

COMP  
110

# Positional Arguments

## Recall: Signature vs Call

```
def sum(num1: int, num2: int) -> int:
```

```
sum(num1 = 11, num2 = 3)
```


The diagram consists of two white double-headed arrows. The first arrow connects the parameter name 'num1' in the function signature to the value '11' in the function call. The second arrow connects the parameter name 'num2' in the function signature to the value '3' in the function call.

These are called **keyword arguments**, since you are assigning the variables based on the parameter names.

## Recall: Signature vs Call

```
def sum(num1: int, num2: int) -> int:
```

```
sum(num1 = 11, num2 = 3)
```



Benefit of this approach: order doesn't matter.

## Recall: Signature vs Call

```
def sum(num1: int, num2: int) -> int:
```



```
sum(num1 = 11, num2 = 3)
```


```
sum(num2 = 3, num1 = 11)
```

Benefit of this approach: order doesn't matter.

# Positional Arguments

```
def sum(num1: int, num2: int) -> int:
```

```
sum(11, 3)
```

Two white arrows originate from the function call 'sum(11, 3)'. One arrow points from the first argument '11' to the parameter 'num1' in the function definition. The other arrow points from the second argument '3' to the parameter 'num2' in the function definition.

For **positional arguments**, variables are assigned value based on the order (or position) of the arguments.