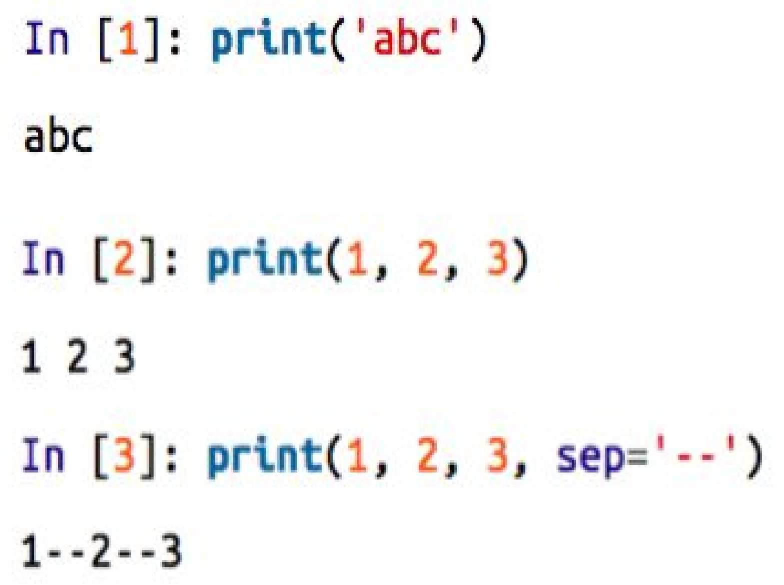
Python IV

Functions and Classes

Using functions

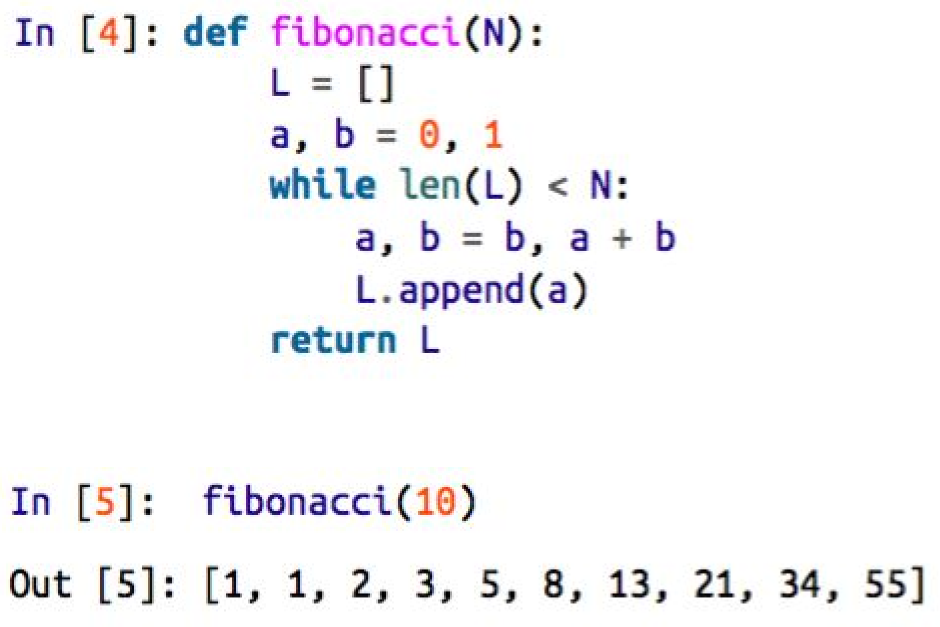
- Using functions: you can use optional keyword arguments (they ordinarily have a default)
- Keyword arguments must come after non-keyword arguments



```
In [1]: print('abc')

abc

In [2]: print(1, 2, 3)

1 2 3

In [3]: print(1, 2, 3, sep='--')

1--2--3
```

