UNIT 30

함수에서 위치 인수와 키워드 인수 사용하기

30.1 위치 인수와 리스트 언패킹 사용하기

위치 인수와 리스트 언패킹 사용하기

- 다음과 같이 함수에 인수를 순서대로 넣는 방식을 위치 인수(positional argument)라고 함
- 즉, 인수의 위치가 정해져 있음

```
>>> print(10, 20, 30)
10 20 30
```

30.1 위치 인수와 리스트 언패킹 사용하기

위치 인수를 사용하는 함수를 만들고 호출하기

- 그럼 간단하게 숫자 세 개를 각 줄에 출력하는 함수를 만들어보자

```
>>> def print_numbers(a, b, c):
...     print(a)
...     print(b)
...     print(c)
...
```

```
>>> print_numbers(10, 20, 30)
10
20
30
```

30.1 위치 인수와 리스트 언패킹 사용하기

언패킹 사용하기

- 인수를 순서대로 넣을 때는 리스트나 튜플을 사용할 수도 있음
- 다음과 같이 리스트 또는 튜플 앞에 *(애스터리스크)를 붙여서 함수에 넣어주면 됨

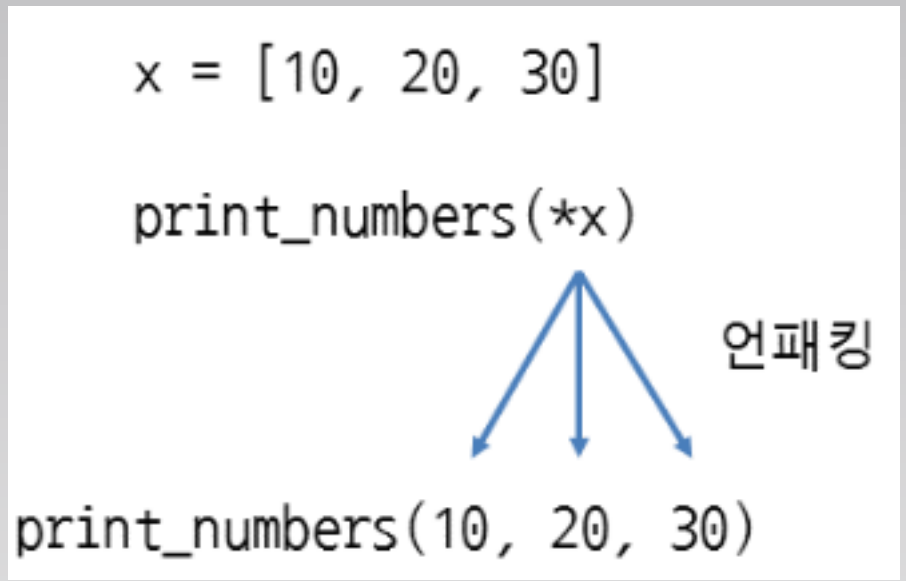
- 함수(*리스트)
- 함수(*튜플)

```
>>> x = [10, 20, 30]
>>> print_numbers(*x)
10
20
30
```

- 리스트(튜플) 앞에 *를 붙이면 언패킹(unpacking)이 되어서 print_numbers(10, 20, 30)과 똑같은 동작이 됨
- 말 그대로 리스트의 포장을 푼다는 뜻임

30.1 위치 인수와 리스트 언패킹 사용하기

▼ 그림 리스트 언패킹



30.1 위치 인수와 리스트 언패킹 사용하기

언패킹 사용하기

- 리스트 변수 대신 리스트 앞에 바로 *를 붙여도 동작은 같음

```
>>> print_numbers(*[10, 20, 30])
10
20
30
```

- 단, 이때 함수의 매개변수 개수와 리스트의 요소 개수는 같아야 함
- 만약 개수가 다르면 함수를 호출할 수 없음
- 여기서는 함수를 def print_numbers(a, b, c):로 만들었으므로 리스트에는 요소를 3개 넣어야 함
- 다음과 같이 요소가 두 개인 리스트를 넣으면 에러가 발생함

```
>>> print_numbers(*[10, 20])
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    print_numbers(*[10, 20])
TypeError: print_numbers() missing 1 required positional argument: 'c'
```

30.1 위치 인수와 리스트 언패킹 사용하기

가변 인수 함수 만들기

- 위치 인수와 리스트 언패킹은 인수의 개수가 정해지지 않은 가변 인수(variable argument)에 사용함
- 다음과 같이 가변 인수 함수는 매개변수 앞에 *를 붙여서 만듬

```
def 함수이름(*매개변수):
    코드
```

```
>>> def print_numbers(*args):
...     for arg in args:
...         print(arg)
...
```

