6. 함수

개요

- 함수 호출
- 함수 정의
- 부울값 함수와 논리식
- 람다 함수
- 지역 변수와 전역 변수
- 프레임과 콜 스택

6.1. 함수 호출

함수 호출 방법

- 함수 호출function call: 함수 이름과 연결된 명령문 실행

```
함수이름(인자1, 인자2, ..., 인자n)
```

In [1]:

```
print(2.3, "hi", 2 + 4)
```

```
2.3 hi 6
```

In [2]:

```
type(2.37)
```

Out[2]:

```
float
```

인자를 받지 않는 함수의 호출

- 인자를 사용하지 않는 함수라 하더라도 함수 호출은 열고닫기 괄호를 반드시 사용해야 함
- `dir()` 함수: 현재 사용 가능한 함수와 변수의 목록

In [3]:
```
print(dir())
```

```
['In', 'Out', '_', '_2', '__', '___', '__builtin__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__', '_dh', '_i', '_i1', '_i2', '_i3', '_ih', '_ii', '_iii', '_oh', 'exit', 'get_ipython', 'quit']
```

- 괄호를 사용하지 않으면 함수 자체에 대한 정보를 확인

In [4]:
```
dir
```

Out[4]:
```
<function dir>
```

함수의 인자

- 함수는 적절한 개수의 인자와 함께 호출되어야 함
- 그렇지 않은 경우 오류가 발생

```
In [1]: type()
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [21], line 1
----> 1 type()

TypeError: type() takes 1 or 3 arguments
```

함수의 반환값

- 반환값: 함수가 지정된 명령문을 실행하면서 반환하는 값
- 반환된 값은 저장되거나 다른 연산에 재활용될 수 있음

In [5]:

```
y = int(3.14)
y + 1
```

Out[5]:

```
4
```

print() 함수의 반환값

- `print()` 함수에 의해 화면에 출력되는 문자열은 반환값이 아님.
- 화면에 지정된 값을 출력하는 명령문의 실행 결과에 불과
- `print()` 함수의 반환값은 `None`

In [6]:
```
x = print(3.14)
```

```
3.14
```

In [7]:
```
print(x)
```

```
None
```

None 값

- None은 아무런 의미가 없는 값
- 어떤 연산에도 사용할 수 없음

```
In [1]: x = None
        x + 1
        ---------------------------------------------------------------------------
        TypeError                                 Traceback (most recent call last)
        Cell In [25], line 2
              1 x = None
        ----> 2 x + 1

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

수식 계산 함수

- `math` 모듈: $\sin x$, $\cos x$, $\sqrt{x}$, $\log x$, e^x 등 수식 계산에 많이 사용되는 함수 제공

In [8]:
```
import math

math.log(2.718281828459045)
```

Out[8]:
```
1.0
```

In [9]:
```
math.sqrt(2)
```

Out[9]:
```
1.4142135623730951
```

In [10]:
```
math.exp(1)
```

Out[10]:
```
2.718281828459045
```

NumPy 라이브러리

- 데이터분석 등 많은 계산이 필요한 영역에서 활용됨
- `math` 모듈에서 지원하는 함수들을 포함해서 보다 많은 함수와 기능 지원

In [11]:
```
import numpy as np

np.log(2.72)
```

Out[11]:
```
1.0006318803079 06
```

In [12]:
```
np.sqrt(2)
```

Out[12]:
```
1.4142135623730951
```

In [13]:
```
np.exp(1)
```

Out[13]:
```
2.718281828459045
```

함수 호출과 표현식

- `f(x1, ..., xn)`와 같은 함수 호출 표현식은 함수의 반환값을 가리킴

In [14]:

```
x = int(float('3.14'))
print(f"x가 가리키는 값은 {x}이며 자료형은 {type(x)}이다.")
```

```
x가 가리키는 값은 3이며 자료형은 <class 'int'>이다.
```

6.2. 함수 정의

헤더와 본문

- **헤더**header: 키워드 `def` 로 시작하는 첫째줄은 함수의 기본 정보 제공
- **본문**body: 함수가 호출되었을 때 실행해야 하는 들여써진 명령문
- 함수 이름과 본문이 서로 연결되어 저장됨

```
def 함수이름(매개변수1, 매개변수2, ..., 매개변수n):
    명령문
```

함수의 반환값과 `return` 키워드

- 아래 모양의 명령문을 실행하는 순간 결정

```
return 표현식
```

- `return` 명령문이 실행되는 순간 지정된 `표현식`이 가리키는 값이 반환되면서 함수의 실행이 멈춤

`return None`

- return 명령문이 없는 함수

In [15]:

```
def double_print(s):
    print(s*2)
```

- 아래처럼 `return None`이 자동으로 추가된다고 생각하면 됨

In [16]:

```
def double_print(s):
    print(s*2)

    return None
```

매개 변수와 인자

- **매개 변수**parameter: 함수 선언할 때 헤더에 사용된 변수

```
def 함수이름(매개변수1, 매개변수2, ..., 매개변수n):
    명령문
```

- **인자**argument: 함수 호출에 사용되는 값

```
함수이름(인자1, 인자2, ..., 인자n)
```

함수 호출과 반환값

In [17]:

```
def myAdd(left, right):
    sum = left + right
    return sum
```

- `myAdd(-2, 5)`를 실행하면 아래 명령문이 실행되어 3을 반환함

```
left = -2
  right = 5
  sum = left + right
```

1종 객체

- **1종 객체**first-class object: 변수 할당, 함수의 인자, 함수의 반환값 등으로 사용될 수 있는 객체(값)
- 프로그래밍 언어에 따라 1종 객체의 기준이 다름
- C 언어 예제: 함수 1종 객체가 아님. 포인터 변수라는 다른 종류의 변수 활용 필요.

함수 활용: 변수 할당

- 함수 자체를 변수 할당에 사용 가능

In [18]:
```
a_function = myAdd
print("함수:", a_function)
```

```
함수: <function myAdd at 0x000001A7679E6320>
```

- 괄호을 사용하면 함수 호출을 의미함.

In [19]:
```
a_return_value = myAdd(-2, 5)
print("반환값:", a_return_value)
```

```
반환값: 3
```

함수 활용: 인자

- 함수를 다른 함수의 인자로 사용 가능

In [20]:
```
def do_twice(fn, arg):
    x1 = fn(arg)
    x2 = fn(x1)
    return x2
```

- 예제

In [21]:
```
def three_times(num):
    return num * 3
```

In [22]:
```
do_twice(three_times, 2) == three_times(three_times(2))
```

Out[22]:
```
True
```

함수 활용: 반환값

- 함수를 반환하는 함수

In [23]:

```
def adding_n_func(n):

    def myAdd_n(m):
        return n + m

    return myAdd_n
```

In [24]:

```
myAdd10 = adding_n_func(10)  # 10 을 더하는 함수
myAdd10(3)                   # 10 + 3
```

Out[24]:

```
13
```

지역 함수

- `myAdd_n()` 함수는 `adding_n_func()` 함수의 본문에서만 사용 가능한 **지역 함수**local function

```
In [1]: myAdd_n(3)
        ---------------------------------------------------------------------------
        NameError                                 Traceback (most recent call last)
        Cell In [43], line 1
        ----> 1 myAdd_n(3)

        NameError: name 'myAdd_n' is not defined
```

map() 함수

In [25]:

```
list(map(myAdd10, [3, 4, 5, 6]))
```

Out[25]:

```
[13, 14, 15, 16]
```

위치 인자와 키워드 인자

- 함수의 인자는 위치 인자와 키워드 인자 두 종류로 구분
- 위치 인자를 먼저, 키워드 인자를 나중에 작성
- `print()` 함수: `sep` 이외에 `end`, `file`, `flush`를 키워드 인자로 사용

```
print(value, ..., sep=' ', end='Wn', file=sys.stdout, flush=False)
```

- 말줄임표(`...`): 여러 개의 위치 인자를 사용할 수 있음을 나타냄

In [26]:

```
print('Hello,', 'Python', '!')
print("===")
```

```
Hello, Python !
===
```

In [27]:

```
print('Hello,', 'Python', '!', sep='₩n')
print("===")
```

```
Hello,
Python
!
===
```

In [28]:

