5주차: 함수

5-1. 함수 만들기

용어 정리

- 함수를 사용하는 것 → “함수를 호출한다.”
- 함수 호출 시 넣는 여러가지 데이터 → 매개변수
- 함수를 호출해서 최종적으로 나오는 결과 → 리턴 값, 반환 값 등

함수의 기본

- 함수를 사용하는 이유
  - 반복적으로 사용되는 부분을 한 뭉치로 묶어서 사용
  - 프로그램의 흐름을 일목요연하게 파악 가능
- `def 함수 이름():`
  `  함수 내용`
- `return` 을 통해 결과 반환

```
# 기본적인 함수
# 매개변수와 리턴값이 존재하지 않음
def print_3_times():
    print("안녕하세요")
    print("안녕하세요")
    print("안녕하세요")

# 함수 호출
print_3_times()
```

함수에 매개변수 만들기

```
def print_n_times(value, n):
    for i in range(n):
        print(value)
```

```
print_n_times("안녕하세요", 5)
```

가변 매개변수

- 매개변수를 원하는 만큼 입력하고 싶을 때
- 가변 매개변수 뒤에는 일반 매개변수가 올 수 없음
    - 그렇지 않으면 어디까지가 가변 매개변수고, 어디가 일반 매개변수인지 구분하기 힘듬
- 가변 매개변수는 하나만 사용 가능
- 가변 매개변수의 타입은 '튜플'임
- `def 함수 이름(매개변수, 매개변수, .., *가변 매개변수):`

```
# 가변 매개변수 함수
def print_n_times(n, *values):
    # n번 반복
    for i in range(n):
        # values는 리스트처럼 활용
        for value in values:
            print(value)
        print()

# 함수를 호출
print_n_times(3, "안녕하세요", "즐거운", "파이썬 프로그래밍")
```

기본 매개변수

- 매개변수를 입력하지 않았을 경우 매개변수에 들어가는 기본값
- 기본 매개변수 뒤에는 일반 매개변수가 올 수 없음

```
def print_n_times(value, n=2):
    # n번 반복
    for i in range(n):
        print(value)
```

```
# 함수를 호출합니다.
print_n_times("안녕하세요")
```

키워드 매개변수

- 가변 매개변수와 기본 매개변수를 둘을 같이 쓴다면?

```
def print_n_times(n=2, *values):
    # n번 반복
  for i in range(n):
    for value in values:
      print(value)
    print()


# 함수 호출, 그러나 "안녕하세요"가 n에 들어감
print_n_times("안녕하세요", "즐거운", "파이썬 프로그래밍")
```

- 순서를 바꿔보자

```
def print_n_times(*values, n=2):
    # n번 반복
  for i in range(n):
    for value in values:
      print(value)
    print()

# 함수 호출, values에 모든 값이 들어감. 가변 매개변수가 우선
print_n_times("안녕하세요", "즐거운", "파이썬 프로그래밍", 3)
```

- 기본 매개변수는 가변 매개변수 앞에 써도 의미가 없다.
- 그러나, 두 가지를 함께 사용하기 위해 '키워드 매개변수' 사용
- 매개변수 이름을 지정해서 입력

```
def print_n_times(*values, n=2):
    # n번 반복
  for i in range(n):
```

```
    for value in values:
      print(value)
    print()

# 함수 호출, 그러나 "안녕하세요"가 n에 들어감
print_n_times("안녕하세요", "즐거운", "파이썬 프로그래밍", n=3)
```

리턴(반환)

- `return` : 함수를 실행했던 위치로 돌아가라는 뜻, 함수가 끝나는 위치

```
# 함수 정의
def return_test():
    print('A 위치입니다.')
    return
    print('B 위치입니다.')

# 함수 호출
return_test()
```

- 데이터 리턴하기

```
# 함수 정의
def return_test():
    return 100

# 함수 호출
value = return_test()
print(value)
```

- 아무것도 리턴하지 않는다면?

```
# 함수 정의
def return_test():
    return

# 함수 호출
```

```
value = return_test()
print(value) # None 출력
```

