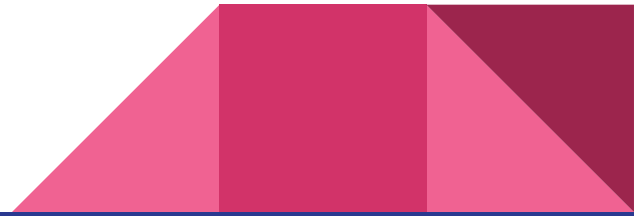
Decorator Examples

Decorator in Python

Decorator in Python is a function that receives another function as an argument. The behavior of the argument function is extended by the decorator without actually modifying it. The decorator function can be applied over a function using the @decorator syntax

Basically, a decorator takes in a function, adds some functionality and returns it.



Examples

```
1   def decorator2(func):
2       """ decorator function"""
3
4       def wrapper(*args,**kwargs):  # define wrapper with *args and **kwargs
5         """ wrapper function to do the sum of squares of the numbers"""
6         s=0
7         for i in args:              # sum the squares of the numbers in the argument
8               s=s+i*i
9         return s                    # returns the sum of squares
10
11      return wrapper                # returns the wrapper
12
13
14  @decorator2
15  def add_num(x,y):
16      return x+y
```

```python
def decorator_list(fnc):
    def inner(list_of_tuples):
        return [fnc(val[0], val[1]) for val in list_of_tuples]
    return inner


@decorator_list
def add_together(a, b):
    return a + b


print(add_together([(1, 3), (3, 17), (5, 5), (6, 7)]))

# add_together = decorator_list(add_together)
```

```
def meta_decorator(arg):
    def decorator_list(fnc):
        def inner(list_of_tuples):
            return [(fnc(val[0], val[1])) ** power for val in list_of_tuples]
        return inner
    if callable(arg):
        power = 2
        return decorator_list(arg)
    else:
        power = arg
        return decorator_list


@meta_decorator
def add_together(a, b):
    return a + b
```

```
print(add_together([(1, 3), (3, 17), (5, 5), (6, 7)]))
```

```
# example of decorator
def sampleDecorator(func):
    def addingFunction():
        # some new statments or flow control
        print("This is the added text to the actual function.")
        # calling the function
        func()

    return addingFunction


@sampleDecorator
def actualFunction():
    print("This is the actual function.")


actualFunction()
```

```
def flowerDecorator(vasel):
    def newFlowerVase(n):
        print("We are decorating the flower vase.")
        print("You wanted to keep %d flowers in the vase." % n)

        vasel(n)

        print("Our decoration is done")

    return newFlowerVase


@flowerDecorator
def flowerVase(n):
    print("We have a flower vase.")


flowerVase(6)
```

Can you assume the output?? The output of the above code will be:

```
We are decorating the flower vase.
You wanted to keep 6 flowers in the vase.
We have a flower vase.
Our decoration is done
```