# Lecture 9:
# Functions II

Stephen Huang

April 10, 2023

# Contents

# 1. Scope of Variables

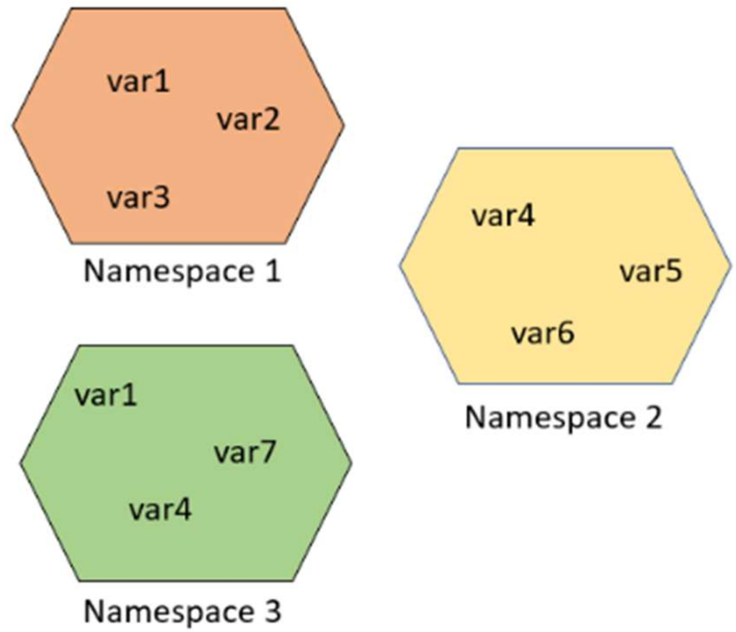
- So far, we have been cautious in using a variable inside a function (mainly parameters).
- We are also careful in returning a value to the calling statement.
- In a function, we did not use or change variables outside the function.  Can we do that?
  - A local variable is a variable defined and used inside a function.
  - A global variable is defined at the top level (outside any function).
- Scope rule.

# Variables in side a function

- There are three types of variables one can use inside a function definition.
  - Variables that are parameters passed to the function,
  - Variables that only exist inside this particular function (local variables), and
  - Variables existed outside the function (global, non-local variables).
- How do we know which type?
  - Parameters are easy to identify,
  - The difference between the other two depends on how we use the variables.

# Namespace

- A Python namespace is a container (of names) where names are mapped to objects.
- A name may exist in a different part of a program.
- The name may be the same, but the object (value associated with the name) may be different.



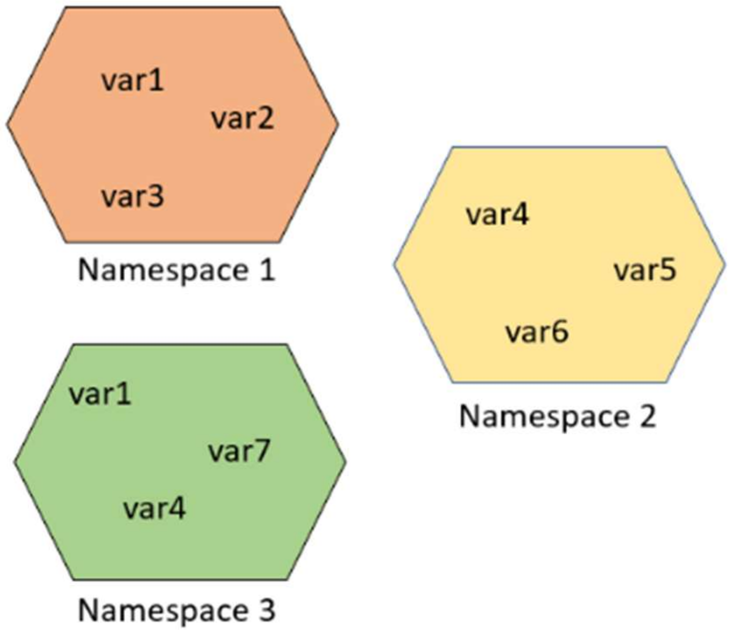

Last_name
City
i, j, k

# Scope

- You can define a name in many places in a program—location matters.
- When you use a name (a variable or a function name), Python searches the program to determine whether the name exists.
- To resolve a name, Python follows a specific order of scope levels.

# Mapping

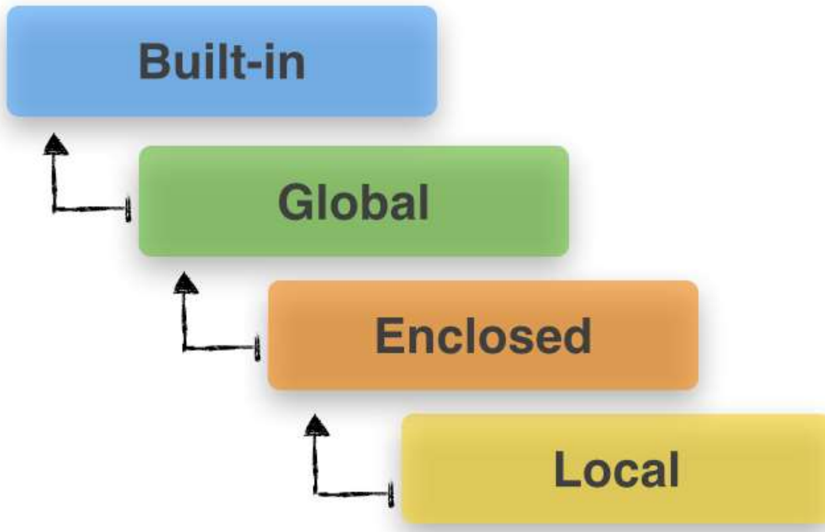
- Everything in Python (literals, lists, dictionaries, functions, classes, etc.) is an object.
- Namespaces are just containers for mapping names to objects.
- The “scope” in Python defines the “hierarchy level” in which we search namespaces for certain “name-to-object” mappings.



