가변 매개변수

앞서 살펴본 함수는 함수를 선언할 때의 매개변수와 함수를 호출할 때의 매개변수가 같아야 했습니다. 적어도 안 되고, 많아도 안 됩니다. 그러나 여러분들이 가장 많이 사용해 왔던 print() 함수는 매개변수를 원하는 만큼 입력할 수 있었습니다. 이처럼 매개변수를 원하는 만큼 받을 수 있는 함수를 **가변 매개변수** 함수라고 부르는데, 여기에서 가변 매개변수란 매개변수 개수가 변할 수 있다는 의미입니다.

가변 매개변수 함수는 다음과 같은 형태로 만듭니다.

```
def 함수 이름(매개변수, 매개변수, ..., *가변 매개변수):
    문장
```

가변 매개변수를 사용할 때는 다음과 같은 제약이 있습니다.

- 가변 매개변수 뒤에는 일반 매개변수가 올 수 없습니다.
- 가변 매개변수는 하나만 사용할 수 있습니다.

약간 어렵게 느껴질 수 있지만, 이러한 제약이 없으면 가변 매개변수가 어디부터 어디까지인지 알 수 없기 때문에 만들어진 규칙입니다. 가변 매개변수 함수를 만들어 보겠습니다.

직접 해보는 손코딩

가변 매개변수 함수 소스 코드 variable_param.py

```
01  def print_n_times(n, *values):
02      # n번 반복합니다.
03      for i in range(n):
04          # values는 리스트처럼 활용합니다.
05          for value in values:
06              print(value)
07          # 단순한 줄바꿈
08          print()
09
10  # 함수를 호출합니다.
11  print_n_times(3, "안녕하세요", "즐거운", "파이썬 프로그래밍")
```

실행 결과

```
안녕하세요
즐거운
파이썬 프로그래밍

안녕하세요
즐거운
파이썬 프로그래밍

안녕하세요
즐거운
파이썬 프로그래밍
```

일단 가변 매개변수 뒤에는 일반 매개변수가 올 수 없다고 했습니다. 만약 print_n_times("안녕하세요", "즐거운", "파이썬 프로그래밍", 3)처럼 사용할 수 있다고 하면 어디까지가 가변 매개변수고, 어디가 매개변수 n인지 구분하기 힘듭니다. 따라서 파이썬 프로그래밍 언어는 내부적으로 가변 매개변수 뒤에 일반 매개변수가 오지 못하게 막은 것입니다.

그래서 매개변수 n을 앞으로 옮기고 매개변수 *values를 뒤로 밀었습니다. 가변 매개변수 *values는 리스트처럼 사용하면 됩니다. 예제에서는 반복문을 두 번 사용해서 쭉 출력하게 만들었습니다.

기본 매개변수

print() 함수의 자동 완성 기능으로 나오는 설명을 다시 적어 보면 다음과 같습니다.

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

가장 앞에 있는 value가 바로 '가변 매개변수'입니다. 앞에서 가변 매개변수 뒤에는 일반 매개변수가 올 수 없다고 했는데 매개변수가 왔습니다. 그런데 뭔가 특이하게 '매개변수=값' 형태로 되어 있습니다. 이는 **기본 매개변수**라고 부르며, 매개변수를 입력하지 않았을 경우 매개변수에 들어가는 기본값입니다. 기본 매개변수도 다음과 같은 제약이 있습니다.

- 기본 매개변수 뒤에는 일반 매개변수가 올 수 없습니다.

예제를 통해 살펴볼까요?

직접 해보는 손코딩

기본 매개변수 소스 코드 default_param.py

```
01  def print_n_times(value, n=3):
02      # n번 반복합니다.
03      for i in range(n):
04          print(value)
05
06  # 함수를 호출합니다.
07  print_n_times("안녕하세요")
```

실행 결과

```
안녕하세요
안녕하세요
안녕하세요
```

276쪽의 〈직접 해보는 손코딩param_basic.py〉을 수정한 것인데, 매개변수 n을 n=2라는 형태로 입력했습니다. n을 입력하지 않을 경우 기본값이 2로 들어갑니다. 그래서 코드를 실행하면 "안녕하세요"라는 문자열을 두 번 출력합니다.

✚ 여기서 잠깐 기본 매개변수 뒤에 일반 매개변수를 오지 못하게 막은 이유

만약 print_n_times(n=2, value) 형태로 사용할 수 있다면 print_n_times("안녕하세요")라고 입력했을 때 "안녕하세요"라는 글자가 첫 번째 매개변수에 할당되어야 하는지, 두 번째 매개변수에 할당되어야 하는지 확실하게 알 수 없습니다. 그래서 파이썬 프로그래밍 언어는 내부적으로 기본 매개변수 뒤에 일반 매개변수가 오지 못하게 막은 것입니다.

