Putting it all Together

Positionals Only: no extra positionals, no defaults (all positionals required)

```
def func(a, b):
    print(a, b)
func('hello', 'world')
hello world

func(b='world', a='hello')
hello world
```

Positionals Only: no extra positionals, defaults (some positionals optional)

```
def func(a, b='world', c=10):
    print(a, b, c)
func('hello')
hello world 10

func('hello', c='!')
hello world !
```

Positionals Only: extra positionals, no defaults (all positionals required)

```
def func(a, b, *args):
    print(a, b, args)
func(1, 2, 'x', 'y', 'z')
1 2 ('x', 'y', 'z')
```

Note that we cannot call the function this way:

```
func(b=2, a=1, 'x', 'y', 'z')
  File "<ipython-input-9-f1b0ffb3b67d>", line 1
    func(b=2, a=1, 'x', 'y', 'z')
                   ^
SyntaxError: positional argument follows keyword argument
```

Keywords Only: no positionals, no defaults (all keyword args required)

```
def func(*, a, b):
    print(a, b)
func(a=1, b=2)
1 2
```

Keywords Only: no positionals, some defaults (not all keyword args required)

```
def func(*, a=1, b):
    print(a, b)
func(a=10, b=20)
10 20

func(b=2)
1 2
```

Keywords and Positionals: some positionals (no defaults), keywords (no defaults)

```
def func(a, b, *, c, d):
    print(a, b, c, d)
func(1, 2, c=3, d=4)
1 2 3 4

func(1, 2, d=4, c=3)
1 2 3 4

func(1, c=3, d=4, b=2)
1 2 3 4
```

Keywords and Positionals: some positional defaults

```
def func(a, b=2, *, c, d=4):
    print(a, b, c, d)
func(1, c=3)
1 2 3 4

func(c=3, a=1)
1 2 3 4

func(1, 2, c=3, d=4)
1 2 3 4

func(c=3, a=1, b=2, d=4)
1 2 3 4
```

Keywords and Positionals: extra positionals

```
def func(a, b=2, *args, c=3, d):
    print(a, b, args, c, d)
func(1, 2, 'x', 'y', 'z', c=3, d=4)
1 2 ('x', 'y', 'z') 3 4
```

Note that if we are going to use the extra arguments, then we cannot actually use a default value for b:

```
func(1, 'x', 'y', 'z', c=3, d=4)
1 x ('y', 'z') 3 4
```

as you can see, **b** was assigned the value **x**

Keywords and Positionals: no extra positionals, extra keywords

```
def func(a, b, *, c, d=4, **kwargs):
    print(a, b, c, d, kwargs)
func(1, 2, c=3, x=100, y=200, z=300)
1 2 3 4 {'x': 100, 'y': 200, 'z': 300}

func(x=100, y=200, z=300, c=3, b=2, a=1)
1 2 3 4 {'x': 100, 'y': 200, 'z': 300}
```

Keywords and Positionals: extra positionals, extra keywords

```
def func(a, b, *args, c, d=4, **kwargs):
    print(a, b, args, c, d, kwargs)
func(1, 2, 'x', 'y', 'z', c=3, d=5, x=100, y=200, z=300)
1 2 ('x', 'y', 'z') 3 5 {'x': 100, 'y': 200, 'z': 300}
```

Keywords and Positionals: only extra positionals and extra keywords

```
def func(*args, **kwargs):
    print(args, kwargs)
func(1, 2, 3, x=100, y=200, z=300)
(1, 2, 3) {'x': 100, 'y': 200, 'z': 300}
```

The Print Function

```
help(print)
Help on built-in function print in module builtins:

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

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.


print(1, 2, 3)
1 2 3

print(1, 2, 3, sep='--')
1--2--3

print(1, 2, 3, end='***\n')
1 2 3***

print(1, 2, 3, sep='\t', end='\t***\t')
print(4, 5, 6, sep='\t', end='\t***\n')
1       2       3       ***     4       5       6       ***
```

Another Use Case

```
def calc_hi_lo_avg(*args, log_to_console=False):
    hi = int(bool(args)) and max(args)
    lo = int(bool(args)) and min(args)
    avg = (hi + lo)/2
    if log_to_console:
        print("high={0}, low={1}, avg={2}".format(hi, lo, avg))
    return avg
avg = calc_hi_lo_avg(1, 2, 3, 4, 5)
print(avg)
3.0

avg = calc_hi_lo_avg(1, 2, 3, 4, 5, log_to_console=True)
print(avg)
high=5, low=1, avg=3.0
3.0
```