# Python Scope

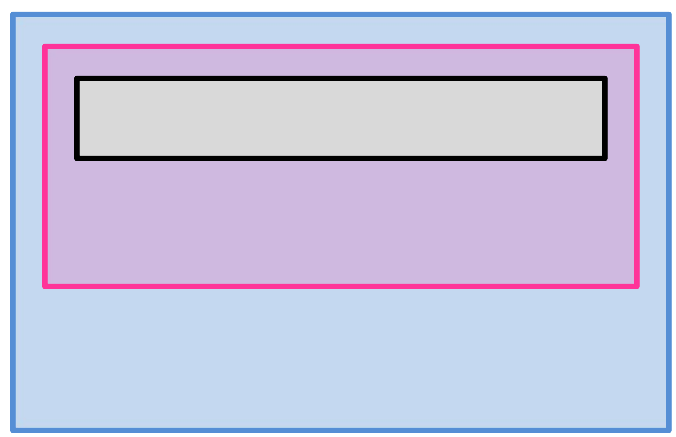
- A scope defines the order in which the namespaces must be searched to obtain the name-to-object (variables) mappings.

- The LEGB stands for
  - Local scope,
  - Enclosing scope,
  - Global scope, and
  - Built-in scope.



# Global and More

- Built-in: Special names that Python reserves for itself.
- Global: before we start using functions, all variables are global.
- Local: all variables defined inside a function are local.
- Enclosed: variable in the enclosing function. This is something new.



# Scopes

- The scope of a variable inside a function definition depends on how it is used.
  - Python assumes that any name assigned to within a function is local to that function unless explicitly told otherwise.
  - If it is only reading (using) from a name that doesn't exist locally, it will try to look up the name in any containing scopes.
- The code over which a variable is accessible or visible is known as the variable's scope.

# Scope Rule

- Suppose we have a variable X in a function; here is how to determine the scope.
  - If X is a parameter, then it is local,
  - Otherwise, if X is assigned a value in the function, it is a local variable (may be used anywhere in the function),
  - If not, check if X is local to a containing block, and stop when found. (not local)

# Local vs. Global

- All the parameters and variables defined in a function are local to the function, meaning that these variables cannot be "seen" by code outside of the function.
- It would be best if you always used parameters to pass data into a function and always use the return statement to export data. Recommended.
- The other way to exchange data with a function is by using global variables, but using a global variable inside a function is considered a dangerous programming practice.

# Constants

- A variable that does not change its value throughout the program is called a constant.
- If many constants are used in many functions, passing them all the time may not be practical.
- It is okay to define all the constants at the beginning of the program and use them throughout the program.
  - Some programmers developed conventions to easily identify variables that are constants (such as SIZE, TAX_RATE). So they know to keep the values unchanged.

# Scope of Variables

- The scope of a variable within a function is from the point it is created either
  - in the parameter list, or
  - in the body via an assignment operation,

  to the end of the function.
- It does not matter whether one made any change to the parameter or the variable. It's their static role that determines the scope.

# Using Local

```python
def f():
    s = 'Go Rockets'
    print(f'Inside  f(): s = {s}')


s = 'Go Coogs'
f()
print(f'Outside f(): s = {s}')
```

```
Inside  f(): s = Go Rockets
Outside f(): s = Go Coogs
```

# Using Global

```
def f():

    print(f'Inside  f(): s = {s}')


s = 'Go Coogs'
f()
print(f'Outside f(): s = {s}')
```

```
Inside  f(): s = Go Coogs
Outside f(): s = Go Coogs
```

# Using Local or Global?

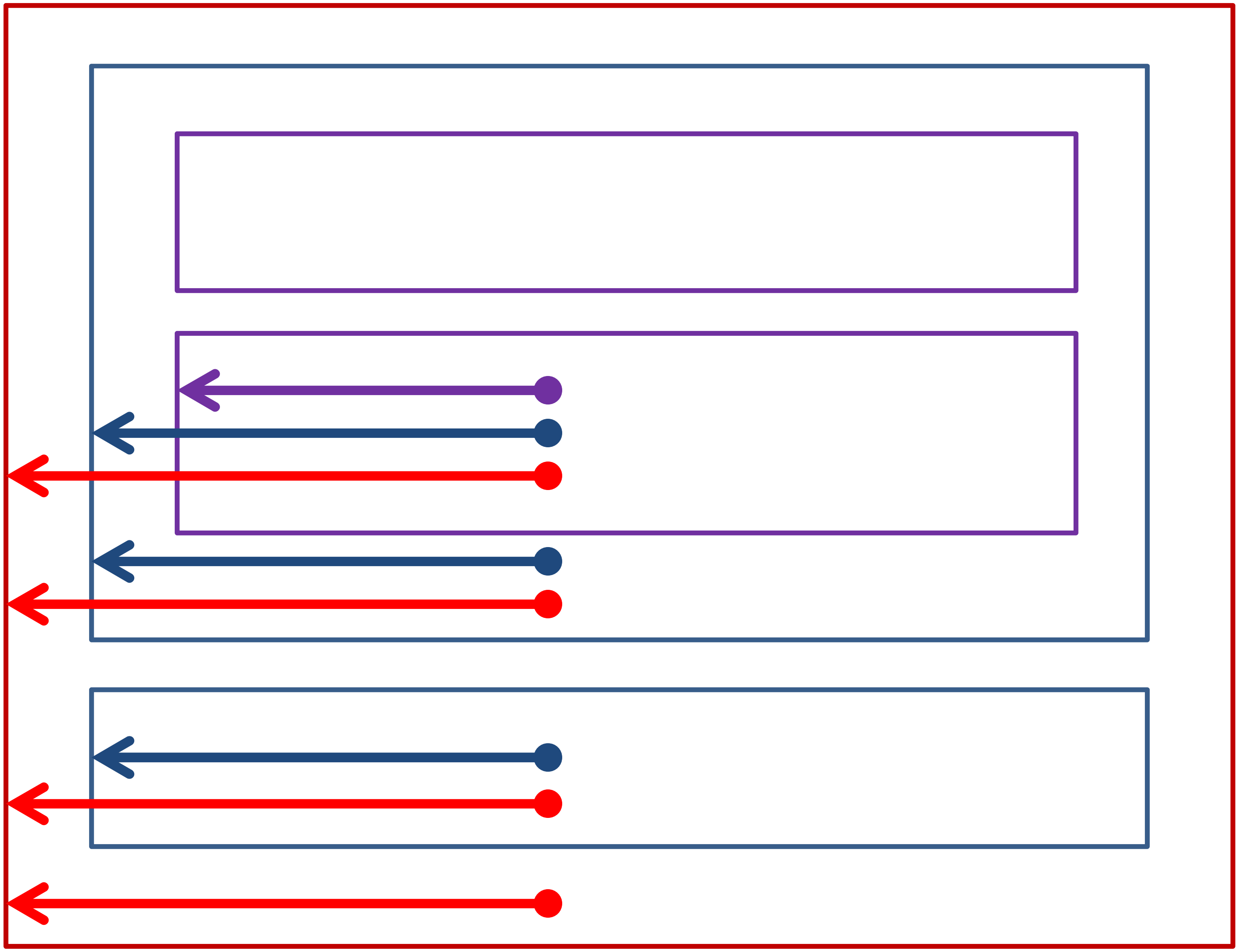
```
def f():

    print(f'Inside  f(): s = {s}')
    s = 'Go Rockets'

s = 'Go Coogs'
f()
print(f'Outside f(): s = {s}')
```

*local variable 's' referenced* ***before*** *assignment*

# Local to what?



# 2. Nested Function

- Python allows the user to define a function within the body of another function.
  - A local function, or
  - An inner function
- The inner function can only be invoked from within the function in which it was defined.
  - Similar to a local variable.
- Not all languages allow the nesting of function definitions as Python does.

# Global, Nonlocal, and Local

- Modify the scope rule.
- The `nonlocal` keyword works with variables inside nested functions, where the variable should not belong to the inner function.
- The `global` keyword specifies global variables from a no-global scope inside a function.
- "Local" is not a keyword in Python.
- To avoid confusion, try not to use the same variable name for different variables.

# But why nested function?

- A function can be defined inside another function.  It is possible to avoid using nested functions, but
- In some cases, there may be some benefits.
  - The inner function can access the variables within the enclosing scope.
  - Two functions may have one inner function each with the same name, such as print_result().

# Scope rules

- The scoping rules for functions are no different than for variables: anything defined inside a function is local to that function.
- Variables and functions defined external to any function have global scope and are visible "everywhere."

# Main()

- For some programming languages, it is required that every program has a function called main().
  - The program execution starts at the main().
- Python does not require the use of a main().
- Some Python programmers put all statements inside one function.
  - Put the statements in the main() function.
  - Call main() as the last (possibly the only) statement. This function call is the sole statement not in a function.
- Main is not a reserved word. A function called main does not have any significance.

# Example: Nonlocal

```
def outer():
    def inner():
        nonlocal x
        print("    inner:", x)
        x = 'defined in inner'
        print("    inner:", x)

    print("  outer:", x)
    x = 'defined in outer'
    inner()
    print("  outer:", x)

x = 'defined in main'
print('main: ', x)
outer()
print('main: ', x)
```

```
main:  defined in main
  outer: defined in outer
    inner: defined in outer
    inner: defined in inner
  outer: defined in inner
main:  defined in main
```

# Example

```python
def func():

    x = 'Hi'


x = 'Welcome'
print(x)
func()
print(x)
```

Welcome
Welcome

```python
def func():
    global x
    x = 'Hi'


x = 'Welcome'
print(x)
func()
print(x)
```

Welcome
Hi

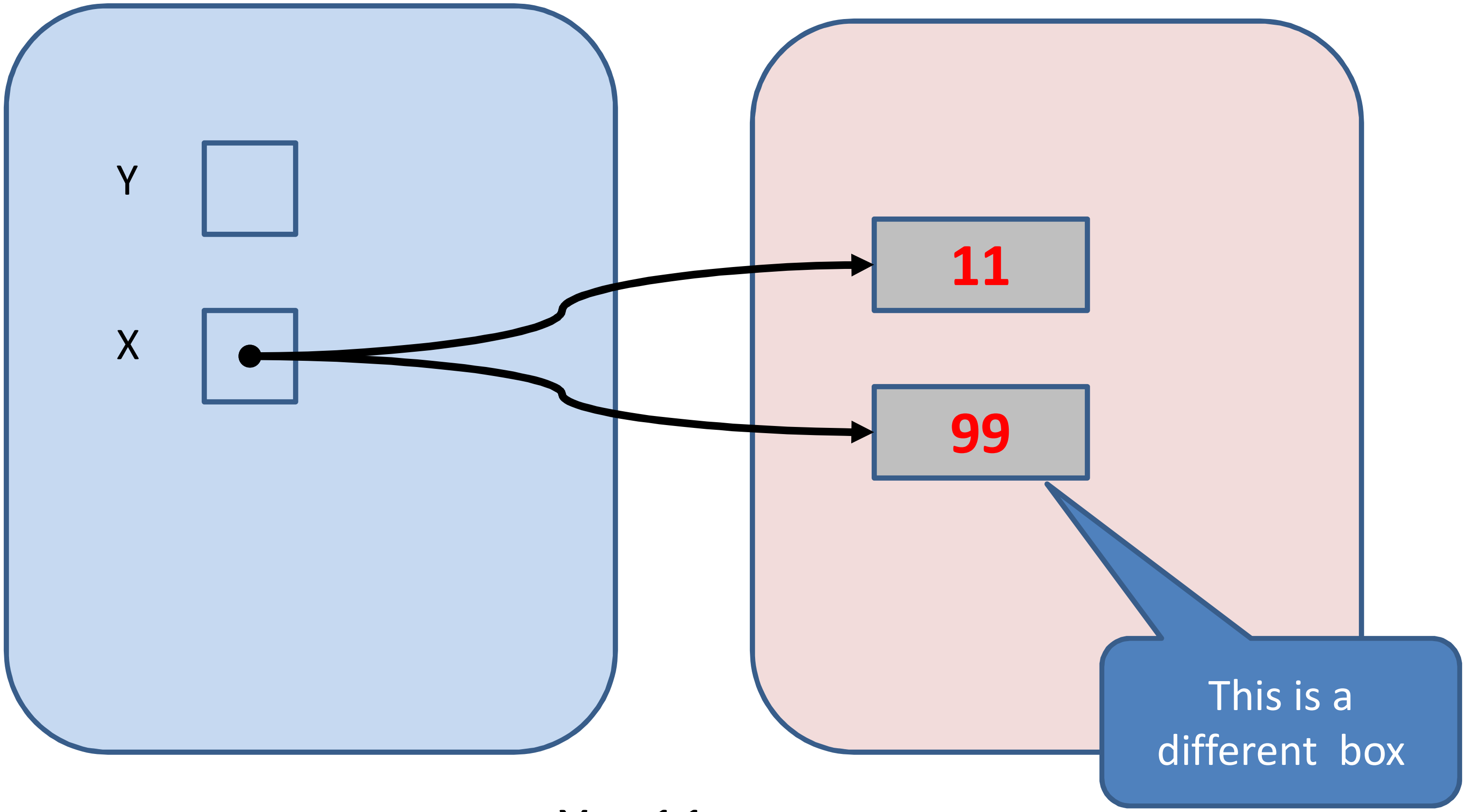
# 3. Mutable & Immutable Objects

- A review of mutable and immutable variables.
- A general explanation from the "Data Model" chapter in the Python Language Reference":
  - The value of some objects can change.
  - Objects whose value can change are said to be mutable;
  - Objects whose value is unchangeable once they are created are called immutable.

