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 poor 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.”

```
def main():
    print("Starting")

    num  = random()

    snum = sin(num)
    cnum = cos(num)
```

Sink functions "consume" data

Source functions "produce" data

The call sin is guaranteed to have no effect on the call to cos because no data passed between them

Copy-on-write

References may be updated only within a function. When a reference is modified from another function a copy is made for the second function.

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 sink