# 19CSE201 :Advanced Programming

# Lecture 21
# Namespaces & Scope in Python

By
Ritwik M
Assistant Professor(SrGr)
Dept. Of Computer Science & Engg

Based on material from Matthes, E. (2019). Python crash course: A hands-on, project-based introduction to programming. No Starch Press and multiple sources from the internet

# A Quick Recap

- Loops
- For and While
- Functions
- Arguments
- Recursion

# An Experiment

- Fire up HPOJ and in the IDE type
  - `Import this`
- What did you see?
- Read it loudly – Especially the last line

# What is Name in Python?

- Name (also called identifier) is simply a name given to objects.
- Everything in Python is an object.
- Name is a way to access the underlying object.
  - `a = 2`
- Here `2` is an object stored in memory and `a` is the name we associate it with.
- We can get the address of `a` using the `id()` function

```
a = 2
print('id(2) =', id(2))
print('id(a) =', id(a))
```

```
Output:
id(2) = 10914528
id(a) = 10914528
```

Here both refer to the same object so same ID

# Name Binding

```
a = 2
print('id(a) =', id(a))

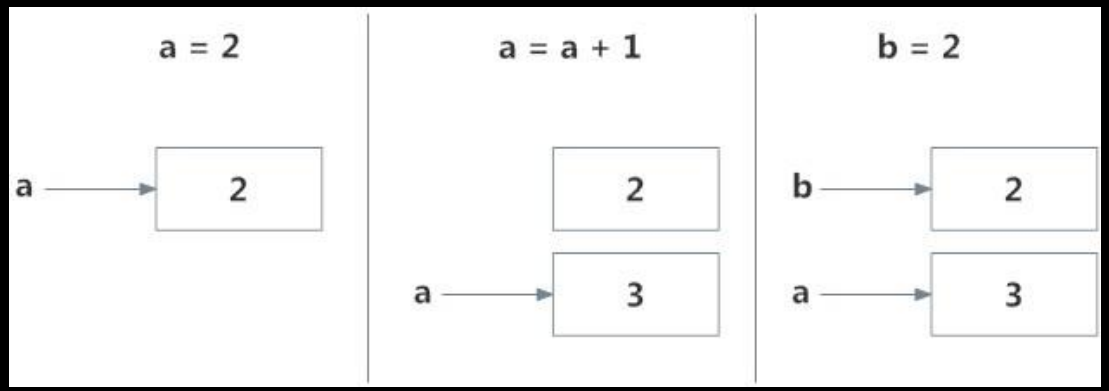
a = a+1
print('id(a) =', id(a))

print('id(3) =', id(3))

b = 2
print('id(b) =', id(b))
print('id(2) =', id(2))
```

- Output

```
id(a) = 10914528
id(a) = 10914560
id(3) = 10914560
id(b) = 10914528
id(2) = 10914528
```




- This is efficient as Python does not have to create a new duplicate object
- This dynamic nature of name binding makes Python powerful;

# Name Binding Cont.

- Remember, in python everything is an object.
- What about functions? - They are objects too.
    - `def printHello():`
    - `    print("Hello")`
    - `a = printHello`
    - `a()`
- The same name a can refer to a function and we can call the function using this name.
- Can you map this functionality to any OOP concept?

# From Names to Namespaces

- A namespace is a collection of currently defined symbolic names along with information about the object that each name references.
- Imagine it as a mapping of every name you have defined to the corresponding objects.
- In a Python program, there are four types of namespaces:
  - Built-In
  - Global
  - Enclosing
  - Local


What about enclosing namespaces?
Hint: Nesting

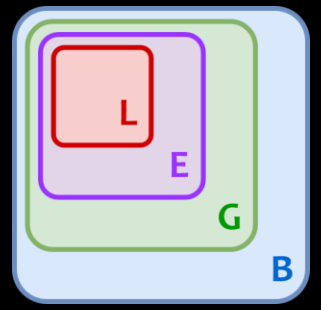
- These have differing lifetimes, can co-exist at a given time but are completely isolated.