30.1 위치 인수와 리스트 언패킹 사용하기

가변 인수 함수 만들기

- 넣은 숫자 개수만큼 출력됨

```
>>> print_numbers(10)
10
>>> print_numbers(10, 20, 30, 40)
10
20
30
40
```

- 이렇게 함수에 인수 여러 개를 직접 넣어도 되고, 리스트(튜플) 언패킹을 사용해도 됨
- 다음과 같이 숫자가 들어있는 리스트를 만들고 앞에 *를 붙여서 넣어봄

```
>>> x = [10]
>>> print_numbers(*x)
10
>>> y = [10, 20, 30, 40]
>>> print_numbers(*y)
10
20
30
40
```

30.2 키워드 인수 사용하기

키워드 인수 사용하기

- 예를 들어 개인 정보를 출력하는 함수를 만들어보자

```
>>> def personal_info(name, age, address):
...     print('이름: ', name)
...     print('나이: ', age)
...     print('주소: ', address)
...
```

```
>>> personal_info('홍길동', 30, '서울시 용산구 이촌동')
이름:  홍길동
나이:  30
주소:  서울시 용산구 이촌동
```

30.2 키워드 인수 사용하기

키워드 인수 사용하기

- 파이썬에서는 인수의 순서와 용도를 매번 기억하지 않도록 키워드 인수(keyword argument)라는 기능을 제공함

- **함수(키워드=값)**

- 그럼 personal_info 함수를 키워드 인수 방식으로 호출해보자

```
>>> personal_info(name='홍길동', age=30, address='서울시 용산구 이촌동')
이름:  홍길동
나이:  30
주소:  서울시 용산구 이촌동
```

- 키워드 인수를 사용하니 함수를 호출할 때 인수의 용도가 명확하게 보임
- 키워드 인수를 사용하면 인수의 순서를 맞추지 않아도 키워드에 해당하는 값이 들어감

```
>>> personal_info(age=30, address='서울시 용산구 이촌동', name='홍길동')
이름:  홍길동
나이:  30
주소:  서울시 용산구 이촌동
```

- 참고로 print 함수에서 사용했던 sep, end도 키워드 인수

```
print(10, 20, 30, sep=':', end='')
```

30.3 키워드 인수와 딕셔너리 언패킹 사용하기

» 키워드 인수와 딕셔너리 언패킹 사용하기

- 다음과 같이 딕셔너리 앞에 **(애스터리스크 두 개)를 붙여서 함수에 넣어줌

• 함수(**딕셔너리)

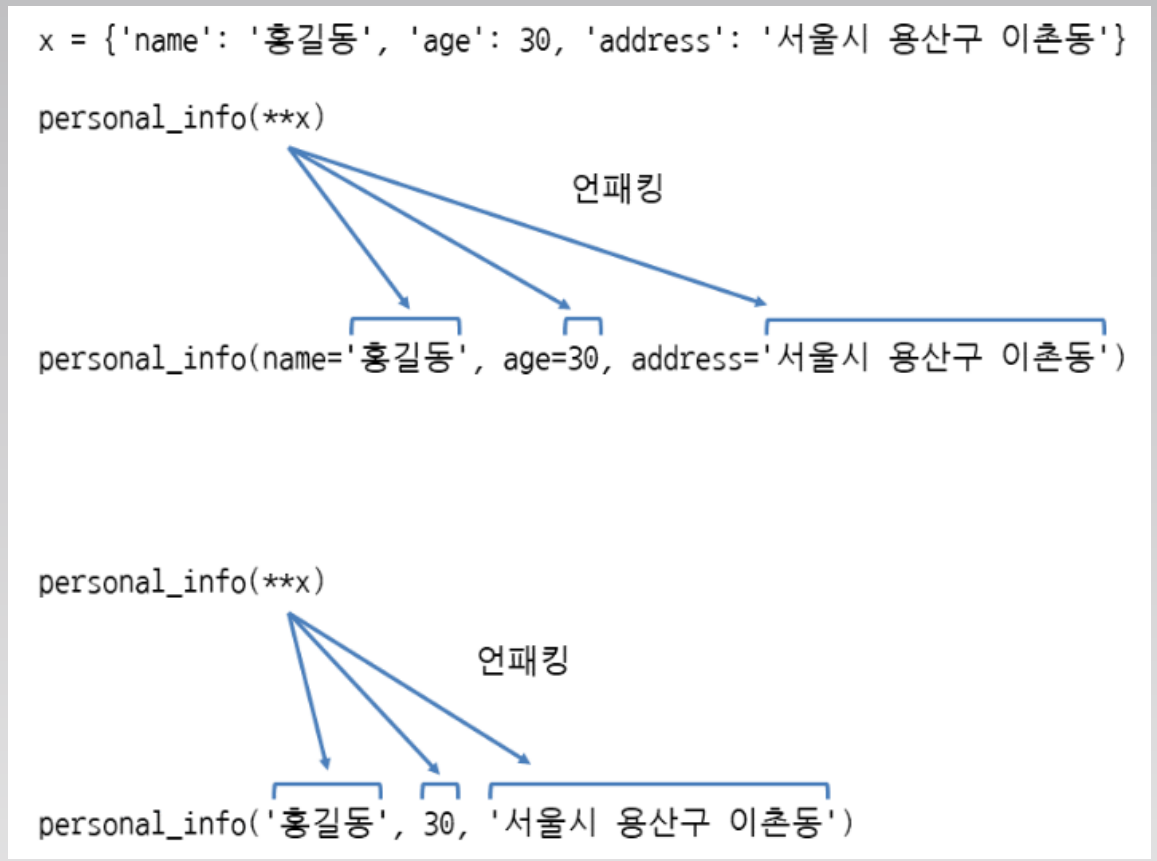
```
>>> def personal_info(name, age, address):
...     print('이름: ', name)
...     print('나이: ', age)
...     print('주소: ', address)
...
```

