Decorators

Decorators

```
decorators                    ::=  decorator+
decorator                     ::=
"@" dotted_name ["(" [argument_list [","]] ")"]
NEWLINE
```

Decorators

```
decorators                ::=  decorator+
decorator                 ::=
"@" dotted_name ["(" [argument_list [","]] ")"] NEWLINE
```

```python
@f1(arg)
@f2
def func(): pass
```

~

```python
def func(): pass
func = f1(arg)(f2(func))
```

Dummy decorator

```
>>> def dummy_decorator(func):
...     print('Hello')
...
...
... @dummy_decorator
... def func():
...     print('Inner execution of f')

Hello
```

~

```
>>> func = dummy_decorator(func)
Hello
>>> type(func)
<class 'NoneType'>
```

Decorators

```
import time

def time_decorator(func):

    def wrapper(*args, **kwargs):
        start = time.time()
        result = func(*args, **kwargs)
        end = time.time()
        print(f'Execution time {end - start}')

        return result

    return wrapper

@time_decorator
def join(seq, delimiter):
    return delimiter.join(seq)

print(join(['h', 'e', 'l', 'l', 'o'], delimiter=''))
Execution time 1.9073486328125e-06
hello
```

Decorators

```
def upper(func):

    def wrapper(*args, **kwargs):
        res = func(*args, **kwargs).upper()
        return res

    return wrapper

@upper
def hello(string: str) -> str:
    return string


>>> hello("hello")
'HELLO'
```

Decorators

```python
def upper(func):

    def wrapper(*args, **kwargs):
        res = func(*args, **kwargs).upper()
        return res

    return wrapper

@upper
def hello(string: str) -> str:
    return string


>>> hello.__name__
'wrapper'
```

Decorators

```python
import functools

def upper(func):

    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        res = func(*args, **kwargs).upper()
        return res

    return wrapper

@upper
def hello(string: str) -> str:
    return string

>>> hello("hello")
'HELLO'

>>> hello.__name__
'hello'
```

Parameterized decorators

```python
def decorator_factory(log=True):

    def decorator(func):

        @functools.wraps(func)
        def wrapper(*args, **kwargs):
            if log:
                print(f"Called with {args!r} {kwargs!r}")
            return func(*args, **kwargs)

        return wrapper

    return decorator

@decorator_factory(log=True)
def hello(string: str) -> str:
    return string

>>> hello("hello")
Called with ('hello',) {}
'hello'
```

Parameterized decorators

```python
def decorator_factory(log=True):

    def decorator(func):

        @functools.wraps(func)
        def wrapper(*args, **kwargs):
            if log:
                print(f"Called with {args!r} {kwargs!r}")
            return func(*args, **kwargs)

        return wrapper

    return decorator

@decorator_factory(log=False)
def hello(string: str) -> str:
    return string

>>> hello("hello")
'hello'
```