Eliminating Side Effects with Pure Functions



Gerald Britton
IT Solutions Designer

@GeraldBritton www.linkedin.com/in/geraldbritton

Overview



- Simple – do one thing and do it well
- Limited number of arguments
- Output depends on input
- Same input ⇒ same outputs
- State not used or modified
- No side effects
- Complexity through combination

One Function or Three?

List of integers

Sometimes you want the odd ones

Sometimes you want the evens

Sometimes all of them

One Function or Three?

```python
def get_ints(ints, odd=True, even=True):
    if odd and even:
        return [i for i in ints]
    elif odd:
        return [i for i in ints if i % 2]
    elif even:
        return [i for i in ints if not i % 2]
    else:
        return []
```

Do it all function

```python
def get_even_ints(ints):
    return [i for i in ints if not i % 2]

def get_odd_ints(ints):
    return [i for i in ints if i % 2]

def get_all_ints(ints):
    return list(ints)
```

Split them!

Demo



Add new functionality

- Expedite an order

Demo



Making functions pure

Recap

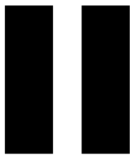


- Added new functionality
- Set the expedited flag
- Eliminated None handling
- Added two pure functions
- Refactored using the pure functions
- Cannot purify everything
- Input and output are impure
- Minimize and isolate impure operations

```
import dis
def f(x):
    return x.g(lambda x: x.good, lambda x: x.member)
dis.dis(f)
```

Lambdas in Python

- **Import the `dis` module**
- **Define a simple test function**
- **Disassemble it**



```
0 LOAD_FAST                0 (x)
2 LOAD_ATTR                0 (g)
4 LOAD_CONST               1
    (<code object <lambda> ...
6 LOAD_CONST               2
    ('f.<locals>.<lambda>')
8 MAKE_FUNCTION            0
0 LOAD_CONST               3
    (<code object <lambda> ...
2 LOAD_CONST               2
    ('f.<locals>.<lambda>')
4 MAKE_FUNCTION            0
6 CALL_FUNCTION            2
8 RETURN_VALUE
```

- Disassembled function
- Load lambda 1
- Make it a function!
- Load lambda 2
- Make it a function
- Call the original function
- Return the results

```
def l1(x): return x.good
def l2(x): return x.member
def f(x): return x.g(l1, l2)
dis.dis(f)
```

Helper Functions

- **Define functions to replace the lambdas**
- **Use the replacements in the main function**
- **Disassemble it**

```
 0 LOAD_FAST                0 (x)
 2 LOAD_ATTR                0 (g)
 4 LOAD_GLOBAL              1 (l1)
 6 LOAD_GLOBAL              2 (l2)
 8 CALL_FUNCTION            2
10 RETURN_VALUE
```

- Disassembled function

- Load function 1
- Load function 2
- Run the original function

Lambdas vs. Helper Functions

Use lambda functions or helpers?

It depends!

Lambdas are used where defined

~~Premature optimization~~

Summary



Introduced pure functions

- Output depends on input
- No side effects

Added ability to set expedited flag

Created pure functions map and filter

Made other new functions pure

Discussed pros and cons of lambdas