```
print('Hello,', 'Python', '!', sep='₩n', end='')
print("===")
```

```
Hello,
Python
!===
```

키워드 인자 활용 예제

In [29]:

```
def myAdd10(left, right=10):
    add10 = left + right
    return add10
```

둘째 인자가 생략되면 자동으로 10이 대신 사용된다.

In [30]:

```
myAdd10(5) # right=10
```

Out[30]:

```
15
```

둘째 인자를 별도로 지정하면 지정된 값이 사용된다.

In [31]:

```
myAdd10(5, right=20)
```

Out[31]:

```
25
```

6.3. 부울값 함수와 논리식

부울값 함수란?

- 부울값을 반환하는 함수는 논리식에 사용되며 복잡한 테스트를 다룰 때 사용
- 예제: 나누셈 가능성 활용

In [32]:

```
def is_divisible(x, y):
    return x % y == 0
```

- 부울값 함수 호출을 조건문 등에 바로 사용 가능

In [33]:

```
x = 16
y = 4

if is_divisible(x, y):
    print(f'{x} 을(를) {y} (으)로 나눌 수 있다.')
```

```
16 을(를) 4 (으)로 나눌 수 있다.
```

filter() 함수

In [34]:

```
def divisible_by_3(x):
    return is_divisible(x, 3)
```

In [35]:

```
list(filter(divisible_by_3, [3, 4, 5, 6, 7]))
```

Out[35]:

```
[3, 6]
```

6.4. 람다 함수

람다 함수 정의

- 람다lambda 함수의 형식

```
  lambda 인자1, ..., 인자n :
expression
```

- 인자에 10을 더해주는 함수

In [36]:
```
lambda a : a + 10
```

Out[36]:
```
<function __main__.<lambda>(a)>
```

- 두 인자의 곱을 반환하는 함수

In [37]:
```
lambda a, b : a * b
```

Out[37]:
```
<function __main__.<lambda>(a, b)>
```

람다 함수 호출

- 람다 함수는 이름이 없기에 호출하려면 함수 전체를 사용

In [38]:
```
(lambda a : a + 10)(5)
```

Out[38]:
```
15
```

In [39]:
```
(lambda a, b : a * b)(2, 5)
```

Out[39]:
```
10
```

6.5. 지역 변수와 전역 변수

지역 변수 대 전역 변수

- 지역 변수local variable: 함수의 매개 변수, 함수 본문에서 선언되는 변수 등 제한된 영역에서만 사용되는 변수
- 전역 변수global variable: 프로그램 전체에서 사용되는 변수

In [40]:

```
def hour2min(hour):
    minutes = hour * 60
    return minutes

two_hour = hour2min(2)
```

- `two_hour` 는 전역변수

In [41]:

```
print(two_hour)
```

```
120
```

- `minutes`와 `hour`는 지역 변수

```
In [1]: print(minutes)
        ---------------------------------------------------------------------------
        NameError                                 Traceback (most recent call last)
        Cell In [61], line 1
        ----> 1 print(minutes)

        NameError: name 'minutes' is not defined
```

```
In [2]: print(hour)
        ---------------------------------------------------------------------------
        NameError                                 Traceback (most recent call last)
        Cell In [61], line 1
        ----> 1 print(minutes)

        NameError: name 'hour' is not defined
```

- 참고: PythonTutor:지역 변수와 전역 변수 1
- 참고: PythonTutor:지역 변수와 전역 변수 2
- 참고: PythonTutor:지역 변수와 전역 변수 3

6.6. 프레임과 콜 스택

프레임frame

- 함수 실행에 필요한 정보 관리
- 함수가 실행되는 동안 발생하는 지역 변수 관리
- 스택stack: 프레임의 생성과 소멸이 발생하는 메모리 영역
- 프레임은 함수가 실행되면 생성되고 함수의 실행이 종료되면 스택에서 사라짐

콜 스택

- 프레임은 생성된 순서 역순으로 사멸, 즉 스택 구조를 따름.
- 콜 스택call stack: 함수의 프레임으로 구성된 스택
- 스택 다이어그램stack diagram: 콜 스택의 변화를 표현한 다이어그램
- 참고: PythonTutor: 프레임의 생성과 사멸