KEYWORD ARGUMENTS

Keyword Arguments

Recall that positional parameters can, optionally be passed as named (keyword) arguments

```
def func(a, b, c):
    # code
```

```
func(1, 2, 3)          →   a = 1, b = 2, c = 3

func(a=1, c=3, b=2)    →   a = 1, b = 2, c = 3
```

Using named arguments in this case is entirely up to the caller.

Mandatory Keyword Arguments

We can make keyword arguments mandatory.

To do so, we create parameters after the positional parameters have been exhausted.

```
def func(a, b, *args, d):
    #code
```

In this case, `*args` effectively exhausts all positional arguments

and d must be passed as a keyword (named) argument

```
func(1, 2, 'x', 'y', d = 100)
```

→ a = 1, b = 2, args = ('x', ‘y'), d = 100

```
func(1, 2, d = 100)
```

→ a = 1, b = 2, args = (), d = 100

```
func(1, 2)
```

✖ d was not a keyword argument

We can even omit any mandatory positional arguments:

```
def func(*args, d):
    #code
```

```
func(1, 2, 3, d=100)   → args = (1, 2, 3), d = 100

func(d=100)            → args = (), d = 100
```

In fact we can force no positional arguments at all:

```
def func(*, d):
    #code
```

* indicates the "end" of positional arguments

```
func(1, 2, 3, d=100)   → Exception

func(d=100)            → d = 100
```

Putting it together

```
def func(a, b=1, *args, d, e=True):
    # code
```

```
def func(a, b=1, *, d, e=True):
    # code
```

`a`: mandatory positional argument (may be specified using a named argument)

`b`: optional positional argument (may be specified positionally, as a named argument, or not at all), defaults to `1`

`args`: catch-all for any (optional) additional positional arguments

`*`: no additional positional arguments allowed

`d`: mandatory keyword argument

`e`: optional keyword argument, defaults to `True`

Code