2025.04.08. SR 파이썬 스터디 3주차

조코딩 04. 함수, 사용자 입출력, 파일 읽기 쓰기, 프로그램 입출력

✅ 함수(function)

: 특정 명령을 수행하는 명령어들을 매번 다 쓸 필요 없이 한 줄로 쓸 수 있도록 미리 묶어놓은 것.

example.
나는 며칠에 한 번씩 빵집에서 우유식빵을 사는 사람이라고 하자.
사람처럼 생긴 컴퓨터 로봇이 자기가 알아들을 수 있게 프로그램만 짜 주면 이 일을 대신 해 주겠다고 한다.
그래서 나는 프로그램을 짜 봤다. (자세히 읽을 필요 없음)

```
1. 카드를 잡는다.
2. 집 문을 열고 엘리베이터 앞에 도달할 때까지 직진한다.
4. 엘리베이터가 나오면 엘리베이터를 바라본다.
5. 엘리베이터 호출 버튼을 누른다.
6. 문이 열린 것이 식별되면 엘리베이터 안으로 들어간다.
7. 뒤로 돈다.
8. 1층 버튼을 누른다.
9. 엘리베이터 문이 다시 열리면 현관문 앞에 도달할 때까지 직진한다.
10. (현관에서 직진하면 빵집이 있다고 가정) 현관문이 열리면 빵집 앞에 도달할 때까지 직진한다.
11. 빵집의 문을 당긴다.
    11-1. 만약 빵집의 문이 열린다면 '우유식빵'이 나올 때까지 2번 빵집 내부를 탐색한다.
        11-1-1. 만약 '우유식빵'을 찾았다면 '우유식빵'을 손에 든다. 카운터 쪽으로 가서 카드를 건넨다.
                점원이 카드를 돌려주면 카드를 다시 잡는다.
                빵집의 문 앞까지 간다. 문을 당긴다.
        11-1-2. 만약 2번 탐색할 동안 '우유식빵'을 찾지 못했다면 빵집의 문 앞까지 간다. 문을 당긴다.
    11-2. 만약 빵집의 문이 열리지 않는다면 뒤로 돈다.
12. 현관이 나올 때까지 직진한다.
13. 현관문이 열리면 엘리베이터 앞에 도달할 때까지 직진한다.
14. 엘리베이터 호출 버튼을 누른다.
15. 문이 열린 것이 식별되면 엘리베이터 안으로 들어간다.
16. 뒤로 돈다.
17. 우리 집 층수와 일치하는 숫자가 써진 버튼을 누른다.
18. 엘리베이터 문이 다시 열리면 엘리베이터 밖으로 나간다.
19. 우리 집 쪽을 바라본다.
20. 집 문 앞에 도달할 때까지 직진한다.
21. 초인종을 누른다.
```

병적으로 디테일한 명령어들이다.
그러나 컴퓨터에 명령을 하려면 이 정도로 디테일하게 말해도 부족하다.
며칠마다 한 번씩 고작 빵 하나 사오게 하려고 이 모든 걸 매번 쓰고 있을 바엔 그냥 직

접 가는 게 나을 것이다.
그래서 고안된 게 함수다.
이 명령어를 하나로 묶어 저장시킨 뒤 컴퓨터에게 "앞으로 내가 '빵 사와' 라고 하면 항상 이걸 전부 실행해" 라고 명령을 해 놓으면,
'빵 사와'라는 한 마디만으로 저 21단계를 모두 실행시킬 수 있다.

실제 파이썬의 함수도 똑같은 개념이다.
def 다음에 함수의 이름을 설정해 주면, 그 이름만 쓰면 그 밑에 딸린 모든 명령어들을 다 같이 실행할 수 있다.

우리가 그 동안 많이 써 왔던 print(), input() 등등의 명령어들이 전부 함수다.