- `None` 은 파이썬에서 '없다'라는 의미

기본적인 함수의 활용

```
# 함수 정의
# 두 매개변수를 더하는 함수
# 두 매개변수를 받고 더한 결과를 리턴함
def add(a, b):
    return a + b

# add(3, 4)의 반환값 출력
# 함수를 호출할 때에는 지정한 매개변수와 같은 수의 매개변수를 입력
print(add(3, 4))
# add(3, 4)의 반환값을 다시 add()의 매개 변수로
print(add(1, add(2, 3)))
```

```
# 기본 매개변수와 키워드 매개변수를 활용해 범위의 정수를 더하는 함수
# 함수 선언
# start부터 end까지 step만큼 더하며 범위의 합을 구함
def sum_all(start=0, end=100, step=1):
    # 변수 선언
    output = 0
    # 반복문을 돌려 숫자를 더함
    for i in range(start, end + 1, step):
        output += i
    # 리턴
    return output

# 함수 호출
print("A. ", sum_all(0, 100, 10))
print("B. ", sum_all(end=50))
print("C. ", sum_all(end=40, step=5))
```

5-2. 함수의 활용

재귀 함수

- 팩토리얼: $n! = n * (n-1) * (n-2) * \cdots * 1$
  - https://www.acmicpc.net/problem/10872 (팩토리얼)
- 반복문으로 팩토리얼 구하기

```
# 함수 선언
def factorial(n):
    output = 1
    for i in range(1, n + 1):
        output *= i
    return output

# 함수 호출
print("1!:", factorial(1))
print("2!:", factorial(2))
print("3!:", factorial(3))
print("4!:", factorial(4))
print("5!:", factorial(5))
```

- 재귀 함수로 팩토리얼 구하기
  - 재귀(recursion): 자기 자신을 호출하는 것
- $n! = n * (n-1)!$
- `factorial(n) = n * factorial(n - 1)` (n≥ 1 일 때)
- `factorial(0) = 1`

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

# 함수 호출
print("1!:", factorial(1))
print("2!:", factorial(2))
print("3!:", factorial(3))
```

```
print("4!:", factorial(4))
print("5!:", factorial(5))
```

재귀 함수의 문제

- 상황에 따라 같은 것을 기하급수적으로 많이 반복하는 문제
- 이 문제를 해결하기 위한 메모화(memoization) 기술
- 피보나치 수열 예제
  - https://www.acmicpc.net/problem/2747

```
def fibonacci(n):
    if n == 1:
        return 1
    if n == 2:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

# 함수 호출
print('fibonacci(1)', fibonacci(1))
print('fibonacci(2)', fibonacci(1))
print('fibonacci(3)', fibonacci(1))
print('fibonacci(4)', fibonacci(1))
print('fibonacci(5)', fibonacci(1))
```

- 너무 오래 걸림. 문제 확인을 위해 수정한 아래 코드

```
# 함수의 호출 횟수를 파악해보자
counter = 0

def fibonacci(n):
    # 어떤 피보나치 수를 구하는지 출력
    print("fibonacci({})를 구합니다".format(n))
    # global 예약어를 통해 전역변수인 counter에 접근
    global counter
    counter += 1
```

```
    # 피보나치 수 구하기
    if n == 1:
        return 1
    if n == 2:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

# 함수 호출
print('fibonacci(10)', fibonacci(10))
print('---')
print('fibonacci(10) 계산에 활용된 함수 호출 횟수는',counter,'번입니다.')
```

메모화(memoization)

```
list = [0 for i in range(46)]

counter = 0
def fibonacci(n):
    if list[n] != 0:
        return list[n]
    else:
        # 어떤 피보나치 수를 구하는지 출력
        print("fibonacci({})를 구합니다".format(n))
        # global 예약어를 통해 전역변수인 counter에 접근
        global counter
        counter += 1

        # 피보나치 수 구하기
        if n == 1:
            list[n] = 1
            return 1
        if n == 2:
            list[n] = 1
            return 1
        else:
            list[n] = fibonacci(n - 1) + fibonacci(n - 2)
            return list[n]
```

