) 다형성과 상속 - inheritance

: 지금까지 우리는 클래스를 만드는 방법을 배워보았습니다. 그런데 실제로 클래스를 사용할 때는 뜻밖에 여러 가지 철학적인 문제들이 발생하게 됩니다. 대표적인 문제 중 하나는 하나의 객체가 한 가지 클래스의 속성들만을 가지는 것이 아니라 여러 클래스의 속성들을 동시에 가지도록 해야 할 것만 같은 상황이 발생한다는 것입니다.

앞서 들었던 예를 확장해보겠습니다. 우리는 강아지, 고양이, 햄스터, 거북이, 앵무새, 금붕어가 반려동물로 많이 길러진다는 점에 주목해서 Pet이라는 하나의 범주에 묶어보았습니다. 하지만 이 각각의 대상들을 묶어낼 수 있는 공통 속성은 그것들이 애완동물이라는 점 뿐만은 아닙니다. '동물'이라는 보다 상위의 개념으로 구분할 수도 있고, 아예 식물과 균류까지 모두 포함해 '생물'이라는 개념으로 규정할 수도 있습니다. 또 반대로 강아지만 해도 '치와와', '닥스훈트', '리트리버' 등등 더 세부적인 종으로 구체화하여 분류하는 클래스를 만들 수도 있습니다. 심지어 이러한 개념들은 고정되어있는 것이 아니고 얼마든지 우리 마음대로 설정해 선언할 수 있다는 성질이 있기 때문에 무한히 다양한 클래스를 만들 수 있겠지요. 이렇게 하나의 객체가 여러 가지 타입을 가질 수 있는 성질을 '다형성 (polymorphism)'이라고 합니다.

문제를 조금 구체적으로 표현하자면 리트리버는 반려동물인 동물이면서 동시에 강아지인데, 리트리버가 가진 속성을 데이터로 표현하려면 어떻게 해야 할까요? 이 모든 특성들을 반영할 수 있을까요? 객체지향 프로그래밍 언어에서는 '상속'이라는 개념을 도입해서 어느 정도 다형성 문제를 해소합니다. 즉, 리트리버는 '반려동물'이라는 클래스의 객체적 속성과 메소드를 유지하면서 '동물'이라는 클래스의 객체적 속성과 메소드를 상속 받는 방식입니다. 이때 상속을 받는 쪽

을 'Sub' 또는 '자식' 클래스, 상속을 하는 쪽을 'Spuer' 또는 '부모' 클래스라고 부릅니다. 이 내용을 코드로 표현하면 다음과 비슷해질 것 같습니다.

Q. 다중상속?

```
ex)
class Animal:
    def __init__(self, ears, eyes, legs, tail, cry, food, fur):
        self.ears = ears
        self.eyes = eyes
        self.legs = legs
        self.tail = tail
        self.cry = cry
        self.food = food
        self.fur = fur

    def cry_sound(self):
        print(self.cry)

class Pet(Animal):
    def __init__(self, ears, eyes, legs, tail, cry, food, fur, nick, master, home):
        Animal.__init__(self, ears, eyes, legs, tail, cry, food, fur)
        self.nick = nick
        self.master = master
        self.home = home

    def find_master(self):
        print(self.master)

    def go_home(self):
        print(self.home)

    def cry_sound(self):
        print(f'저희집 {self.nick}은(는) {self.cry}하고 울어요.')


doggy1 = Pet(2, 2, 4, True, '멍멍', '고기', True, '짱가', 'Tom', '서울')
doggy1.cry_sound()

doggy2 = Animal(2, 2, 4, True, '왈왈', '사료', True)
doggy2.cry_sound()
```

3) 정보은닉

: 효율적이고 직관적인 객체를 설계하기 위해서 신경 써야할 것은 사용자에게 어떤 정보를 노출하고, 접근 가능하도록 할지를 정하는 것입니다. 모든 데이터를 공개하고 접근할 수 있도록 한다면 사용자의 자유도는 올라가겠지만, 그만큼 프로그램의 보안이나 신뢰성, 사용성은 떨어지게 됩니다. 내가 유튜브 앱을 사용할 때 원하는 것은 그냥 보려는 동영상이 잘 나오는 것인데, 어떤 함수가 어떻게 몇 번 호출되어서 어느 서버에서 어떤 파일을 어떤 속도로 다운로드하는지는 알 필요도, 알 권한도 없을 테니 말입니다.