✅ 함수의 구조

In [1]:
```
def 함수이름(매개변수) :
    print('함수 실행됨')
    print('이렇게 여러 명령어들을 한 함수로 묶을 수 있음')
    return('함수의 자체 출력')
```

‼️ 설명을 위해 함수명, 변수명을 한국어로 썼음. 실제 코딩시에 함수명과 변수명을 한국어로 쓰는 건 절대 권장하지 않음.

✅ 매개변수(parameter)

함수 안에서 정의되는 변수. 함수 실행 종료시 삭제됨.

✅ return값, 반환값

함수가 반환하는 값. return()으로 지정. return값이 없는 함수도 가능함.

❓ return('asdf')와 print('asdf')는 뭐가 다른가?

1. return('asdf')

In [12]:
```
a = 1
b = 2
a + b
```

Out[12]: 3

In [3]:
```
def add(a, b) :
    return(a + b)

add(1, 2)
```

Out[3]: 3

위 두 코드는 사실상 다를 게 없다.
1 + 2든 add(1, 2) 든 똑같이 '3'을 반환한다.

이건 사칙연산을 해 보면 더 확실히 알 수 있다.

1 + 2 + 3 실행해 보기

In [33]:
```
1 + 2 + 3
```

Out[33]: 6

위의 def add(a, b) 부분을 실행한 뒤,
add(1, 2) + 3 실행해 보기

In [4]:
```
add(1, 2) + 3
```

Out[4]: 6

차이가 없이 정상적으로 실행되어 6이 나오는 걸 확인할 수 있다.

2. print('asdf')

In [6]:
```
def add(a, b) :
    print(a + b)

add(1, 2)
```

3

이것도 얼핏 보면 3을 '반환'한 것처럼 보일 수 있다.
하지만 이건 그냥 3을 출력했을 뿐, 반환한 건 아니다.

사칙연산을 실행해 보면 알 수 있다.

바로 위의 실행문을 실행한 뒤,
add(1, 2) + 3 실행해 보기

In [7]:
```
add(1, 2) + 3
```

3

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[7], line 1
----> 1 add(1, 2) + 3

TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
```

에러가 뜬다. 왜냐면 여기서의 add(1, 2)는 3을 print할 뿐이지 그 자체가 3은 아니기 때문이다.

return( )은 입력한 함수 자체를 return( )의 괄호 안에 들어있는 것과 동일시하게 해 줌.

- 동시에 함수가 끝났음을 의미. 그래서 return이 일단 실행되면 그 다음에 나오는 코드가 있다 해도 실행되지 않음.
- return()의 괄호 안에 여러 개의 자료가 있을 경우, 튜플의 형태로 반환함.

✅ 함수에 여러 입력 받기

: 매개변수 앞에 * 붙여주면 됨.

In [45]:
```
def add(*args) :
    sumation = 0
    for i in args :
        sumation += i
    return(sumation)
```

In [47]:
```
add(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

Out[47]: 55

다른 매개변수와 섞을 수도 있음.

In [56]:
```
def add(a, *args) :
    sumation = 0
    for i in args :
        sumation += i
    print(a)
    return(sumation)
```

In [55]:
```
add('프린트', 1, 2, 3)
```

프린트

Out[55]: 6

✅ global()

: 함수 안에서 함수 밖의 변수를 불러오게 해 줌.

안 쓸 수 있으면 안 쓰는 게 나음. 남발할 경우 코드가 난잡해짐.

참고: 스파게티 코드

난잡하게 짜여져서 가독성과 효율이 안 좋은 코드를 칭하는 말.
프로그래밍을 할 때 이런 코드를 짜면 당장은 잘 돌아간다 쳐도 나중에 유지보수가 어려워짐.

✅ lambda 함수

def 없이 함수를 정의하는 방법.

```
add = lambda a, b: a + b
add(1, 2)
```

간간히 쓰임.

✅ input()

: 자료를 출력창에서 직접 입력받을 수 있음.

