# Namespaces – Local and Global

- The Purpose of Functions
- Global versus Local Namespaces
- The Program Stack
- How Python Evaluates Names

# The purpose of functions

Functions make it easier for us to code solutions to problems. They provide

- Modularity: They allow us to break down complex problems that require complex programs into small, simple, self-contained pieces. Each small piece can be implemented, tested, and debugged independently.

- Abstraction: A function has a name that should clearly reflects what it does. That action can then be performed by calling the function by name, abstracting what the function does from how it does it.

- Code reuse: Code that may be used multiple times should be packaged in a function. The code only needs to be written once. And any time that it needs to be modified, extended or debugged, the changes need to be made only once.

# Local variables hide what goes on inside a function

Enter this code in pythontutor and trace its execution.

```
def double(y):
    x = 2
    y *= x
    print('inside double', 'x = ', x, 'y = ', y)

x, y = 3, 4
print('outside double', 'x = ', x, 'y = ', y)
double(y)
print('after double', 'x = ', x, 'y = ', y)
```

First, `x` and `y` are created in the global frame.

While `double` executes, local `x` and `y` are created with their own values. These local variables cease to exist when the function exits.

Separating what happens inside a function from what happens contributes to your program's modularity.

# Function namespace

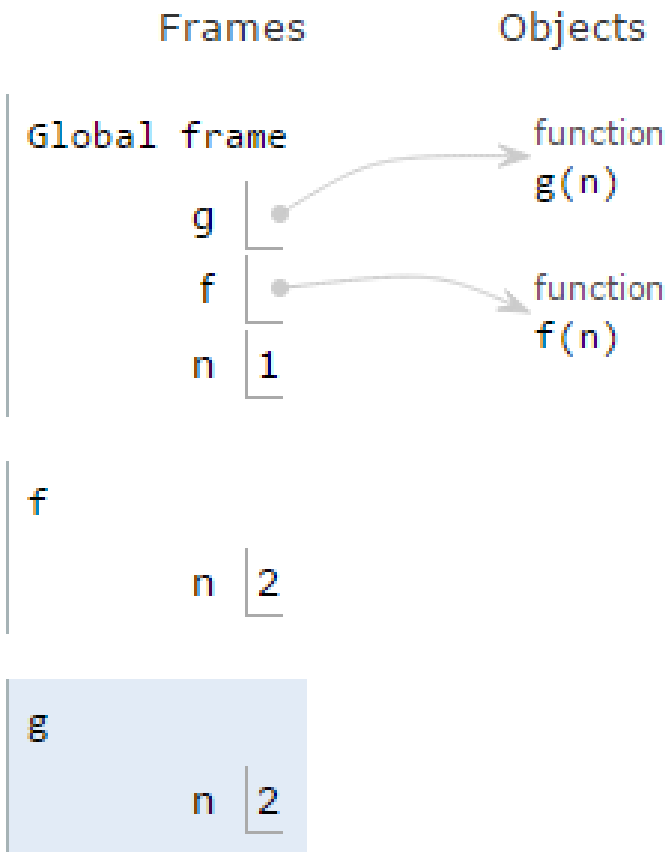
Every execution of a function creates a local namespace that contains its local variables.

In this example there are multiple values of `n`, one in each namespace. How are all the namespaces managed by Python? Which line does Python return to?

```
def g(n):
    print('Start g')
    n += 1
    print('n = ', n)

def f(n):
    print('Start f')
    n += 1
    print('n = ', n)
    g(n)

n = 1
print('Outside a function, n = ',n)
f(n)
```



# Scope and global vs. local namespace

Every function call has its own (local) namespace

- This namespace is where names defined during the execution of the function (e.g., local variables) live.
- This namespace comes into existence when the function is called. It goes out of existence when the function exits (returns).