- 딕셔너리에 '키워드': 값 형식으로 인수를 저장하고, 앞에 **를 붙여서 함수에 넣어줌
- 딕셔너리의 키워드(키)는 반드시 문자열 형태라야 함

```
>>> x = {'name': '홍길동', 'age': 30, 'address': '서울시 용산구 이촌동'}
>>> personal_info(**x)
이름:  홍길동
나이:  30
주소:  서울시 용산구 이촌동
```

30.3 키워드 인수와 딕셔너리 언패킹 사용하기

▼ 그림 딕셔너리 언패킹



30.3 키워드 인수와 딕셔너리 언패킹 사용하기

» 키워드 인수와 딕셔너리 언패킹 사용하기

- 딕셔너리 변수 대신 딕셔너리 앞에 바로 **를 붙여도 동작은 같음

```
>>> personal_info(**{'name': '홍길동', 'age': 30, 'address': '서울시 용산구 이촌동'})
이름:  홍길동
나이:  30
주소:  서울시 용산구 이촌동
```

- 딕셔너리 언패킹을 사용할 때는 함수의 매개변수 이름과 딕셔너리의 키 이름이 같아야 함
- 매개변수 개수와 딕셔너리 키의 개수도 같아야 함
- 만약 이름과 개수가 다르면 함수를 호출할 수 없음
- 함수를 def personal_info(name, age, address):로 만들었으므로 딕셔너리도 똑같이 맞춰주어야 함

30.3 키워드 인수와 딕셔너리 언패킹 사용하기

키워드 인수와 딕셔너리 언패킹 사용하기

- 다음과 같이 매개변수 이름, 개수가 다른 딕셔너리를 넣으면 에러가 발생함

```
>>> personal_info(**{'name': '홍길동', 'old': 30, 'address':'서울시 용산구 이촌동'})
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: personal_info() got an unexpected keyword argument 'old'
>>> personal_info(**{'name': '홍길동', 'age': 30})
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: personal_info() missing 1 required positional argument: 'address'
```

30.3 키워드 인수와 딕셔너리 언패킹 사용하기

**를 두 번 사용하는 이유

- 딕셔너리는 **처럼 *를 두 번 사용하는 이유는 딕셔너리는 키-값 쌍 형태로 값이 저장되어 있기 때문임
- 먼저 *를 한 번만 사용해서 함수를 호출해봄

```
>>> x = {'name': '홍길동', 'age': 30, 'address': '서울시 용산구 이촌동'}
>>> personal_info(*x)
이름:  name
나이:  age
주소:  address
```

- 딕셔너리를 한 번 언패킹하면 키를 사용한다는 뜻이 됨
- **처럼 딕셔너리를 두 번 언패킹하여 값을 사용하도록 만들어야 함

```
>>> x = {'name': '홍길동', 'age': 30, 'address': '서울시 용산구 이촌동'}
>>> personal_info(**x)
이름:  홍길동
나이:  30
주소:  서울시 용산구 이촌동
```

