Metaprogramming in Python

By Rajat Goyal

@rajat404

What is metaprogramming?

Code manipulating Code

Why use metaprogramming?

- Better understanding of Python libraries/frameworks
- DRY
- Simplify your workflow

Closures

Closures are functions that return functions

In [5]:
```
def create_func(a, b):
    def multiply():
        return a*b
    return multiply
```

In [6]:
```
>>> temp = create_func(5, 10)
>>> temp()
```

Out[6]:
```
50
```

Decorators

A decorator is a function that wraps around another function, therby modifiying it's functionality

In [22]:
```
from decorator import decorator

@decorator
def print_something(fn, *args, **kwargs):
    print('Something is printed before the function\'s execution!')
    return fn(*args, **kwargs)

@print_something
def adder(x, y):
    return x + y
```

In [23]:
```
adder(1,2)
```

```
Something is printed before the function's execution!
```

Out[23]:
```
3
```

Using Decorators in everyday programming

- logging
- debugging
- validation

In [1]:

```python
# Decorator for Logging
from functools import wraps
import logging

def logger_decorator(f):
    log = logging.getLogger(f.__module__)
    text = f.__qualname__
    @wraps(f)
    def wrapper(*args, **kwargs):
        log.debug(text)
        return f(*args, **kwargs)
    return wrapper
```

In [9]:

```
# Decorator for Debugging
from decorator import decorator

@decorator
def debug_something(fn, *args, **kwargs):
    print('args: {} \nkwargs: {}'.format(args, kwargs))
    return fn(*args, **kwargs)
```

Classes & Types

- Brief intro to Types in Python
- How to create custom Types

Class Decorators

- Decorators for Classes
- Can wrap only instance methods
- Better Solutions?

In [10]:
```
def class_deco(cls):
    print('Inside class: {}'.format(cls.__qualname__))
    return cls
```

In [12]:
```
@class_deco
class A:
    pass
```

```
Inside class: A
```

Metaclasses

- Class of a Class
- class whose instances are themselves classes

In [16]:

```
# Basic Metaclass
class SampleMeta(type):
    def __new__(cls, classname, bases, clsdict):
        print('clsdict:', clsdict)
        return super().__new__(cls, classname, bases, clsdict)

# Define a new class using the SampleMeta metaclass
class A(metaclass=SampleMeta):
    pass
```

```
clsdict: {'__module__': '__main__', '__qualname__': 'A'}
```

Metaprogramming Best Practices

General principles & syntax that should be followed while writing decorators & metaclasses