Decorators as Composable Abstractions

Functional Programming

λ

Higher Order Functions

λ :: λ → λ

Functions as Arguments

$$\lambda :: \lambda \rightarrow x$$

Functions as Results

$$\lambda :: x \rightarrow \lambda$$

Both

$$\lambda :: \lambda \rightarrow \lambda$$

Functions as Arguments

```
λ :: λ → x
```

Functions as Arguments

λ :: λ → x

```
map(double,[1,2,3]) → [2,4,6]

filter(is_even,[1,2,3]) → [2]

reduce(add,[1,2,3]) → 6
```

Functions as Results

```
λ :: x → λ
```

Functions as Results

λ :: x → λ

```
make_adder(6) → add6

add6(5) → 11
```

Both

$\lambda :: \lambda \rightarrow \lambda$

Both

$\lambda :: \lambda \rightarrow \lambda$

```
partial(add,6) → add6

add6(5) → 11
```

@Decorators are syntactic sugar for functions that do both

$$\lambda :: \lambda \rightarrow \lambda$$

doubler

```
def doubler(f):
    def decorate(*args,**kwargs):
        return 2*f(*args,**kwargs)
    return decorate
```

doubler

```
@doubler
def add(x,y):
   return x + y

add(2, 3) → 10
```

Parametrized decorators are functions that return decorators

```
λ :: x → λ → λ
```

nler

```
def nler(n):
  def decorator(f):
    def decorate(*args,**kwargs):
      return n * f(*args,**kwargs)
    return decorate
  return decorator
```

nler

```
@nler(3)
def add(x,y):
   return x + y

add(2, 3) → 15
```

Decorators as an Abstraction Mechanism

Decorators as a Composable Abstraction Mechanism

```
@doubler
@nler(2)
def add(x,y):
   return x + y

add(2, 3) → 20
```

Real World Problems

Real World Problems

A REST api framework

Library API

GET /books/ - A list of book ids
POST /books/ - Adds a new book
GET /books/<id>/ - Gets a book
PUT /books/<id>/ - Edits a book
DELETE /books/<id>/ - Deletes a book

Add a Book

```
@dejsonify_response
@jsonify_request
@request
def add_book(title, author):
  return {"method": "POST",
          "url": gen_url("books")
          "body": {"title": title,
                   "author": author}}
```

Add a Book

```
@dejsonify_response
@jsonify_request
@request
def add_book(title, author):
  return {"method": "POST",
          "url": gen_url("books")
          "body": {"title": title,
                   "author": author}}
```

add_book => request object

Add a Book

```
@dejsonify_response
@jsonify_request
@request
def add_book(title, author):
  return {"method": "POST",
          "url": gen_url("books")
          "body": {"title": title,
                   "author": author}}
```

add_book => JSON encodes body and adds "Content-Type" header

Add a Book

```
@dejsonify_response
@jsonify_request
@request
def add_book(title, author):
  return {"method": "POST",
          "url": gen_url("books")
          "body": {"title": title,
                   "author": author}}
```

add_book => Decodes the JSON response body

Request Class

```
class Request:
  def __init__(function=lambda(x): x,
               payload={}):
    self.function = function
    self.payload = payload

  def __call__():
    return self.function(self.payload)
```

Request Decorator

```
def request(f):
  def decorate(*args, **kwargs):
    req = f(*args, **kwargs)
    res = Request(http_request, req)
    return res
  return decorate
```

Request Decorator

```
def request(f):
  def decorate(*args, **kwargs):
    req = f(*args, **kwargs)
    res = Request(http_request, req)
    return res
  return decorate
```

http_request(req) => res

Request Decorator

```
def request(f):
  def decorate(*args, **kwargs):
    req = f(*args, **kwargs)
    res = Request(http_request, req)
    return res
  return decorate
```

```
req.keys()
=> ["method", "headers", "url", "body"]
```

Request Decorator

```
def request(f):
  def decorate(*args, **kwargs):
    req = f(*args, **kwargs)
    res = Request(http_request, req)
    return res
  return decorate
```

```
res.keys()
=> ["status", "headers", "body"]
```

"Prequest" Decorator

```
def prequest(f):
  def decorate(*args, **kwargs):
    req = f(*args, **kwargs)
    payload = req.payload
    # Modify Payload Here
    req.payload = payload
    return req
  return decorate
```

"Prequest" Decorator

```
def prequest(f):
  def decorate(*args, **kwargs):
    req = f(*args, **kwargs)
    payload = req.payload
    # Modify Payload Here
    req.payload = payload
    return req
  return decorate
```

"Prequest" Decorator

```
def prequest(f):
  def decorate(*args, **kwargs):
    req = f(*args, **kwargs)
    payload = req.payload
    # Modify Payload Here
    req.payload = payload
    return req
  return decorate
```

"Prequest" Decorator

```
def prequest(f):
  def decorate(*args, **kwargs):
    req = f(*args, **kwargs)
    payload = req.payload
    # Modify Payload Here
    req.payload = payload
    return req
  return decorate
```

Jsonify Decorator

```
def jsonify(f):
  def decorate(*args, **kwargs):
    req = f(*args, **kwargs)
    payload = req.payload
    body = payload["body"]
    payload["body"] = json.dumps(body)
    req.payload = payload
    return req
  return decorate
```

"Postquest" Decorator

```
def postquest(f):
  def decorate(*args, **kwargs):
    req = f(*args, **kwargs)
    function = req.function
    # wrap function here
    req.function = function
    return req
  return decorate
```

"Postquest" Decorator

```
def postquest(f):
  def decorate(*args, **kwargs):
    req = f(*args, **kwargs)
    function = req.function
    # wrap function here
    req.function = function
    return req
  return decorate
```

"Postquest" Decorator

```
def postquest(f):
  def decorate(*args, **kwargs):
    req = f(*args, **kwargs)
    method = req.method
    # wrap function here
    req.function = function
    return req
  return decorate
```

"Postquest" Decorator

```
def postquest(f):
  def decorate(*args, **kwargs):
    req = f(*args, **kwargs)
    method = req.method
    # wrap function here
    req.function = function
    return req
  return decorate
```

Dejsonify Decorator

```
def dejsonify(f):
  def decorate(*args, **kwargs):
    req = f(*args, **kwargs)
    function = req.function
    function = compose(jloads,function)
    req.method = new_method
    return req
  return decorate
```

Add a Book

```
@dejsonify_response
@jsonify_request
@request
def add_book(title, author):
  return {"method": "POST",
          "url": gen_url("books")
          "body": {"title": title,
                   "author": author}}
```

Add a Book

```
@dexmlify_response
@xmlify_request
@request
def add_book(title, author):
  return {"method": "POST",
          "url": gen_url("books")
          "body": {"title": title,
                   "author": author}}
```

Add a Book

```
@dejsonify_response
@xmlify_request
@request
def add_book(title, author):
  return {"method": "POST",
          "url": gen_url("books")
          "body": {"title": title,
                   "author": author}}
```

Composable Abstractions

Thank You

Questions?

inspired by
http://blip.tv/clojure/mark-mcgranaghan-one-ring-to-bind-them-4724955