PUTTING IT ALL TOGETHER

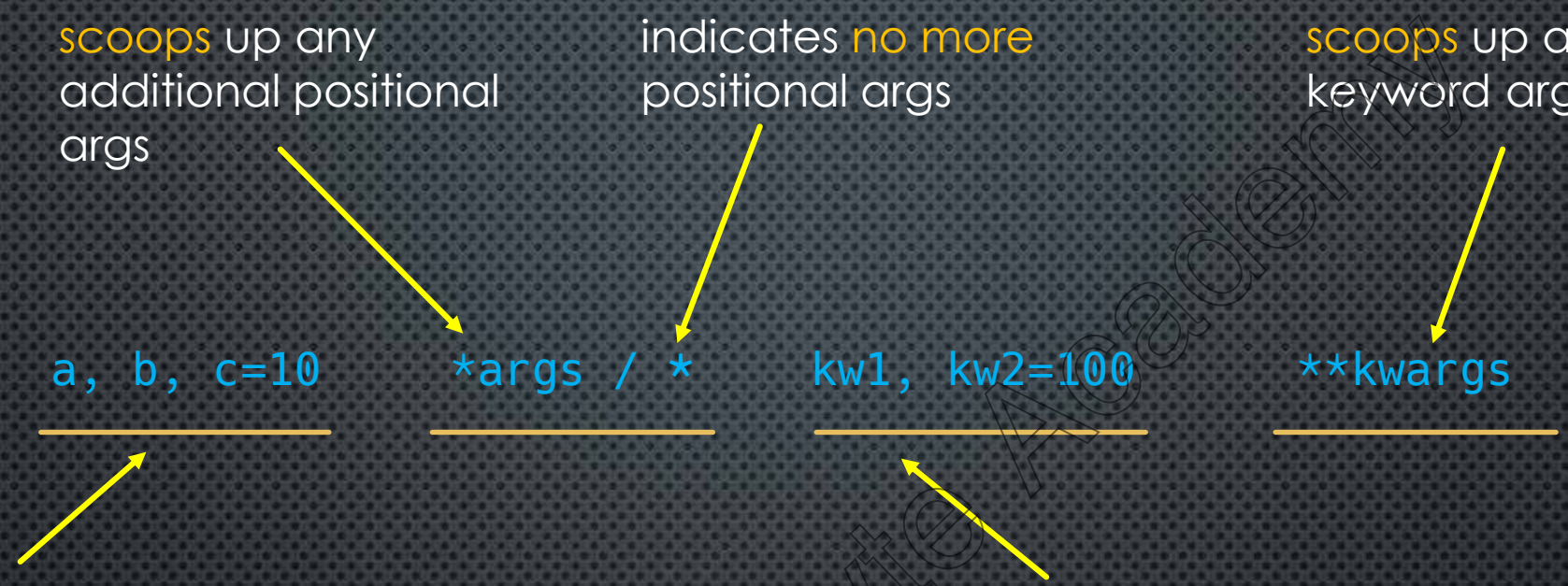
Recap

positional arguments

specific	may have default values
`*args`	collects, and exhausts remaining positional arguments
`*`	indicates the end of positional arguments (effectively exhausts)

keyword-only arguments

after positional arguments have ben exhausted	
specific	may have default values
`**kwargs`	collects any remaining keyword arguments



positional parameters
can have default values
non-defaulted params are mandatory args
user may specify them using keywords

specific keyword-only args
can have default values
non-defaulted params are mandatory args
user must specify them using keywords
if used, * or *args must also be used

Examples

```
def func(a, b=10)

def func(a, b, *args)

def func(a, b, *args, kw1, kw2=100)

def func(a, b=10, *, kw1, kw2=100)

def func(a, b, *args, kw1, kw2=100, **kwargs)

def func(a, b=10, *, kw1, kw2=100, **kwargs)

def func(*args)

def func(**kwargs)

def func(*args, **kwargs)
```

Typical Use Case: Python's `print()` function

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

Print *objects* to the text stream *file*, separated by *sep* and followed by *end*. *sep*, *end*, *file* and *flush*, if present, must be given as keyword arguments.

All non-keyword arguments are converted to strings like `str()` does and written to the stream, separated by *sep* and followed by *end*. Both *sep* and *end* must be strings; they can also be `None`, which means to use the default values. If no *objects* are given, `print()` will just write *end*.

The *file* argument must be an object with a `write(string)` method; if it is not present or `None`, `sys.stdout` will be used. Since printed arguments are converted to text strings, `print()` cannot be used with binary mode file objects. For these, use `file.write(...)` instead.

Whether output is buffered is usually determined by *file*, but if the *flush* keyword argument is true, the stream is forcibly flushed.

Changed in version 3.3: Added the *flush* keyword argument.

`*objects` arbitrary number of positional arguments

after that are keyword-only arguments

they all have default values, so they are all optional

Typical Use Cases

Often, keyword-only arguments are used to modify the default behavior of a function

such as the `print()` function we just saw

```
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
```

Other times, keyword-only arguments might be used to make things clearer.

Having many positional parameters can become confusing, and extra care has to be taken to ensure the correct parameters are passed in the correct sequence.

Code