30.3 키워드 인수와 딕셔너리 언패킹 사용하기

키워드 인수를 사용하는 가변 인수 함수 만들기

- 다음과 같이 키워드 인수를 사용하는 가변 인수 함수는 매개변수 앞에 **를 붙여서 만듦

```
def 함수이름(**매개변수):
    코드
```

```
>>> def personal_info(**kwargs):
...     for kw, arg in kwargs.items():
...         print(kw, ': ', arg, sep='')
...
```

- 매개변수 이름은 원하는 대로 지어도 되지만 관례적으로 keyword arguments를 줄여서 kwargs로 사용함
- kwargs는 딕셔너리라서 for로 반복할 수 있음

30.3 키워드 인수와 딕셔너리 언패킹 사용하기

키워드 인수를 사용하는 가변 인수 함수 만들기

- 그럼 personal_info 함수에 키워드와 값을 넣어서 실행해보고 값을 한 개 넣어도 되고, 세 개 넣어도 됨

```
>>> personal_info(name='홍길동')
name: 홍길동
>>> personal_info(name='홍길동', age=30, address='서울시 용산구 이촌동')
name: 홍길동
age: 30
address: 서울시 용산구 이촌동
```

- 인수를 직접 넣어도 되고, 딕셔너리 언패킹을 사용해도 됨

```
>>> x = {'name': '홍길동'}
>>> personal_info(**x)
name: 홍길동
>>> y = {'name': '홍길동', 'age': 30, 'address': '서울시 용산구 이촌동'}
>>> personal_info(**y)
name: 홍길동
age: 30
address: 서울시 용산구 이촌동
```

30.3 키워드 인수와 딕셔너리 언패킹 사용하기

키워드 인수를 사용하는 가변 인수 함수 만들기

- 함수를 만들 때 def personal_info(**kwargs):와 같이 매개변수에 **를 붙여주면 키워드 인수를 사용하는 가변 인수 함수를 만들 수 있음
- 함수를 호출할 때는 키워드와 인수를 각각 넣거나 딕셔너리 언패킹을 사용하면 됨
- 보통 **kwargs를 사용한 가변 인수 함수는 다음과 같이 함수 안에서 특정 키가 있는지 확인한 뒤 해당 기능을 만듦

```
def personal_info(**kwargs):
    if 'name' in kwargs:    # in으로 딕셔너리 안에 특정 키가 있는지 확인
        print('이름: ', kwargs['name'])
    if 'age' in kwargs:
        print('나이: ', kwargs['age'])
    if 'address' in kwargs:
        print('주소: ', kwargs['address'])
```

30.4 매개변수에 초깃값 지정하기

매개변수에 초깃값 지정하기

```
def 함수이름(매개변수=값):
    코드
```

- 매개변수의 초깃값은 주로 사용하는 값이 있으면서 가끔 다른 값을 사용해야 할 때 활용함

```
>>> def personal_info(name, age, address='비공개'):
...     print('이름: ', name)
...     print('나이: ', age)
...     print('주소: ', address)
...
```

```
>>> personal_info('홍길동', 30)
이름:  홍길동
나이:  30
주소:  비공개
```

```
>>> personal_info('홍길동', 30, '서울시 용산구 이촌동')
이름:  홍길동
나이:  30
주소:  서울시 용산구 이촌동
```

30.4 매개변수에 초깃값 지정하기

초깃값이 지정된 매개변수의 위치

- 매개변수의 초깃값을 지정할 때 한 가지 주의할 점은 초깃값이 지정된 매개변수 다음에는 초깃값이 없는 매개변수가 올 수 없음
- 이번에는 address를 두 번째 매개변수로 만들고, 그 다음에 초깃값을 지정하지 않은 age가 오도록 만들어보자

```
>>> def personal_info(name, address='비공개', age):
...     print('이름: ', name)
...     print('나이: ', age)
...     print('주소: ', address)
...
  File "<stdin>", line 1
SyntaxError: non-default argument follows default argument
```

- 함수를 만들어보면 문법 에러가 발생함
- 다음과 같이 초깃값이 지정된 매개변수는 뒤쪽에 몰아주면 됨

```
def personal_info(name, age, address='비공개'):
def personal_info(name, age=0, address='비공개'):
def personal_info(name='비공개', age=0, address='비공개'):
```