Defining functions

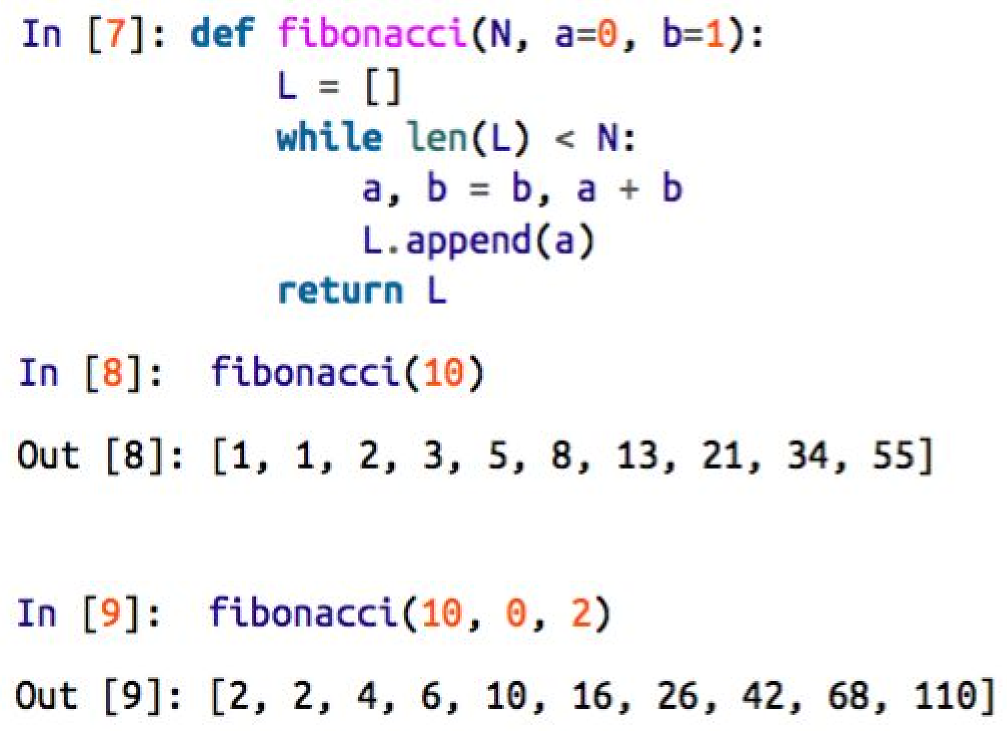
- No return type declarations



```
In [4]: def fibonacci(N):
            L = []
            a, b = 0, 1
            while len(L) < N:
                a, b = b, a + b
                L.append(a)
            return L


In [5]:  fibonacci(10)

Out [5]: [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

Default values

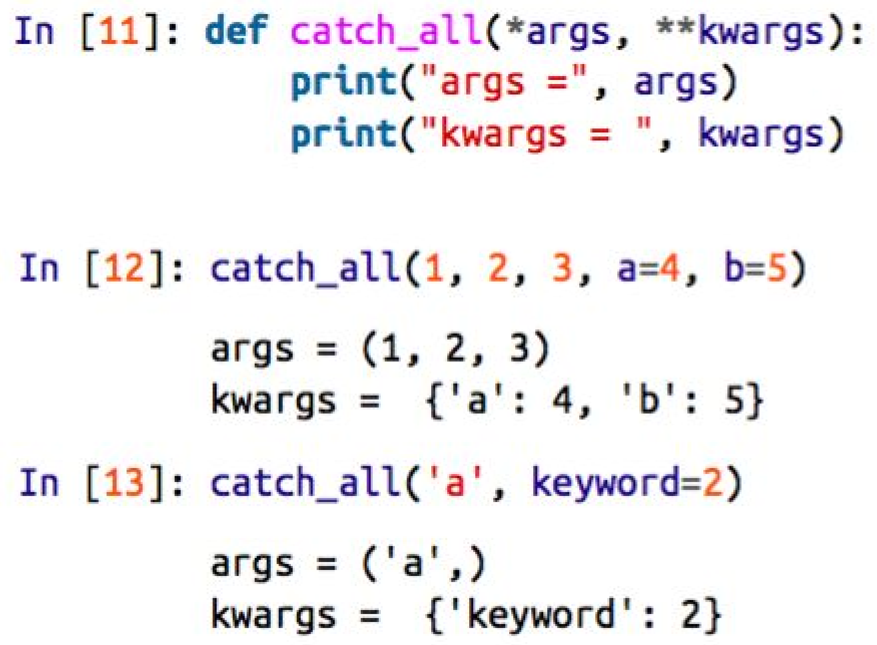
- You can add and use them with the syntax on the right