키워드 매개변수

지금까지의 설명을 보면서 가변 매개변수와 기본 매개변수 둘을 같이 써도 되는지 궁금하지 않았나요? 상황을 나누어 살펴보겠습니다.

기본 매개변수가 가변 매개변수보다 앞에 올 때

기본 매개변수가 가변 매개변수보다 앞에 올 때는 기본 매개변수의 의미가 사라집니다. 다음 코드의 실행 결과를 예측해 봅시다. n에는 무엇이 들어갈까요?

```
def print_n_times(n=2, *values):
    # n번 반복합니다.
    for i in range(n):
        # values는 리스트처럼 활용합니다.
        for value in values:
            print(value)
        # 단순한 줄바꿈
        print()

# 함수를 호출합니다.
print_n_times("안녕하세요", "즐거운", "파이썬 프로그래밍")
```

매개변수가 순서대로 입력되므로 n에는 "안녕하세요"가 들어가고, values에는 ["즐거운", "파이썬 프로그래밍"]이 들어옵니다. 그런데 range() 함수의 매개변수에는 숫자만 들어올 수 있으므로 다음과 같은 오류가 발생합니다.

오류

```
Traceback (most recent call last):
  File "test5_03.py", line 11, in <module>
    print_n_times("안녕하세요", "즐거운", "파이썬 프로그래밍")
  File "test.py", line 3, in print_n_times
    for i in range(n):
TypeError: 'str' object cannot be interpreted as an integer
```

따라서 기본 매개변수는 가변 매개변수 앞에 써도 의미가 없다는 것을 기억해 주세요.

가변 매개변수가 기본 매개변수보다 앞에 올 때

그러면 반대로 가변 매개변수가 기본 매개변수보다 앞에 올 때는 어떻게 될까요? 다음 코드의 실행 결과를 예측해 봅시다.

```
def print_n_times(*values, n=2):
    # n번 반복합니다.
    for i in range(n):
        # values는 리스트처럼 활용합니다.
        for value in values:
            print(value)
        # 단순한 줄바꿈
        print()

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

다음과 같은 두 가지 예측을 할 수 있습니다.

- ["안녕하세요", "즐거운", "파이썬 프로그래밍"]을 세 번 출력합니다.
- ["안녕하세요", "즐거운", "파이썬 프로그래밍", 3]을 두 번 출력합니다.

코드를 실행하면 두 번째 예상대로 실행됩니다. 가변 매개변수가 우선되는 것입니다.

```
안녕하세요
즐거운
파이썬 프로그래밍
3

안녕하세요
즐거운
파이썬 프로그래밍
3
```

그렇다면 두 가지를 함께 사용할 수 있는 방법은 없을까요? 파이썬 프로그래밍 언어는 이런 상황에 대비해서 **키워드 매개변수**라는 기능을 만들었습니다.

키워드 매개변수

다시 print() 함수의 기본 형태를 살펴보겠습니다.

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

value를 여러 개 입력할 수 있으므로 가변 매개변수를 앞에 두고, 뒤에 기본 매개변수들이 들어가 있는 형태입니다. 이러한 기본 매개변수가 지정된 함수를 사용할 때는 다음과 같이 사용합니다. 240쪽의 〈while 반복문〉을 살펴볼 때 짧게 보았던 코드infinite_loop.py인데, 매개변수 이름을 직접적으로 지정해서 값을 입력합니다.

```
# while 반복문을 사용합니다.
while True:
    # "."을 출력합니다.
    # 기본적으로 end가 "\n"이라 줄바꿈이 일어나는데
    # 빈 문자열 ""로 바꿔서 줄바꿈이 일어나지 않게 합니다.
    print(".", end="") ⟶ 키워드 매개변수입니다.
```

따라서 이전 코드에서 ["안녕하세요", "즐거운", "파이썬 프로그래밍"]을 세 번 출력하도록 실행하려면 다음과 같이 매개변수 이름을 직접적으로 지정해서 값을 입력합니다.

직접 해보는 손코딩

키워드 매개변수 소스 코드 param_keyword01.py

