In [9]:
```python
#Decorator to Ensure All Arguments Are Non-Negative Integers
def non_negative(func):
    def wrapper(*args,**kwargs):
        for arg in args:
            if not isinstance(arg,int) or arg<0:
                raise ValueError("the number should be positive")
        return func(*args,**kwargs)
    return wrapper

@non_negative
def calculate_square_root(x):
    return x**0.5

print(calculate_square_root(9))
```

```
3.0
```

In [14]:
```python
#Decorator to Print Function Name, Arguments, and Return Value
def trace_call(func):
    def wrapper (*args,**kwargs):
        print(f"function name:{func.__name__}")
        print(f"arguments: args:{args} ,kwargs:{kwargs}")
        result=func(*args,**kwargs)
        print(f"returned :{result}")
        return result
    return wrapper

@trace_call
def add(a,b):
    return a+b
add(2,3)
```

```
function name:add
arguments: args:(2, 3) ,kwargs:{}
returned :5
```

Out[14]: 5

In [22]:
```python
# Decorator to Repeat Function Execution
def repeat(times):
    def decorator(func):
        def wrapper(*args,**kwargs):
            result=None
            for i in range(times):
                print(f"execution:{i+1}")
                result=func(*args,**kwargs)
            return result
        return wrapper
    return decorator
@repeat(3)
def greet(name):
    return print(f"hello {name}")
greet("uzma")
```

```
execution:1
hello uzma
execution:2
hello uzma
execution:3
hello uzma
```

In [25]:
```python
#Decorator to Count How Many Times a Function Has Been Called
def count(func):
    def wrapper(*args,**kwargs):
        wrapper.call_count+=1
        print(f"function {func.__name__} called {wrapper.call_count} times")
        return func(*args,**kwargs)
    wrapper.call_count=0
    return wrapper
@count
def greet():
    return "hello"

greet()
greet()
```

```
function greet called 1 times
function greet called 2 times
```

Out[25]:
```
'hello'
```

In [ ]:

In [ ]:

In [ ]: