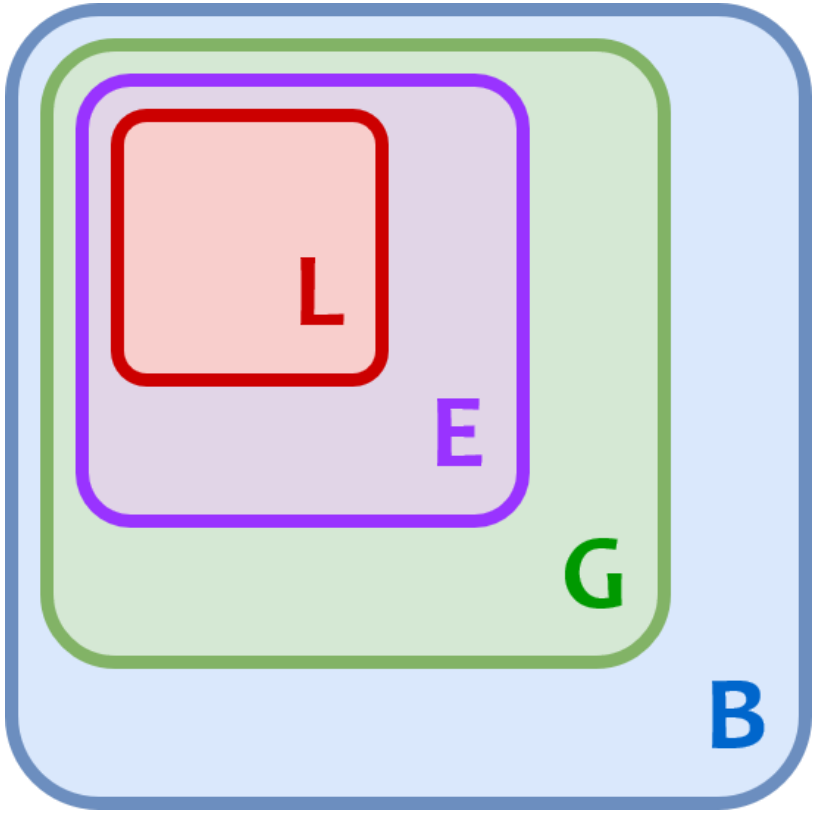
Programming in Python

Scope, Modules, Dates and Strings

Variable Scope and Namespaces

Python resolves the scope through “Namespaces”

- Local Scope
- Enclosing Scope
- Global Scope
- Built-in Scope



Order of resolution

[1] Is there a LOCAL variable? (L)
[2] Is there a variable in my ENCLOSING function? (E)
[3] Is there a GLOBAL variable? (G)?
[4] Is there a python BUILT-IN variable? (B)

https://realpython.com/python-namespaces-scope

Summary on “Variable Visibility” (aka Scope)

PYTHON PROGRAMMING by SIVA JASTHI

Variable Scope: refers to the area of the program where a variable is accessible. It is of two types.

Local Scope: Variables defined inside a function have a local scope.
They are only visible and accessible within the function in which they are defined.

Global Scope: Variables defined outside any function or block have a global scope.
They can be accessed from anywhere in the program, including inside functions.

What does python think?

- Do I have any local variables? If yes, use it.
- If not, look for a global variable!

Rebinding of a variable: (Redefining, Redeclaring, Using the same variable name → local) (shadowing a global)

Rebinding a variable inside a function means creating a new local variable with the same name as an existing global variable. When you assign a value to a variable inside a function with the same name as a global variable, it creates a new local variable that shadows (hides) the global variable inside the function's scope. This local variable is separate from the global variable, and changes made to it inside the function do not affect the global variable.

When to use “global” keyword?

You use the global keyword in Python when you want to indicate that a variable inside a function should refer to the global variable of the same name, rather than creating a new local variable with the same name. This is necessary when you want to modify the value of a global variable from within a function.

Scope

A variable is only available from inside the region it is created. This is called **scope**.

Local Scope:
x is visible only to “myfun( )”.

```
def myfunc():
  x = 300
  print(x)

myfunc()
```

Global Scope:
x is visible everywhere.

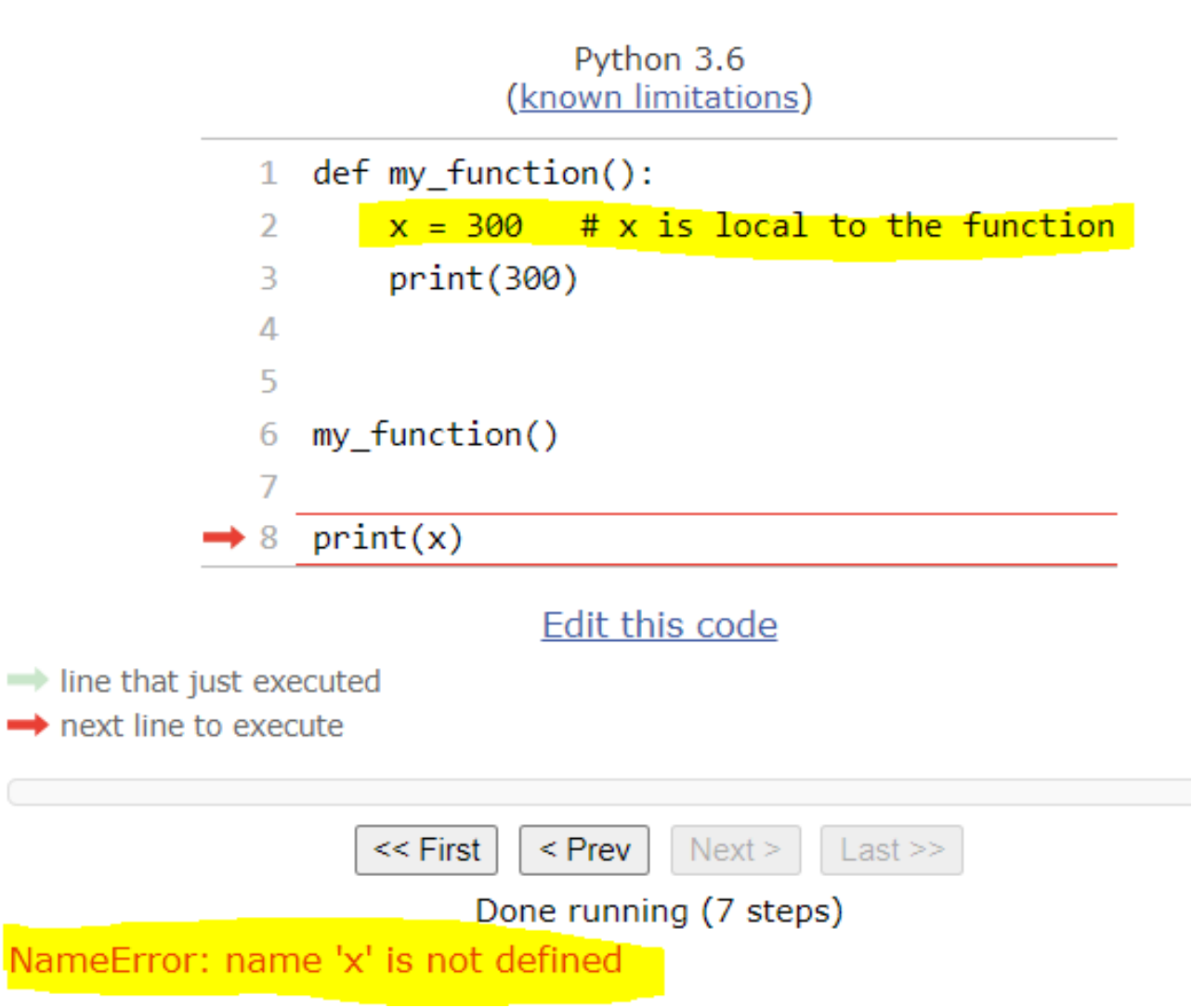
```
x = 300

def myfunc():
  print(x)

myfunc()

print(x)
```

https://www.w3schools.com/python/python_scope.asp

Scope: Local variables

Local variables are visible only to the functions.

You can not access those outside of the function.



Enclosing Scope

Sometimes, a function may define some inner functions.

In such case, the outer function is called “Enclosing Function”.

```
def myfunc():
  x = 300
  def myinnerfunc():
    print(x)
  myinnerfunc()

myfunc()
```

https://www.w3schools.com/python/python_scope.asp

Scope & Variable Naming

What happens if you a local variable and global variable with the same name?

If you operate with the same variable name inside and outside of a function, Python will treat them as two separate variables, one available in the global scope (outside the function) and one available in the local scope (inside the function):

```
x = 300

def myfunc():
  x = 200
  print(x)

myfunc()

print(x)
```

What is printed here?

What is printed here?

https://www.w3schools.com/python/python_scope.asp

Best Practice:

To avoid confusion, just the name the variables differently so that those don't clash with the global variable names.

In OO programming, having global variables is also not a good idea

Scope & Variable Naming (Contd.)

How can we access the global variable inside a function if that function also has a variable with the same name?

Prefix the keyword “global”

```
x = 200

def myfunc():
  global x
  x = 300

myfunc()

print(x)
```

What is printed here?

What is printed here?

https://www.w3schools.com/python/python_scope.asp

Best Practice:

To avoid confusion, just the name the variables differently so that those don’t clash with the global variable names.

In OO programming, having global variables is also not a good idea

LEGB Scope (link)


```python
# LOCAL, ENCLOSING, GLOBAL

x = 200

def change1():
    x = 300  # still local to change1

def change2():
    global x
    x = 500  # it now became global

def my_function():
    x = 400   # x is local to the function
    print(x)

    def my_inner_function():
        print(x+1)  # my_function is enclosing function
                    # inner functions can access variables from enclosing

    my_inner_function()

# let us print x
print(x)

my_function()
print(x)

change1()
print(x)

change2()
print(x)
```

Best Practice:

To avoid confusion, just the name the variables differently so that those don't clash with the global variable names.