```
01  def print_n_times(*values, n=2):
02      # n번 반복합니다.
03      for i in range(n):
04          # values는 리스트처럼 활용합니다.
05          for value in values:
06              print(value)
07          # 단순한 줄바꿈
08          print()
09
10  # 함수를 호출합니다.
11  print_n_times("안녕하세요", "즐거운", "파이썬 프로그래밍", n=3)
```

실행 결과

```
안녕하세요
즐거운
파이썬 프로그래밍

안녕하세요
즐거운
파이썬 프로그래밍

안녕하세요
즐거운
파이썬 프로그래밍
```

이처럼 매개변수 이름을 지정해서 입력하는 매개변수를 **키워드 매개변수**라고 부릅니다.

기본 매개변수 중에서 필요한 값만 입력하기

키워드 매개변수는 기본 매개변수들로 구성된 함수에서도 많이 사용됩니다. 다음 코드의 실행 결과로 자세히 살펴보겠습니다.

직접 해보는 손코딩

여러 함수 호출 형태 소스 코드 param_examples.py

```
01  def test(a, b=10, c=100):
02      print(a + b + c)
03
04  # 1) 기본 형태
05  test(10, 20, 30)
06  # 2) 키워드 매개변수로 모든 매개변수를 지정한 형태
07  test(a=10, b=100, c=200)
08  # 3) 키워드 매개변수로 모든 매개변수를 마구잡이로 지정한 형태
09  test(c=10, a=100, b=200)
10  # 4) 키워드 매개변수로 일부 매개변수만 지정한 형태
11  test(10, c=200)
```

실행 결과

```
60
310
310
220
```

일단 첫 번째 매개변수 a는 일반 매개변수이므로 해당 위치에 반드시 입력해야 합니다(1번 형태). 일반 매개변수이지만 키워드 매개변수처럼 사용할 수도 있습니다(2번과 3번 형태).

8행의 3번 형태를 보면 조금 이상합니다. 매개변수의 순서가 마구잡이로 쓰여 있습니다. 키워드를 지정해서 매개변수를 입력하는 경우에는 이처럼 매개변수 순서를 원하는 대로 입력할 수 있습니다.

10행의 4번은 b를 생략한 형태입니다. 이렇게 키워드 매개변수를 사용하면 필요한 매개변수에만 값을 전달할 수 있습니다.

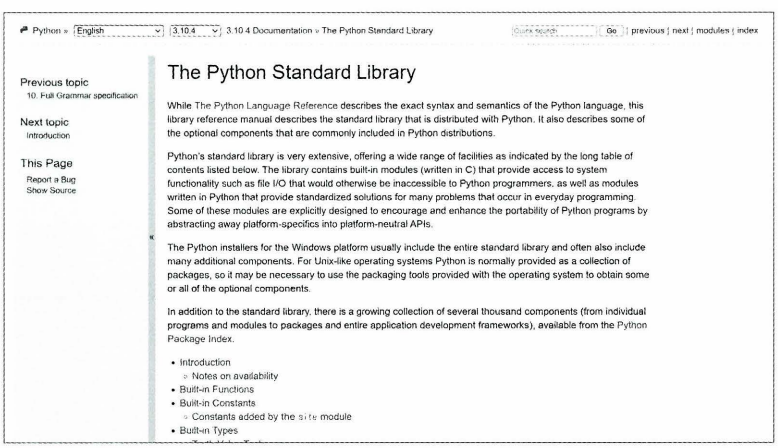
즉 '일반 매개변수'는 필수로 입력합니다. 순서에 맞게 입력하면 됩니다. '기본 매개변수'는 필요한 것만 키워드를 지정해서 입력하는 경우가 많습니다.

지금까지 매개변수의 다양한 형태에 대해서 살펴보았습니다. 이러한 것들은 내가 직접 함수를 만들 때도 필요하지만, 다른 사람들이 만든 함수를 살펴볼 때도 유용합니다.

✚ 여기서 잠깐 파이썬 라이브러리 문서 284하

파이썬 공식 홈페이지에서 제공하는 라이브러리의 문서를 틈틈이 읽어 보시기 바랍니다.

- 파이썬 라이브러리 문서 URL https://docs.python.org/3/library/index.html



홈페이지에 접속해서 몇 가지 내용을 뽑아 보았습니다. 무엇을 하는 함수인지 구체적으로는 몰라도 어떤 형태로 매개변수를 입력하면 되는지 이해할 수 있을 것입니다.

- send_error(code, message=None, explain=None)
- timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0)
- urllib.parse.urlsplit(urlstring, scheme='', allow_fragments=True)
- urllib.parse.urljoin(base, url, allow_fragments=True)