Python Function Decorators Cheat Sheet
https://github.com/danielpacker/Public-Code/tree/master/misc/python/python-decorators

```python
from functools import wraps, update_wrapper


def decorator_name(func):
    """Boilerplate for standard function-type decorator
with no params"""
    @wraps(func)
    def wrapper(*args, **kwargs):
        # do something with func()
        return func(*args, **kwargs)
    return wrapper


@decorator_name
def hello(name):
    print('hello {}'.format(name))


def decorator_factory_name(*args, **kwargs):
    """Boilerplate for function-type decorator with
params"""
    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            # do something with func()
            return func(*args, **kwargs)
        return wrapper
    return decorator


@decorator_factory_name('arg1', arg2=True)
def goodbye(name):
    print('goodbye {}'.format(name))


class DecoratorName:
    """Simple class-type decorator with no params"""
    def __init__(self, func):
        self.func = func
        # this is similar to functools.wraps():
        update_wrapper(self, func)

    def __call__(self, *args, **kwargs):
        # do something with self.func
        return self.func(*args, **kwargs)


@DecoratorName
def apologize(name):
    print('Sorry, {}!'.format(name))


class DecoratorFactoryName:
    """Class-type decorator with params"""
    def __init__(self, positional, **named):
        # do something with positional or named params
        pass

    def __call__(self, func):
        # this is similar to functools.wraps():
        update_wrapper(self, func)

        def wrapper(*args, **kwargs):
            # do something with func()
            return func(*args, **kwargs)
        return wrapper


@DecoratorFactoryName('arg1', arg2=True)
def adore(name):
    print('{}, you\'re amazing!'.format(name))


if __name__ == "__main__":

    hello('Daniel')
    goodbye('Britney')
    apologize('Harry')
    adore('Melinda')
```