```
# 함수 호출
print('fibonacci(10)', fibonacci(10))
print('---')
print('fibonacci(10) 계산에 활용된 함수 호출 횟수는',counter,'번입니다.')
```

조기 리턴

- 위 예시처럼 `return` 문을 중간에 사용하는 형태

```
list = [0 for i in range(46)]

counter = 0
def fibonacci(n):
    if list[n] != 0:
        return list[n]
    # 조기에 return 되므로 else문을 사용할 필요가 없음

    # 어떤 피보나치 수를 구하는지 출력
    print("fibonacci({})를 구합니다".format(n))
    # global 예약어를 통해 전역변수인 counter에 접근
    global counter
    counter += 1

    # 피보나치 수 구하기
    if n == 1:
        list[n] = 1
        return 1
    if n == 2:
        list[n] = 1
        return 1
    # 조기에 return 되므로 else문을 사용할 필요가 없음
    list[n] = fibonacci(n - 1) + fibonacci(n - 2)
    return list[n]

# 함수 호출
print('fibonacci(10)', fibonacci(10))
```

```
print('---')
print('fibonacci(10) 계산에 활용된 함수 호출 횟수는',counter,'번입니다.')
```

5-3. 내장 함수와 표준 모듈

내장 함수

- `abs(n)` : `n` 의 절댓값 리턴
- `all(iterable)` : `iterable` 한 자료형을 받아 모든 원소가 참이면 `True` , 아니면 `False` 리턴
- `any(iterable)` : `iterable` 한 자료형을 받아 원소가 하나라도 참이면 `True` , 아니면 `False` 리턴
- `chr(n)` : 아스키 코드 값을 받아 그 코드에 해당하는 문자 출력
- `ord(c)` : 문자의 아스키 코드 값을 반환
- `input()` : 사용자 입력을 한 줄 입력받아 문자열로 반환
- `int(x)` : `x` 를 정수형태로 반환
- `list(s)` : 반복 가능한 자료형 `s` 를 리스트로 반환
- `map(f, iterable)` : 입력받은 자료형 `iterable` 의 각 요소를 함수 f가 수행한 결과를 묶어서 반환
- `max(iterable)` : 반복 가능한 자료형을 입력받아 그 최댓값 반환
- `min(iterable)` : 반복 가능한 자료형을 입력받아 그 최솟값 반환
- `pow(x, y)` : x의 y제곱을 반환
- `round(number[, ndigits])` : 숫자를 입력받아 반올림
  - `round(4.6)`
  - `round(4.678, 2)`
- `sorted(iterable)` : 입력 값을 정렬한 후 그 결과를 리스트로 반환
- `str(object)` : 문자열 형태로 객체를 변환하여 반환
- `sum(iterable)` : 입력 변수의 모든 요소의 합을 반환
- `type(object)` : 입력 변수의 자료형이 무엇인지 반환

표준 모듈

- 모듈(module): 여러 변수와 함수를 가지고 있는 집합체

- 표준 모듈과 외부 모듈로 나눔
- `import 모듈 이름`
- 파이썬 공식 문서 참고
  - https://docs.python.org/library/index.html

`math` 모듈

```
import math

# 사인, 코사인, 탄젠트
print(math.sin(1))
print(math.cos(1))
print(math.tan(1))

# 내림
print(math.floor(2.5))
# 올림
print(math.ceil(2.5))
```

`from` 구문

- `from 모듈 이름 import 가져오고 싶은 변수 또는 함수`

```
from math import sin, cos, tan, floor, ceil

# 사인, 코사인, 탄젠트
print(sin(1))
print(cos(1))
print(tan(1))

# 내림
print(floor(2.5))
# 올림
print(ceil(2.5))
```

`as` 구문

- `import 모듈 as 사용하고 싶은 식별자`

```
import math as m
# 사인, 코사인, 탄젠트
print(m.sin(1))
print(m.cos(1))
print(m.tan(1))

# 내림
print(m.floor(2.5))
# 올림
print(m.ceil(2.5))
```