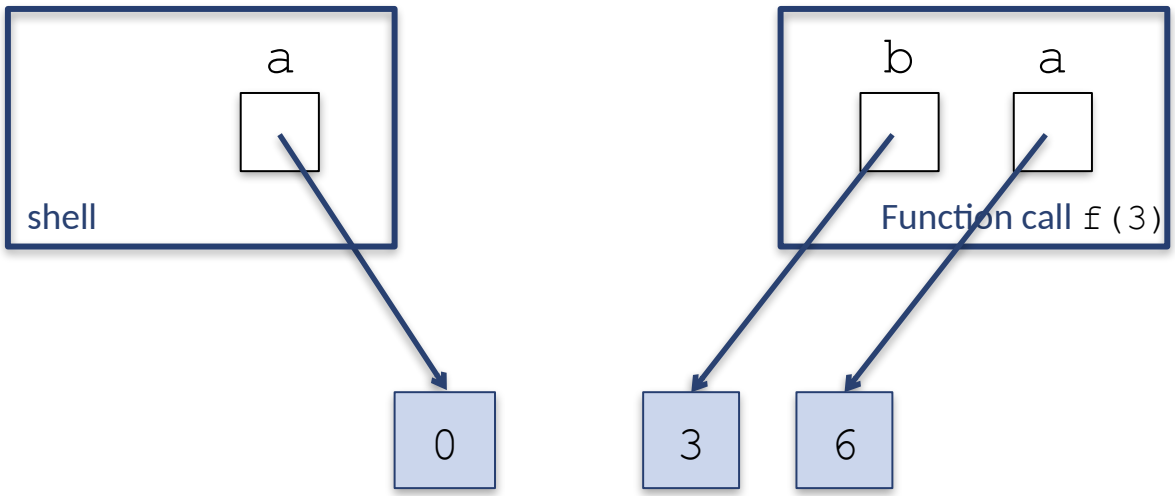
Every name in a Python program has a scope

- This applies to the name of a variable, a function, a class, …
- Outside its scope, the name does not exist. Any reference to it will result in an error.
- Names created in the interpreter shell or in a module and outside of any function have global scope.

# Example: variable with local scope

```
def f(b): # f has global scope, b has local scope
    a = 6 # this a has scope local to this call of function f()
    return a*b  # this a is the local a

a = 0  # this a has global scope
print('f(3) = ', f(3))
print('a = ', a)  # global a is still 0
```

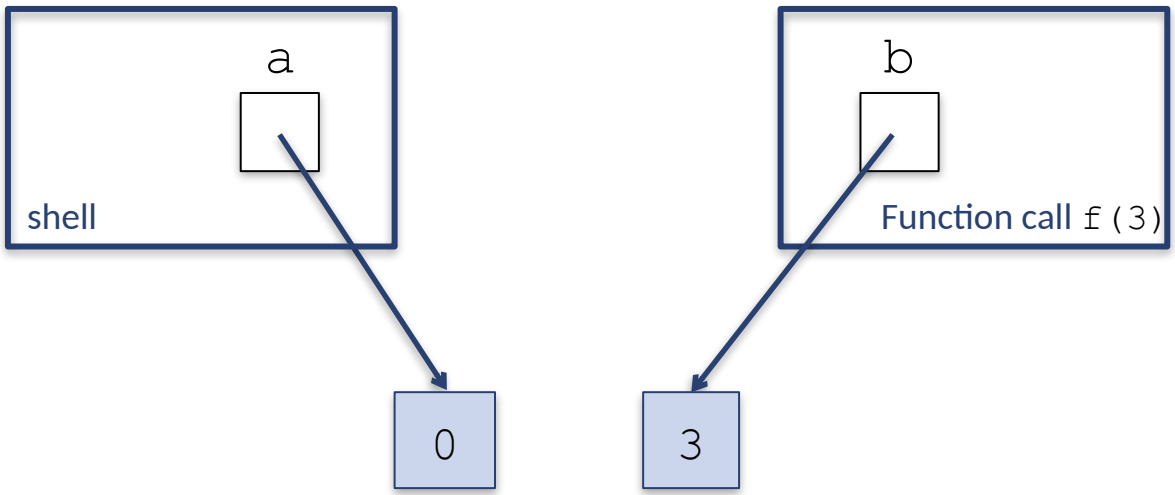



```
>>> === RESTART ====
>>>
f(3) = 18
a = 0
>>>
```

# Example: variable with global scope

```
def f(b):        # f has global scope, b has local scope
    return a*b   # this a is the global a

a = 0            # this a has global scope
print('f(3) = ', f(3))
print('a = ', a)         # global a is still 0
```




```
>>> === RESTART ====
>>>
f(3) = 0
a = 0
>>>
```