반대로 모든 데이터를 외부에서는 볼 수 없도록 하면 사용자는 개발자가 허락한 작업 외에는 아무것도 할 수 없게 될 것입니다. 동영상 하나를 넘길 때마다 새로 로그인을 해야하는 유튜브 앱을 떠올려 보세요. 끔찍하지 않으신가요? 그래서 프로그래머는 내부적으로만 작동하고 외부에는 드러내지 않을 함수는 어떤 것이어야 하는지를 잘 판단해야합니다. 이렇게 어떤 데이터, 어떤 함수나 메소드를 드러내고 숨길지 결정하는 과정을 '정보은닉'이라고 부릅니다.

Ⅲ. 모듈과 패키지

1) 모듈

: 객체와 메소드 즉, 캡슐화된 클래스가 다양하고 많아질수록 제대로 기억하고 효율적으로 호출해 사용하는 것이 어려워집니다. 처음부터 비슷한 기능이나 사용처가 비슷한 객체들끼리 분류해 모아두는 것이 좋습니다. 지금까지는 main.py라는 하나의 py 파일 내에서만 프로그램을 작성했지만, 여러 개의 py 파일을 만들고 필요할 때 불러와 사용할 수도 있습니다. 이렇게 main 스크립트 이외의 py 파일에 클래스와 메소드들을 모아둔 것을 '모듈'이라고 부릅니다. 다른 사람이 작성한 모듈이라고 할지라도 내 컴퓨터에 다운로드하면 코드에 포함시켜 사용할 수 있습니다.

2) 패키지 (라이브러리)와 프레임워크

: 다행스럽게도, 실제로 프로그램을 만들 때마다 우리가 스스로 모든 객체와 메소드, 모듈을 설계할 필요는 없습니다. 우리가 사용하기를 원하는 대부분의 기능은 이미 전 세계 수백, 수천만의 프로그래머들이 만들어두었기 때문입니다. 크고 작은 규모의 모듈을 기능별로 모아놓은 '패키지'가 셀 수 없이 다양하게 존재합니다. 파이썬 재단에서는 이러한 패키지들 중에서 누구나 무료로 사용할 수 있도록 공개된 오픈소스 패키지들을 한 곳에 모아 관리하고 검색해 사용할 수 있도록 돕고 있습니다.

또, 일반적으로 기업이나 개발자 커뮤니티에서 협력을 통해 개발하는 '프레임워크'라는 것도 있습니다. 프레임워크는 패키지보다 정보은닉의 강도가 높아 전체적인 프로그램의 작동 흐름을 내 마음대로 바꿀 수는 없지만, 앱 개발이나 게임 개발, 웹 개발처럼 아무것도 없는 맨땅에서 시작하기 어려운 작업들을 한결 편리하게 만들어줍니다. 파이썬을 개발언어로 사용하는 프레임워크 중에서는 '장고(Django)'나 '플라스크(Flask)'같은 웹 프레임워크나, '싸이킷런(Scikit-learn)', '파이토치(PyTorch)' 등의 AI 프레임워크가 널리 알려져 있습니다.

파이썬 패키지 인덱스 (Python Package Index, pypi.org)에 방문해서 어떤 패키지들이 있는지 꼭 확인해보시기 바랍니다. 그리고 언젠가는 여러분도 pypi에 기여할만한 패키지를 직접 만들어 보실 수 있기를 기대합니다.

ex)

```
import 모듈명

from 패키지명 import 모듈명(.객체명)
```

Q. 오픈소스와 Log4j 취약점?