# Mutable Parameter

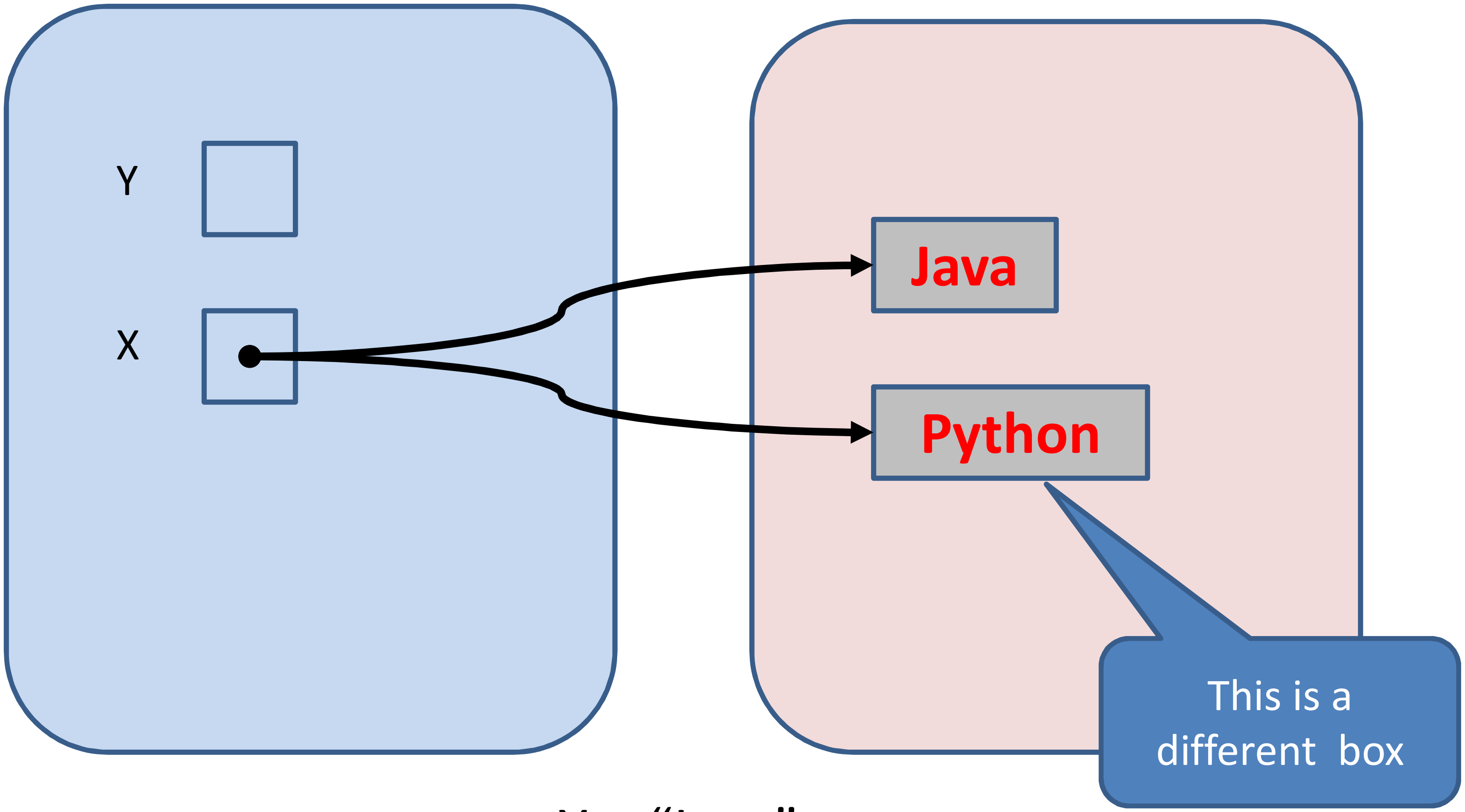
- We cannot change a parameter in a function. If we do change it, it becomes a local variable and not associated with the parameter anymore.
- For a parameter of a mutable type, such as a list, we cannot change the parameter (the list), but we can change some of the components in the structure.
- What is going on?