```
In [7]: def fibonacci(N, a=0, b=1):
            L = []
            while len(L) < N:
                a, b = b, a + b
                L.append(a)
            return L

In [8]:  fibonacci(10)

Out [8]: [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]


In [9]:  fibonacci(10, 0, 2)

Out [9]: [2, 2, 4, 6, 10, 16, 26, 42, 68, 110]
```

*ARGS AND **KWARGS

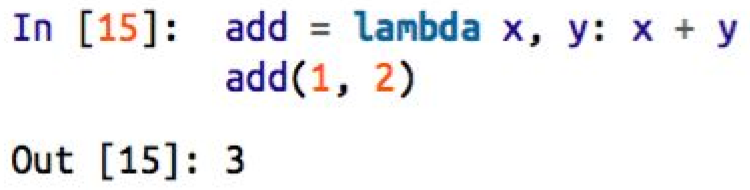
- *args is for a variable number of non-keyword arguments, and **kwargs is for a variable number of keyword arguments
- *args come stored in a tuple, **kwargs in a dictionary
- The asterisks are the operative symbols



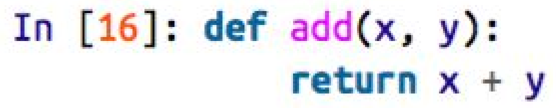
```
In [11]: def catch_all(*args, **kwargs):
             print("args =", args)
             print("kwargs = ", kwargs)


In [12]: catch_all(1, 2, 3, a=4, b=5)

         args = (1, 2, 3)
         kwargs =  {'a': 4, 'b': 5}

In [13]: catch_all('a', keyword=2)

         args = ('a',)
         kwargs =  {'keyword': 2}
```

Anonymous lambda functions

- Everything, including functions, are objects in Python
- add can be passed like other variables into other functions



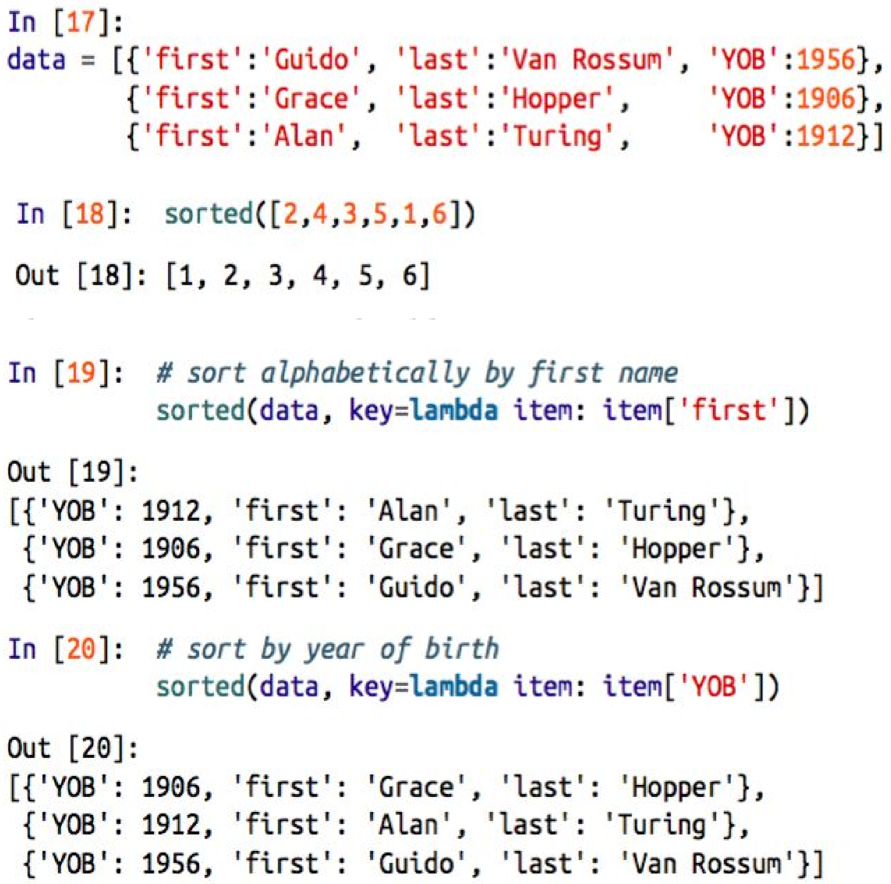
```
In [15]:  add = lambda x, y: x + y
          add(1, 2)

