

Global Variables are Evil!

Modular programming

Global Variables

A global variable is a variable defined in the 'main' program. Such variables are said to have 'global' scope—they can be accessed within any function.

A local variable is a variable defined within a function. Such variables are said to have local 'scope'—they are known only to the function.

Functions can access global variables and modify them.

Modifying global variables in a function is considered poor programming practice. It is better to send a variable in as a parameter (or have it be returned in the 'return' statement).

Modular Programming

Think of a function as a black-box with inputs and outputs.

The function's input/output should be completely defined by its function signature and return statement.

The function should not access (read) or modify global variables directly. Send everything in as a parameter.

A function signature is the function name and parameter list

```
def pow(x,n): # this line is the function signature
    total=1
    count = 0
    while count<n:
        total=total*x
        count=count+1
    return total # in Java the return value must also be specified in the function signature

# main
x=3
exp=2
result = pow(x,exp) # if modular, assume pow effects only result, with no side effects
print result
```

With modular programming, someone reading the program can tell the effect of the function from looking at the call only.

With modular programming, global variables are not changed directly in the function.

The only effect of a function is the return value and modifications of list/object parameters (scalars cannot be modified).

Python's Scope Rules

Inside a function definition,

If a variable is read, Python checks to see if it has been defined locally. If it has not, Python will check to see if there is a global with that name.

```
def func(): # read example
    print x # python will use global x if it exists
```

if a variable is assigned to, a local slot is created for it.

```
def func(): # write example
    x=7 # will not use global even if global 'x' exists.
        # if you want to write to a global variable ,
        # you must use the 'global' statement.
```

```
def func(): # how to write to a global
    global x
    x=7
    x=x+1 # modifies global x
```

Consider the following program snippets. What is the result of each?

```
def f():
    x=10
```

```
x=5
f()
print x
```

```
def f():
    print x
x = 5
```

```
f()
*****
```

```
def f():
    global x
    x=10
```

```
x=5
f()
print x
```

```
*****
```

```
def f():
    x=x+1
x=5
f()
print x
```

```
*****
```

```
def f():

    list.append(5)
```

```
list=[1,2,3,4]
f()
print list
```

```
*****
```

```
def f():
    list=[]
    list.append(5)
```

```
list=[1,2,3,4]
f()
print list
```

```
*****
```

In-Class Worksheet

1. Rewrite pow and the main (see p.1 of these notes) to make it EVIL, that is, NOT MODULAR. Just write it with pencil and paper.
2. Which of the following functions a) works correctly, b) Uses global variables in an un-modular way

```
# 1
def generateSecret():
    secretCode=[]
    i=0
    while i<4:
        color = random.randint(0,5)
        secretCode.append(colors[color])
        i=i+1
# main function call to it
colors=['r','b','g','p','y','b']
generateSecret()
print secretCode;
```

```
# 2
def generateSecret():

    i=0
    while i<4:
        color = random.randint(0,5)
        secretCode.append(colors[color])
        i=i+1
```

```
# main function call to it:
```

```
colors=['r','b','g','p','y','b']
secretCode=[]
generateSecret()
print secretCode
```

```
# 3
def generateSecret():
    secretCode=[]
    i=0
    while i<4:
        color = random.randint(0,5)
        secretCode.append(colors[color])
        i=i+1
    return secretCode
```

```
# main function call to it:
```

```
colors=['r','b','g','p','y','b']
secretCode = generateSecret()
print secretCode
```

3. Rewrite #3 so that it is completely modular