# Immutable Assignment



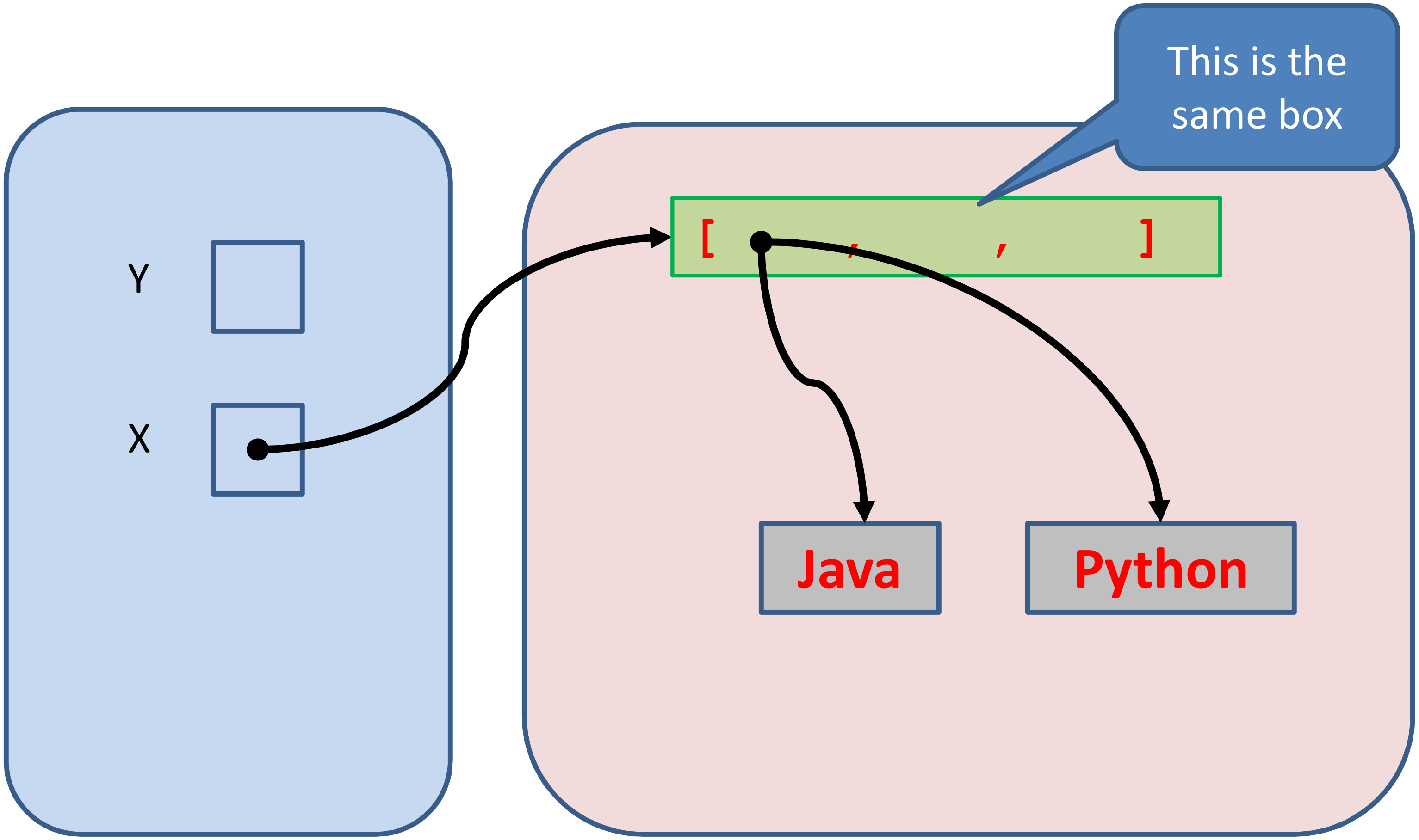

X = 11
X = 99

# Immutable Assignment




X = "Java"
X = "Python"

# Mutable Assignment




X = [“Java”, “C++”, “Pascal”]
X [0] = “Python”
X.append(…)

# Summary

- If we have a mutable variable such as a list,
  - We cannot change the list object, but
  - We can change elements inside the list
- What if we pass a mutable variable to a function?

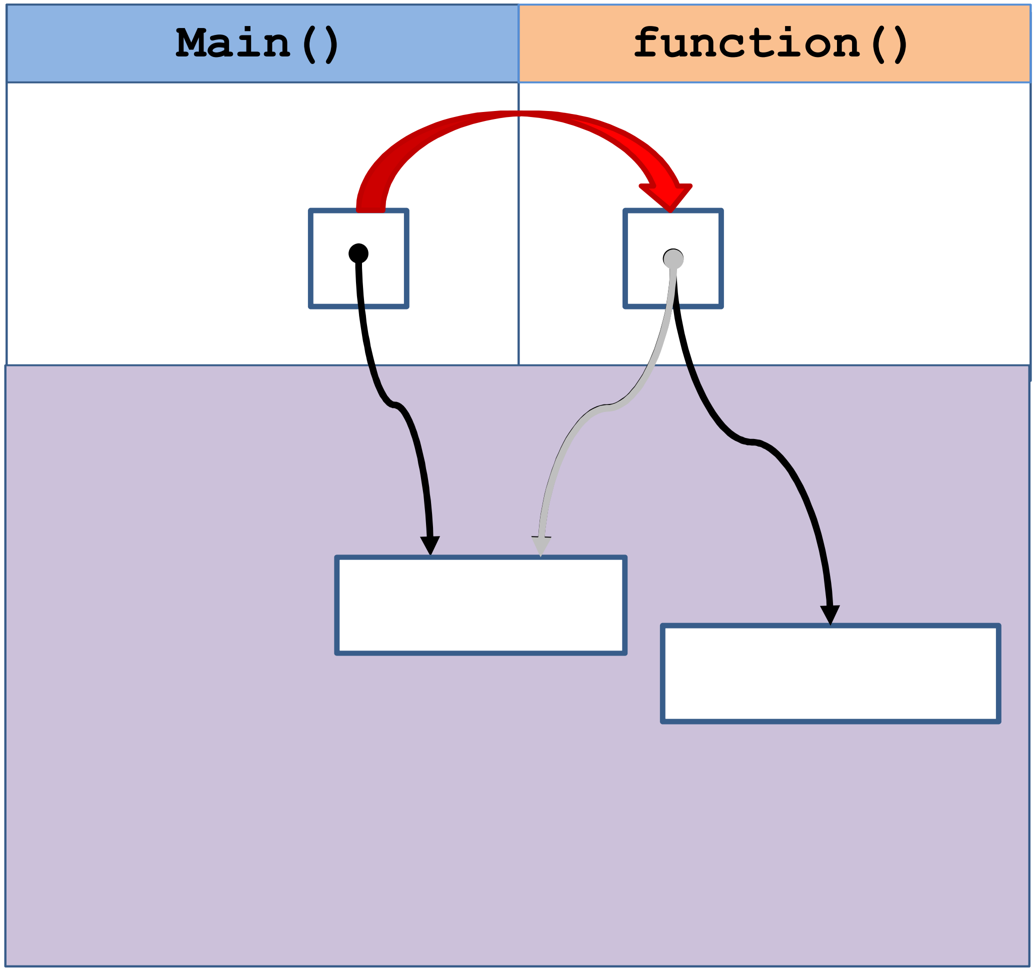
# 4. Parameter Passing

- The process behind parameter passing in Python is simple: the function call binds to the formal parameter, the object referenced by the actual parameter.
- The kinds of objects we have considered so far—integers, floating-point numbers, and strings—are classified as immutable objects.
- This means a programmer cannot change the value of the object.
- Parameters are "pass-by-value".