Out [15]: 3
```



```
In [16]: def add(x, y):
             return x + y
```

Using lambda functions

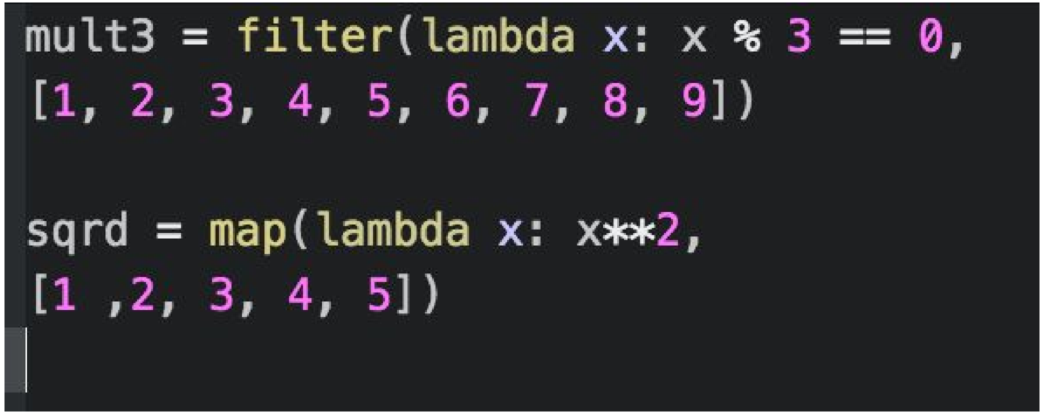
- For ordered collections, sorted is straightforward
- For unordered collections, a lambda function is required to tell sorted how to order the items



```
In [17]:
data = [{'first':'Guido', 'last':'Van Rossum', 'YOB':1956},
        {'first':'Grace', 'last':'Hopper',     'YOB':1906},
        {'first':'Alan',  'last':'Turing',     'YOB':1912}]

In [18]:  sorted([2,4,3,5,1,6])

Out [18]: [1, 2, 3, 4, 5, 6]

In [19]:  # sort alphabetically by first name
          sorted(data, key=lambda item: item['first'])

Out [19]:
[{'YOB': 1912, 'first': 'Alan', 'last': 'Turing'},
 {'YOB': 1906, 'first': 'Grace', 'last': 'Hopper'},
 {'YOB': 1956, 'first': 'Guido', 'last': 'Van Rossum'}]

In [20]:  # sort by year of birth
          sorted(data, key=lambda item: item['YOB'])

Out [20]:
[{'YOB': 1906, 'first': 'Grace', 'last': 'Hopper'},
 {'YOB': 1912, 'first': 'Alan', 'last': 'Turing'},
 {'YOB': 1956, 'first': 'Guido', 'last': 'Van Rossum'}]
```

More lambda function examples

Lambda functions are used in functional programming



```
mult3 = filter(lambda x: x % 3 == 0,
[1, 2, 3, 4, 5, 6, 7, 8, 9])

sqrd = map(lambda x: x**2,
[1 ,2, 3, 4, 5])
```