Keyword Arguments

Recall: positional parameters defined in functions can also be passed as named (keyword) arguments.

```
def func1(a, b, c):
    print(a, b, c)

func1(10, 20, 30)
10 20 30

func1(b=20, c=30, a=10)
10 20 30

func1(10, c=30, b=20)
10 20 30
```

Using a named argument is optional and up to the caller.

What if we wanted to force calls to our function to use named arguments?

We can do so by **exhausting** all the positional arguments, and then adding some additional parameters in teh function definition:

```
def func1(a, b, *args, d):
    print(a, b, args, d)
```

Now we will need at least two positional arguments, an optional (possibly even zero) number of additional arguments, and this extra argument which is supposed to go into **d*. *This argument can *only*** be passed to the function using a named (keyword) argument:

So, this will not work:

```
func1(10, 20, 'a', 'b', 100)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-50a3343cf093> in <module>()
----> 1 func1(10, 20, 'a', 'b', 100)

TypeError: func1() missing 1 required keyword-only argument: 'd'
```

But this will:

```
func1(10, 20, 'a', 'b', d=100)
```

As you can see, **d** took the keyword argument, while the remaining arguments were handled as positional parameters.

We can even define a function that has only optional positional arguments and mandatory keyword arguments:

```
def func1(*args, d):
    print(args)
    print(d)

func1(1, 2, 3, d='hello')
```

We can of course, not pass any positional arguments:

```
func1(d='hello')
```

but the positional argument is mandatory (since no default was provided in the function definition):

```
func1()
```

To make the keyword argument optional, we just need to specify a default value in the function definition:

```
def func1(*args, d='n/a'):
    print(args)
    print(d)

func1(1, 2, 3)

func1()
```

Sometimes we want **only** keyword arguments, in which case we still have to exhaust the positional arguments first - but we can use the following syntax if we do not want any positional parameters passed in:

```
def func1(*, d='hello'):
    print(d)

func1(10, d='bye')

func1(d='bye')
```

Of course, if we do not provide a default value for the keyword argument, then we effectively are forcing the caller to provide the keyword argument:

```
def func1(*, a, b):
    print(a)
    print(b)

func1(a=10, b=20)
```

but, the following would not work:

```
func1(10, 20)
```

Unlike positional parameters, keyword arguments do not have to be defined with non-defaulted and then defaulted arguments:

```
def func1(a, *, b='hello', c):
    print(a, b, c)

func1(5, c='bye')
```

We can also include positional non-defaulted (first), positional defaulted (after positional non-defaulted) followed lastly (after exhausting positional arguments) by keyword args (defaulted or non-defaulted in any order)

```
def func1(a, b=20, *args, d=0, e='n/a'):
    print(a, b, args, d, e)

func1(5, 4, 3, 2, 1, d=0, e='all engines running')

func1(0, 600, d='goooood morning', e='python!')

func1(11, 'm/s', 24, 'mph', d='unladen', e='swallow')
```

As you can see, defining parameters and passing arguments is extremely flexible in Python! Even more so, when you account for the fact that the parameters are not statically typed!

In the next video, we'll look at one more thing we can do with function parameters!