# Parameters

- Changing the parameter inside a function does not change the actual parameter in the calling function.
- The actual parameters may be an expression (or constant) that cannot receive a value anyway.
- Other languages have a different way of passing parameters (call-by-reference), which allows a statement inside a function to cause change to an actual parameter (which must be a variable).

# Pass-by-value

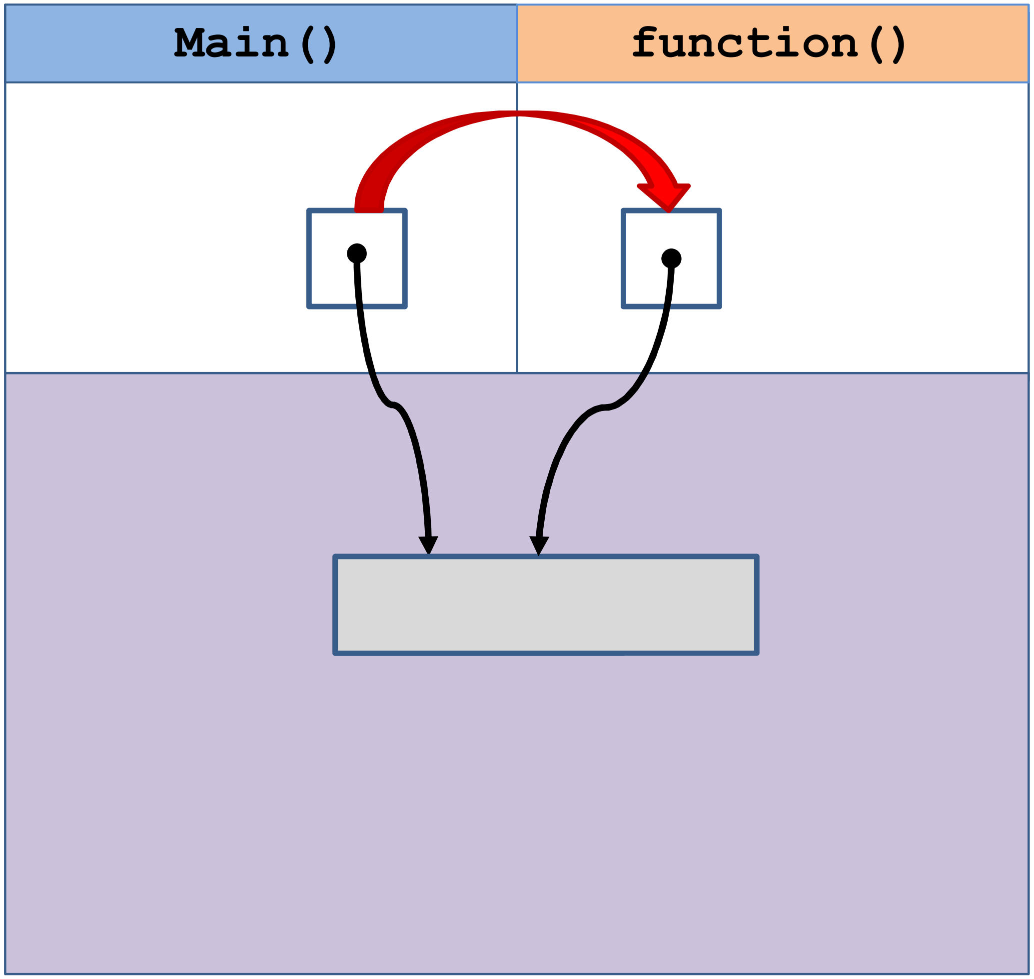



When value changes:

If the parameter is of immutable type, then a new space is allocated for the new value.

If the parameter is of mutable type, then the value is changed

# Pass-by-value




When value changes:

If the parameter is of immutable type, then a new space is allocated for the new value.

If the parameter is of mutable type, then the value is changed

# Passing Changes Back

- There are three ways to change, from inside a function, the values in the calling function.
  - Use a global variable—very bad idea. Read from the global variable is bad; changing the global variable is very bad. (Exception: Constants)
  - Use return and assign to the variable. This is the recommended way. Remember that we can return multiple values, an improvement over other languages.
  - Make a change to the parameter if it is mutable. Use this if you know what you are doing. There is no need to use a return in this case.

# Cheating the System

- We can use a mutable wrapper y to contain an immutable variable x
- Pass the wrapper y to a function
- Change the value of x
- Upon return, take off the wrapper
- The value has been changed.

# 5. Functions as an Argument

- You can think about a method (or function) as a variable whose value is the actual callable code object.
- We can pass functions as arguments to other functions.

# Example

```python
def foo(f, para):
    print(f"Calling {f}(\"{para}\") inside foo().")
    f(para)

def bar(para):
    print(f"Inside bar(\"{para}\").")

bar("Hello world!")
foo(bar, "Howdy")
```

```
Executing f inside foo().
Inside bar("Hello world.").
Inside bar("Hello world!!!").
```

# Example

```python
def norm(s):
    return s.casefold()

fruits = ['cherry', 'banana', 'Apple',
          'Pear', 'Watermelon', 'peach']

print(sorted(fruits))
print(sorted(fruits, key=norm))
```

```
['Apple', 'Pear', 'Watermelon', 'banana', 'cherry', 'peach']
['Apple', 'banana', 'cherry', 'peach', 'Pear', 'Watermelon']
```

# 6. Default Arguments

- Sometimes an argument of a function has values that are the usual values in most calls.
- Python allows the programmer to indicate the usual (default) values.
- Any parameters that have default values must be the rightmost parameters in the parameter list.
- If one or more of the arguments to the function are missing, then the default value of the corresponding parameter is used.