# How Python evaluates names

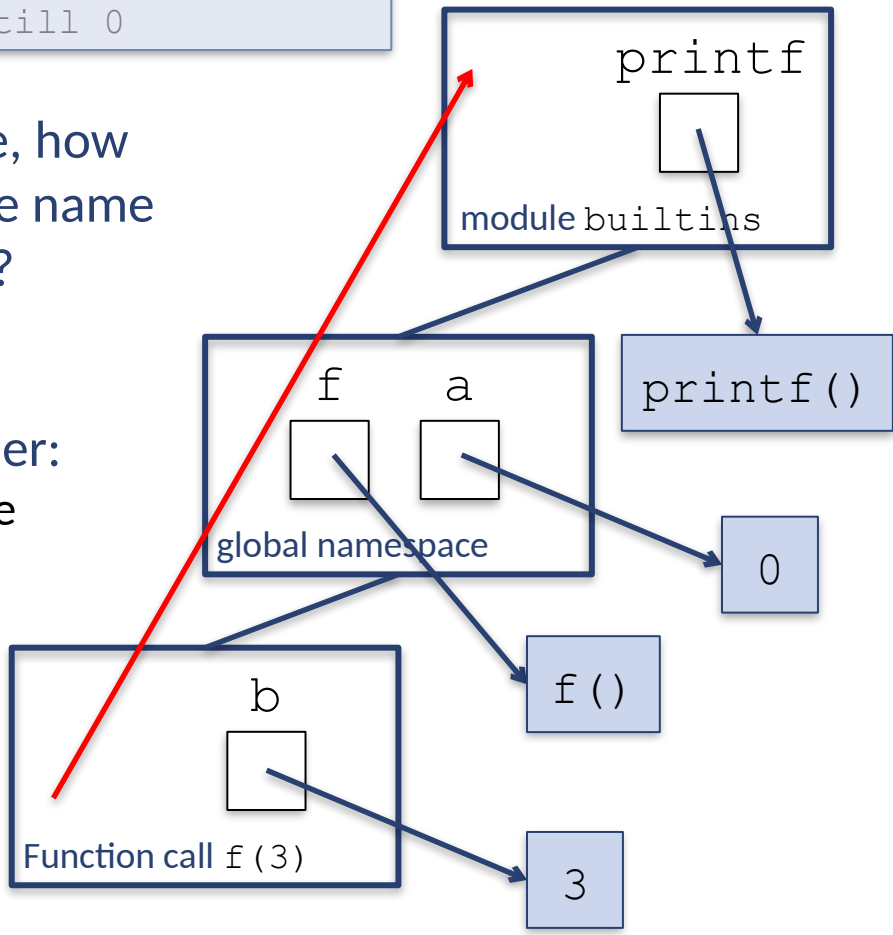
```
def f(b):          # f has global scope, b has local scope
    return a*b     # this a is the global a

a = 0              # this a has global scope
print('f(3) = ', f(3))
print('a = ', a)          # global a is still 0
```

When there are duplicate uses of a name, how does Python decide which instance of the name (e.g., local or global) you are referring to?

To find the value of a name, Python searches through namespaces in this order:

1. First, the local (function) namespace
2. Then the global namespace
3. Finally the namespace of module `builtins`

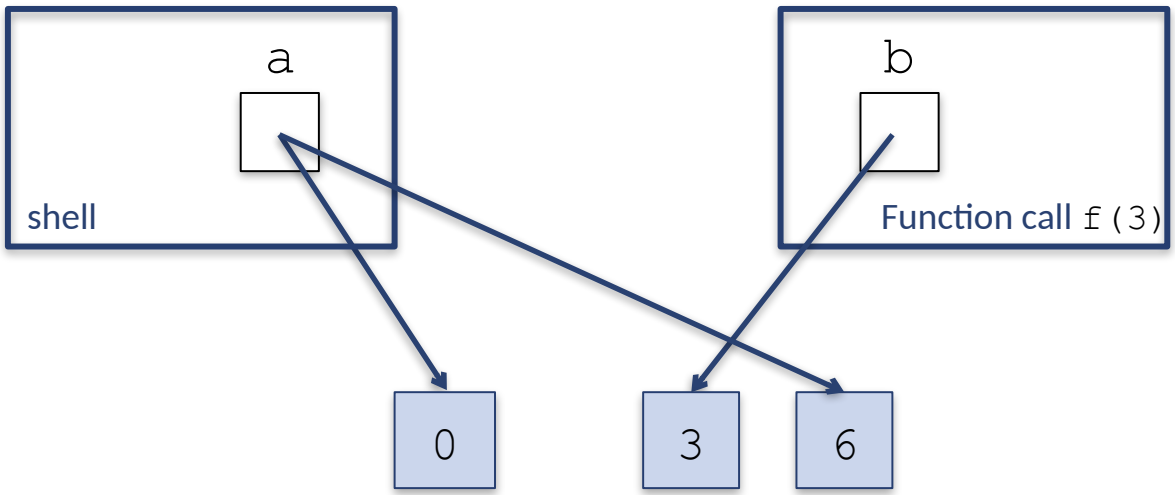


# Modifying a global variable inside a function

To modify a global variable from inside a function, use the keyword 'global'.

```
def f(b):
    global a      # all references to a in f() are to the global a
    a = 6         # global a is changed
    return a*b    # this a is the global a

a = 0             # this a has global scope
print('f(3) = ', f(3))
print('a = ', a) # global a has been changed to 6
```




```
>>> === RESTART ====
>>>
f(3) = 18
a = 6
>>>
```