In [60]:
```
a = input()
print(a)
```

```
asdf
```

✅ 파일 읽기, 쓰기

✅ 파일 열기: open

open(파일 위치/이름, 파일 열기 모드)

파일 위치/이름

- 이름만 쓸 경우: 현재 작업 중인 프로그램이 있는 위치에서만 파일 탐색.
- 위치를 써 줄 경우: 현재 작업 중인 프로그램이 있는 위치에 없는 파일도 불러올 수 있음.
- 존재하지 않는 파일명을 썼을 경우: 새로운 파일을 만듦.

파일 열기 모드

- r: 읽기 모드(read): 파일 변경 없이 읽기만 함.
- w: 쓰기 모드(write): 파일의 내용을 새로 씀.
- a: 추가 모드(append): 기존 파일 내용을 유지한 채 내용을 추가함.

ex.

```
open('abcd', 'r') # 파일 경로 지정 없이 이름만 입력. 현재 작업중인 프로그램이 있는 위치에서 탐색함. 같은 이름을 가진 파일이 없으면 새로 만듦.
open('C:/SR_OneDrive/25-1 파이썬 스터디/250508_SR_PythonStudy_CH5_Function_ReadnWrite.ipynb') # 파일 이름에 더해 경로까지 지정. 주어진 경로에서 탐색함. 같은 이름을 가진 파일이 없으면 새로 만듦.
```

- 윈도우 파이썬에서 경로의 구분은 /(슬래시)로 함.
- 윈도우 시스템에서 경로의 구분은 ₩(역슬래시 또는 원 표시)로 함.

인코딩 형식 지정

open(파일 위치/이름, 파일 열기 모드, encoding='인코딩 형식')

- 인코딩: 컴퓨터가 처리할 수 있게 써진 문자열을 사람의 언어로 해독하는 방법.
- 인코딩의 종류는 UTF-8, EUC-KR, CP949 등 여러 가지가 있음.
- 파일이 만들어졌을 때 사용한 인코딩 방식과 파일을 열 때 사용한 인코딩 방식이 다르면 파일이 깨짐.

(소위 말하는 외계어. '��녕하세요, �� ����분!' 이런 식으로..)
- 이렇게 파일이 깨지면 인코딩의 종류를 바꿔봐야 함.

✅ 파일 읽기만 하기: read

```
f = open(파일, 'r')
f.close()
```

사용 가능한 함수 f = open(파일, 'r')이 써져 있을 때

- f.readline(): 파일을 한 줄씩 읽음.
- f.readlines(): 파일을 한 줄씩 전부 읽어와 리스트 형태로 바꿈.
- f.read(): 파일을 줄로 구분하지 않고 통째로 읽어 옴.
- `for i in f :`
  이런 식으로도 가능함.
  이렇게 쓰면 f라는 파일 자체를 줄바꿈 문자(₩n, 엔터) 기준으로 구분된 자료의 묶음으로 취급함.

✅파일 쓰기: write

```
f = open('file.txt', 'w')
f.write(자료)
f.close()
```

: file이라는 텍스트파일을 열어서 f라는 변수에 집어넣음. f로 지정된 파일의 내용을 모두 지우고 자료를 새로 입력함. 이후 파일 닫음.

❗ 사용에 주의해야 함!! 'w'모드로 파일을 여는 즉시 그 파일의 내용은 모두 날아감.

✅ 파일 추가하기: append

```
f = open('file.txt', 'a')
f.write(자료)
f.close()
```

: file이라는 텍스트파일을 열어서 f라는 변수에 집어넣음. 기존 파일 내용을 유지한 채로 새로운 자료를 추가함. 이후 파일 닫음.

- write 모드: 사용에 조심해야 함. 이 모드로 원래 있던 파일을 열면, 기존에 존재하던 내용은 모두 사라져 버림.
- append 모드: 파일을 열어도 파일의 내용이 날아가지 않음. 동시에 내용을 추가할 수 있음

✅ 파일 닫기: close

```
f.close()
```

open을 하고 작업을 끝냈으면 반드시 close를 해 줘야 함.

with문 사용: 작업 종료 시 close를 자동으로 하게 해 줌.

```
with open('file.txt', 'w') as f :
    f.write('abcdefg')
```