Side Effects

The Good

```
obj = Square(w=10, h=10)
obj.move_to(x=10, y=10)
obj.rotate(deg=45)
```

```
while true:
   noisy = read_input()
   clean = filter(noisy)
   output(clean)
```

Stateful Objects
Inputs / Output

The Bad

```
width  = 0
height = 0

def Square.move_to(x, y):
   this.pos_x = x
   this.pos_y = y
   width  = max(width,  x)
   height = max(height, y)
```

Unexpected results

The Ugly

```
def Square.rotate(deg):
   this.rotate += deg
   system("rm -rf /")
```

Malicious coders
(*should be dealt with accordingly*)

Side effects can benefit programmers by providing concise and simple APIs, they mimic the physical world and can improve encapsulation and modularity. However, they can also lead to poorly designed software by reaching beyond their intended scope.

Inside Effects

“The execution of one function may not effect the execution of another function, except though arguments and return values. However, internal state may be maintained and I/O may be performed as usual.”

Data structures

```
def HashMap():
  data = {}
  def self('set', key, v):
     data[key] = v
  def self('get', key):
     return data[key]
  return self

obj  = HashMap()
obj('set', 'x', 10)
x = obj('get', 'x') // 10
```

self is a single pattern matched function. It maintains it's own copy of *data*, separate from any other functions.

Objects can be created using message passing style for method calls.

Multithreading

Race Conditions only occur within reentrant calls to the same function

Dataflow analysis is simplified because functions cannot effect outside functions

Processes provide a hard boundary and can communicate though sources and sinks

Lexical Scoping

```
num = 0

def foo():
  num = 1
  return num

def bar():
  num = 2
  return num
```

```
x = foo() // 1
y = bar() // 2
z = num   // 0
```

State is maintained lexically within closures

Global data is copied for each closure

$$\begin{tikzcd} & \text{num=0} \arrow[dl, "\text{foo()}"'] \arrow[dr, "\text{bar()}"] & \\ \text{num=1} & & \text{num=2} \end{tikzcd}$$

Copy-on-write

References modifications to refs are only visible within the same function.

State is maintained between calls to the same function.

Input and ***Output*** functions have no effect within the current process.

Call by Value with deep copies prevents shared access to refs.