# Default Arguments

- If two parameters have default values and only one of the arguments is missing, then the rightmost of the arguments are assumed to be the missing one.
- Python, like other languages, provides support for default argument values, that is, function arguments that can
  - either be specified by the caller, or
  - left blank to automatically receive a predefined value.
- All standard arguments first, then the default ones.

# Examples

- The following example below illustrates default values for parameters.
  - Suppose a function is used to compute the cost of putting in a concrete driveway.
  - Suppose the lengths of driveways are different, but the width and depth of driveways are usually 6.5 feet wide and 0.5 feet deep.
  - Then we could write the function that computes the cost with these as default values.

# Example

```
def cost(unitCost,len,w=6.5,d=0.5):
    return unitCost*len*w*d

print(cost(100, 10, 7, 1))
print(cost(100, 10, 7))
print(cost(100, 10))
```

```
7000
3500.0
3250.0
```

# 7. Recursive functions

- We have introduced the concept in Lecture 3.
- More examples are here.

**Skip this page**

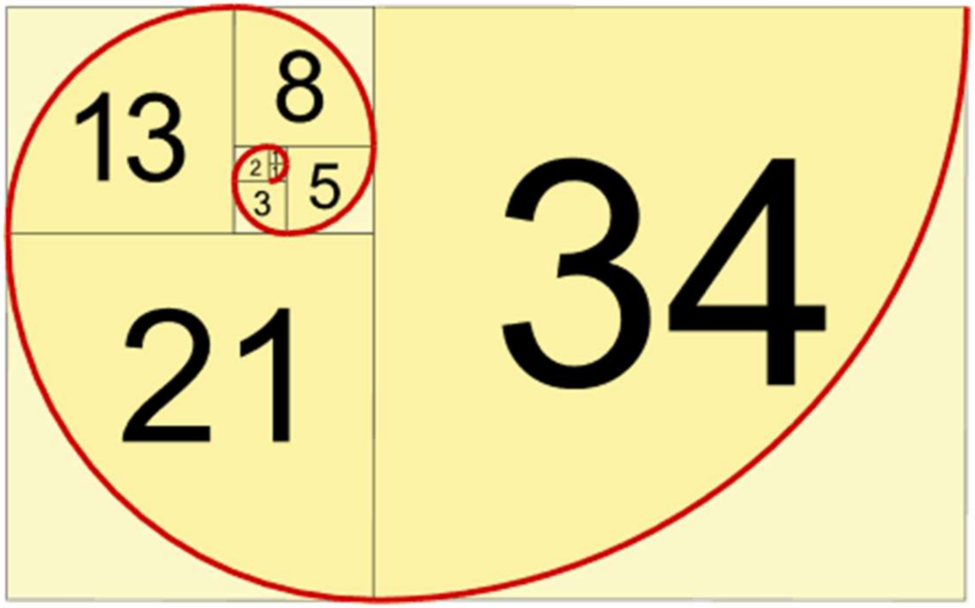
# Factorial

```
def factorial(n):
    if n<1:
        return None
    elif n==1:
        return 1
    else:
        return factorial(n-1)*n
```

Factorial.py

**Skip this page**

# Fibonacci Numbers




**Skip this page**

# Fibonacci Numbers

```
def fib(n):
    if n==0:
        return 0
    elif n==1:
        return 1
    else:
        return fib(n-1)+fib(n-2)
```

**Skip this page**

# 8. Lambda Functions

λ

- A lambda function is a small anonymous function.
  - Small: the body is an expression
  - Anonymous: without a name
  - May have parameters/arguments
- Syntax:
  - Lambda <arguments> : <expression>
- Pretty much anything you can do with lambda function, you can do better with a named function. You don’t absolutely need it.

# A Comparison

```
def function_name(<parameters>) :
        <statements>
        return <expression>

lambda <parameters> : <expression>
```

# An Example

```
def double (x) :
    return (x*2)
```

```
lambda x : x*2
```

# Using Lambda

- It is possible to use a lambda function directly.
- In many cases, we do have to give an anonymous function a name so we can use it. See the following example.
- A lambda function is typically only used in <u>one place</u> and does just <u>one thing</u>.
- One may avoid using this feature. However, you may want to know what it is when you see one.

# Physical Comparison

```
def main():
…
…
y = square(some_num)
…
return something
…
…
# Many lines later
def square(x):
    return x**2
```

```
def main():
…
…
square = lambda x: x**2
y = square(some_num)
…
return something
```

**Skip this page**

# Sorting Tables

```
>>> student_tuples = [
... ('john', 'A', 15),
... ('jane', 'B', 12),
... ('dave', 'B', 10),
... ]

>>> sorted(student_tuples, key=lambda
student: student[2]) # sort by age

[('dave', 'B', 10), ('jane', 'B', 12),
('john', 'A', 15)]
```