FUNCTION INTROSPECTION

Functions are first-class objects

They have attributes `__doc__` `__annotations__`

We can attach our own attributes

```
def my_func(a, b):
    return a + b

my_func.category = 'math'
my_func.sub_category = 'arithmetic'

print(my_func.category)          → math

print(my_func.sub_category)      → arithmetic
```

The dir( ) function

dir( ) is a built-in function that, given an object as an argument, will return a list of valid attributes for that object

```
dir(my_func)
```

```
['__annotations__', '__call__', '__class__', '__closure__',
'__code__', '__defaults__', '__delattr__', '__dict__',
 '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__get__', '__getattribute__', '__globals__', '__gt__',
 '__hash__', '__init__', '__init_subclass__', '__kwdefaults__',
'__le__', '__lt__', '__module__', '__name__',
 '__ne__', '__new__', '__qualname__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__sizeof__',
 '__str__', '__subclasshook__', 'category', 'sub_category']
```

Function Attributes: `__name__`, `__defaults__`, `__kwdefaults__`

`__name__`	→ name of function
`__defaults__`	→ tuple containing positional parameter defaults
`__kwdefaults__`	→ dictionary containing keyword-only parameter defaults

```
def my_func(a, b=2, c=3, *, kw1, kw2=2):
    pass
```

`my_func.__name__`	→ `my_func`
`my_func.__defaults__`	→ `(2, 3)`
`my_func.__kwdefaults__`	→ `{'kw2': 2}`

Function Attribute: `__code__`

```
def my_func(a, b=1, *args, **kwargs):
    i = 10
    b = min(i, b)
    return a * b
```

```
my_func.__code__
    → <code object my_func at 0x00020EEF … >
```

This `__code__` object itself has various properties, which include:

`co_varnames` parameter and local variables

```
my_func.__code__.co_varnames    → ('a', 'b', 'args', 'kwargs', 'i')
```

parameter names first, followed by local variable names

`co_argcount` number of parameters

```
my_func.__code__.co_argcount    → 2
```

does not count `*args` and `**kwargs`!

The inspect Module

```
import inspect
```

`ismethod(obj)`   `isfunction(obj)`   `isroutine(obj)`   and many others…

What's the difference between a function and a method?

Classes and objects have attributes – an object that is bound (to the class or the object)

An attribute that is callable, is called a method

```
def my_func():
    pass

def MyClass:
    def func(self):
        pass

my_obj = MyClass()
```

`func` is bound to `my_obj`, an instance of `MyClass`

```
isfunction(my_func) → True
ismethod(my_func) → False
isfunction(my_obj.func) → False
ismethod(my_obj.func) → True

isroutine(my_func) → True
isroutine(my_obj.func) → True
```

Code Introspection

We can recover the source code of our functions/methods

`inspect.getsource(my_func)` → a string containing our entire def statement, including annotations, docstrings, etc

We can find out in which module our function was created

```
inspect.getmodule(my_func)      → <module '__main__'>

inspect.getmodule(print)        → <module 'builtins' (built-in)>

inspect.getmodule(math.sin)     → <module 'math' (built-in)>
```

Function Comments

```
# setting up variable
i = 10

# TODO: Implement function
# some additional notes
def my_func(a, b=1):
    # comment inside my_func
    pass
```

```
inspect.getcomments(my_func)

  → '# TODO: Implement function\n# some additional notes'
```

Many IDE's support the TODO comment to flag functions and other callables

Note that this is not the same as docstrings

Callable Signatures

`inspect.signature(my_func)` → `Signature` instance

Contains an attribute called `parameters`

Essentially a dictionary of parameter names (keys), and metadata about the parameters (values)

- keys → parameter name
- values → object with attributes such as `name`, `default`, `annotation`, `kind`

`kind`	`POSITIONAL_OR_KEYWORD`
	`VAR_POSITIONAL`
	`KEYWORD_ONLY`
	`VAR_KEYWORD`
	`POSITIONAL_ONLY`

Callable Signatures

```
def my_func(a: 'a string',
            b: int = 1,
            *args: 'additional positional args',
            kw1: 'first keyword-only arg',
            kw2: 'second keyword-only arg' = 10,
            **kwargs: 'additional keyword-only args') -> str:
    """does something
       or other"""
    pass



for param in inspect.signature(my_func).parameters.values():
        print('Name:', param.name)
        print('Default:', param.default)
        print('Annotation:', param.annotation)
        print('Kind:', param.kind)
```

Code