Refer the textbook for a very thorough discussion of variable scope and namespaces in programming languages.

# Namespaces Cont.

- A namespace containing all the built-in names is created when we start the Python interpreter and exists as long as the interpreter runs.
- Unlike C++, in Python the namespace is created automatically
- Each module creates its own global namespace.
- Modules can have various functions and classes.
  - A local namespace is created when a function is called, which has all the names defined in it. Similar, is the case with class.

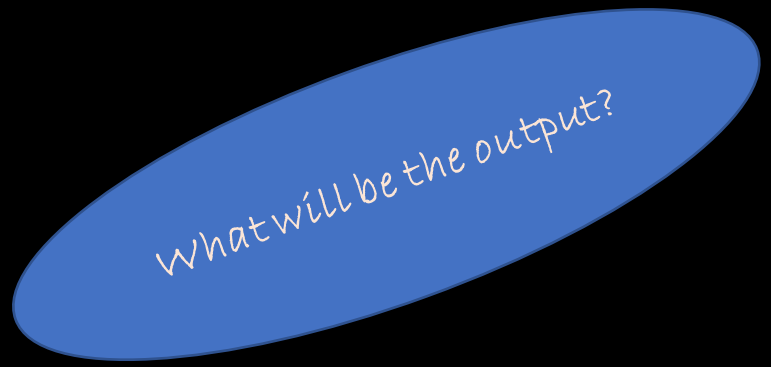
# Python Variable Scope




- A scope is the portion of a program from where a namespace can be accessed directly without any prefix.
- The interpreter determines this at runtime based on where the name definition occurs and where in the code the name is referenced.
- Python searches for a name x in the following namespaces in the order shown:
  - Local:
    - If you refer to x inside a function, then the interpreter first searches for it in the innermost scope that's local to that function.
  - Enclosing:
    - If x isn't in the local scope but appears in a function that resides inside another function, then the interpreter searches in the enclosing function's scope.
  - Global:
    - If neither of the above searches is fruitful, then the interpreter looks in the global scope next.
  - Built-in:
    - If it can't find x anywhere else, then the interpreter tries the built-in scope.

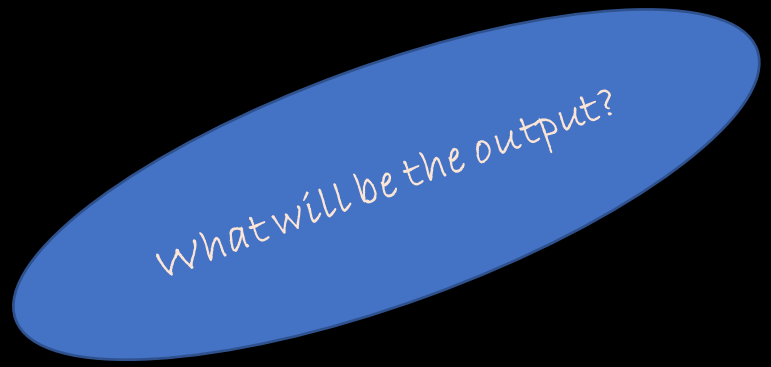
# Exercise 1

```
x='global'
def f():
    x='enclosing'
    def g():
        x='local'
        print(x)
    g()
    print(x)
f()
print(x)
```



# Exercise 2

```
def f():
    x='enclosing'
    def g():
        x='local'
        print(x)
    g()
f()
```



# Exercise 3

- What happens when you type the following on the HPOJ terminal
  - `print(global())`
  - ```
    def f(x,y):
        s= 'foo'
        print(locals())
    f(10,0.5)
    ```
- Build another example showing the various scopes of the variables
- Explore the built-in scope as well
- Additional reading: Python Namespace Dictionaries
    ```

# Quick Summary

- Names
- Name Binding
- Namespaces
- Variable Scope
- Examples
- Exercises

# Up Next

Classes and Objects