Decorator Design Techniques

Oleh : Agus Priyanto, M.Kom

INSTITUT TEKNOLOGI TELKOM PURWOKERTO
Smart, Trustworthy, And Teamwork

Outline Materi

- Introduction Decorator
- Create Decorator
- Multiple Decorators
- Memoization using decorators

Decorator

- Dalam Python, ***Decorator*** merupakan salah satu **design pattern** berfungsi menambah fungsionalitas pada kode program.

- **Decorator** disebut juga dengan istilah *metaprogramming* karena ada bagian dari program yang mencoba untuk memodifikasi bagian lainnya pada saat eksekusi.

Composite Design

$$f \circ g(x) = f\left(g(x)\right)$$

Jika kita perhatikan fungsi diatas, kita dapat mendefiniskan fungsi tersebut dengan menggunakan dua atau lebih fungsi yaitu **f (y)** dan **g(x)**

Create Decorator

```
def lowercase(func):
   def wrapper():
       func_ret = func()
       change_to_lowercase = func_ret.lower()
   return change_to_lowercase
 return wrapper
```

Testing 1

```
def hello_function():
   return 'HELLO WORLD'

decorate = lowercase(hello_function)
print(decorate())
```

Testing 2

```
@lowercase
def hello_function():
   return 'HELLO WORLD'

print(hello_function())
```

Whenever youare writing a decorator, always be sure to add **functools.wraps** to wrap the inner function.
Without wrapping it, you will lose all properties from the original function, which can lead to confusion.

Multiple Decorators

- Dalam Python, kita dapat menggunakan lebih dari satu decorator dalam single function.

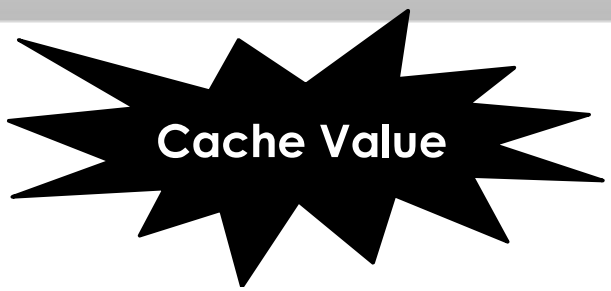
```
def split_sentence(func):
  def wrapper():
      func_ret = func()
      output = func_ret.split()
      return output
  return wrapper
```

Testing 3

```
@split_sentence
@lowercase
def hello_function():
  return 'HELLO WORLD'
  print(hello_function())
```

Memoization using decorators

```
import functools
   def memoize(function):
        function.cache = dict()
        @functools.wraps(function)
        def _memoize(*args):
               if args not in function.cache:
                      function.cache[args] = function(*args)
               return function.cache[args]
        return _memoize
```

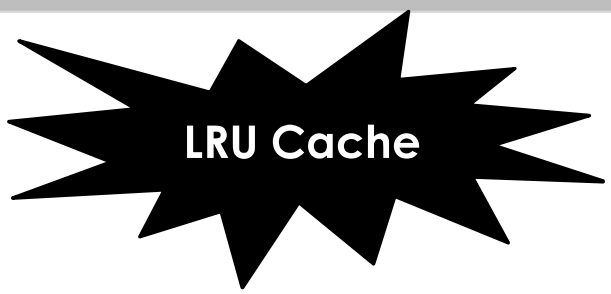


Testing 4

```
@memoize
def fibonacci(n):
  if n < 2:
      return n
  else:
      return fibonacci(n - 1) + fibonacci(n - 2)
```

```
for i in range(1, 7):
  print('fibonacci %d: %d' % (i, fibonacci(i)))
```

```
import functools
   def counter(function):
        function.calls = 0
        @functools.wraps(function)
        def _counter(*args, **kwargs):
               function.calls += 1
        return function(*args, **kwargs)
   return _counter
```



Testing 5

```
@functools.lru_cache(maxsize=3)
@counter
def fibonacci(n):
  if n < 2:
      return n
  else:
      return fibonacci(n - 1) + fibonacci(n - 2)
```

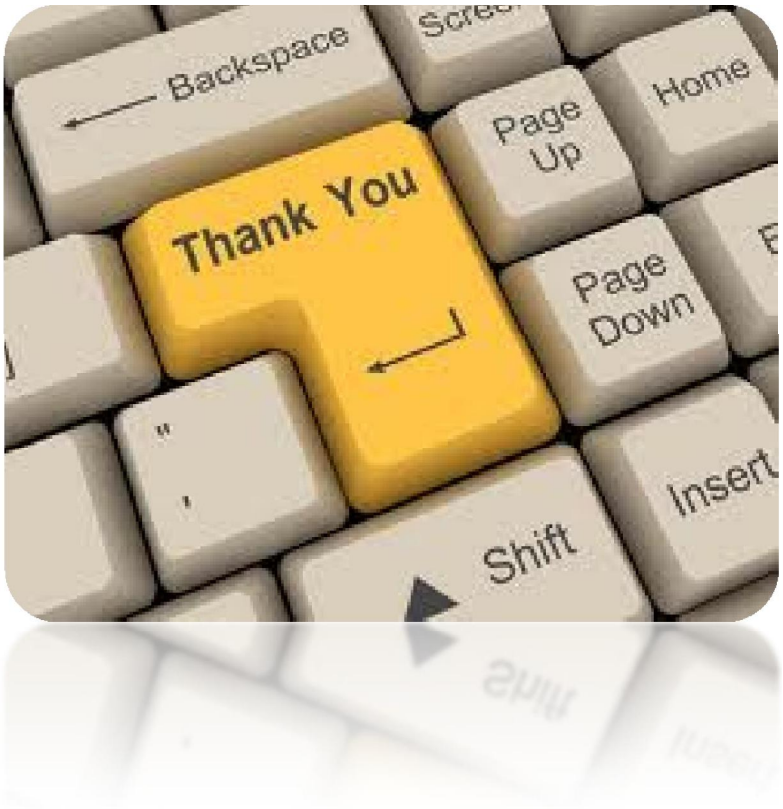
```
fibonacci(100)
354224848179261915075
```

Final Project (Take Home)

- Buatlah kelompok kerja dengan anggota maksimal 3 orang anggota

- **Studi Kasus :** The Functional Approach to Web Services.
  - WSGI application
  - Serializing data
    - JSON or CSV format
    - XML
    - HTML

- Buatlah karya tulis kelompok anda dalam 2 format, yaitu :
  - Hardcopy (PDF)
  - Softcopy dalam blog http://student.ittelkom-pwt.ac.id/
- Hasil Karya Tulis diemail ke : agus_priyanto@ittelkom-pwt.ac.id
- Paling lambat 13 Januari 2020


Backspace
Thank You
Page Up
Home
Page Down
Insert
Shift