In OO programming, having global variables is also not a good idea

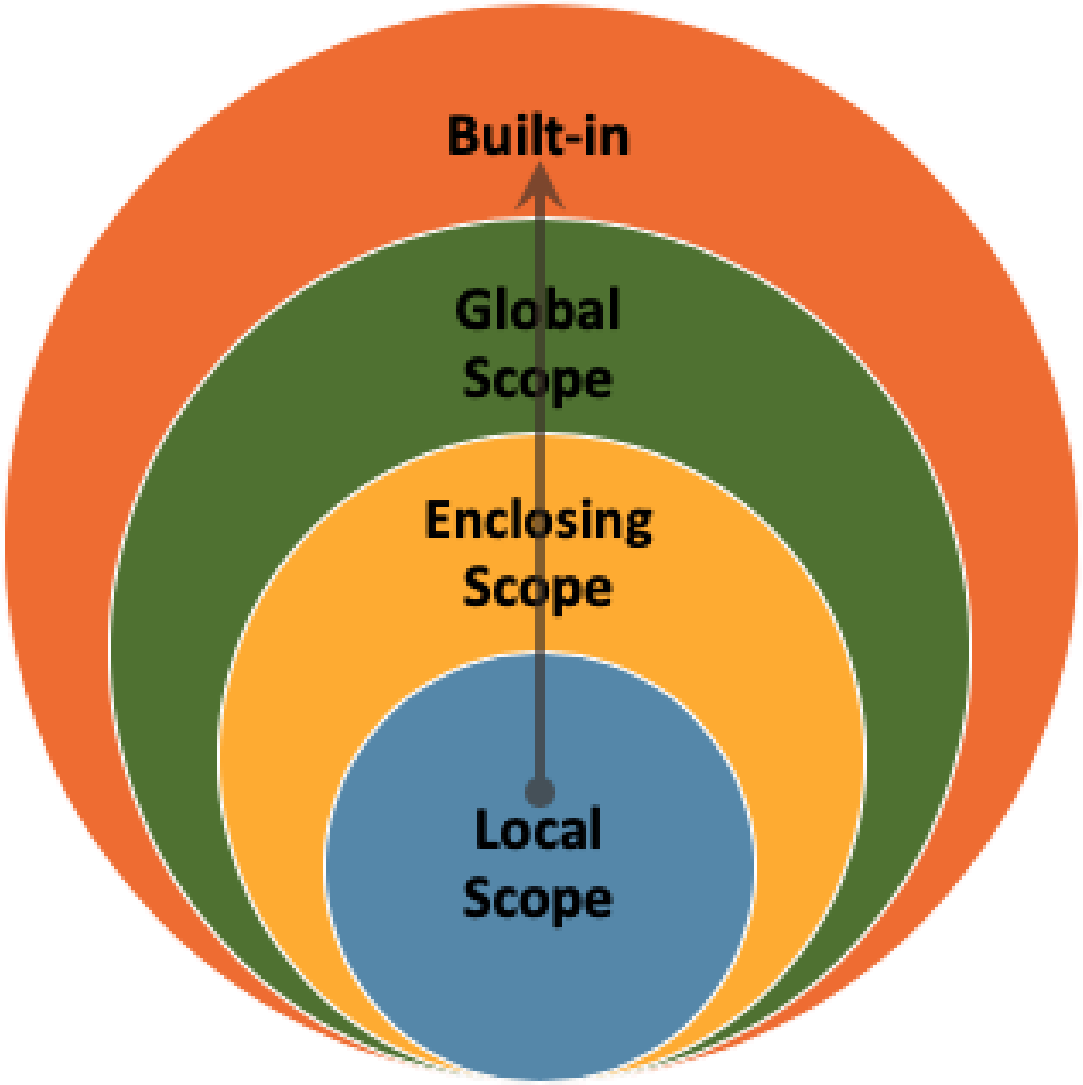
Variable Scope and Namespaces

You can print the __builtins__ using the built-in dir function.

```
x = dir(__builtins__)
print(x)

y = dir(__builtins__)
for i in y:
    print(i)
```

['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'RecursionError', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError', '__build_class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__', '__spec__', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'breakpoint', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec', 'exit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']

LEGB Rule and Variable Scope



LEGB Rule and Variable Scope

Local Scope: This is the innermost scope and refers to the variables defined within the current function or method. Local variables can only be accessed within the function or method in which they are defined.

Enclosing Scope: Also known as the non-local scope, this refers to the variables defined in the enclosing function or method. In other words, it includes the local scopes of the parent functions or methods. Variables in the enclosing scope can be accessed by the inner functions or methods.

Global Scope: This scope includes variables defined at the top level of a module or declared as global within a function. Global variables can be accessed from anywhere within the module.

Built-in Scope: This is the outermost scope and includes the built-in functions and modules that are available by default in Python, such as print(), len(), and range(). The built-in scope is always accessible from any part of the code.

LEGB Rule and Variable Scope

When a variable is referenced in a Python program, the interpreter follows the "LEGB" rule to determine its scope resolution:

Local scope is checked first. If the variable is found in the local scope, it is used.
If not found in the local scope, the enclosing scope is checked.
If still not found, the global scope is checked.
If not found in the global scope, the built-in scope is checked as the last resort.
If the variable is not found in any of these scopes, a NameError is raised, indicating that the variable is undefined.

It's important to note that assigning a value to a variable within a scope creates a new local variable, even if a variable with the same name exists in an outer scope. However, if you want to modify the value of a variable from an outer scope, you need to use the global or nonlocal keywords to indicate your intention explicitly.

Summary

Scope (LEGB Rule)
- Local
- Enclosing
- Global
- Built-In

Modules
- Creating modules
- Importing Modules
- dir( ) function
- Namespaces
- Running a module as a script
- Built-in Modules

String Formatting

Thank You. धन्यवाद