Closures

Closures

Python's functions are *first-class* objects. You can assign them to variables, store them in data structures, pass them as arguments to other functions, *define inside* other functions, and even *return them as values* from other functions.

Closures

```
>>> def fabric_function(number):
...     """ Enclosing function """
...     def print_function():
...         """ Nested function """
...         print(number)
...
...     return print_function
```

Closures

```
>>> def fabric_function(number):
...     """ Enclosing function """
...     def print_function():
...         """ Nested function """
...         print(number)
...
...     return print_function
```

```
>>> print_1 = fabric_function(1)
... print_2 = fabric_function(2)
... print_3 = fabric_function(3)
```

Closures

```
>>> print_1 = fabric_function(1)
... print_2 = fabric_function(2)
... print_3 = fabric_function(3)

>>> print_1()
... print_2()
... print_3()
1
2
3
```

Closures

```
>>> print(fabric_function.__closure__)
None

>>> print(print_1.__closure__)
(<cell at 0x7fcac5aeeaf8: int object at 0x108f285a0>,)

>>> print(print_1.__closure__[0].cell_contents)
1
```

Closures

```
>>> def log_prefix(prefix):
...     def log(text):
...         print(f'{prefix}: {text}')
...     return log


>>> log_info = log_prefix("INFO")
>>> log_info("Call status Ok")

INFO: Call status Ok
```

Overview

- The closure is a function, or more strictly speaking, **an inner function**, which is defined within the scope of the other function (termed outer function).

- The inner function **binds variables defined outside of its own scope**.

Namespaces and scopes

Symbolic names

Namespace

A ***namespace*** is a collection of currently defined symbolic names along with information about the object that each name references

Scope

A ***scope*** defines which namespaces will be looked in and in what order.

The ***scope*** of any reference always starts in the local namespace, and moves outwards until it reaches the module's global namespace

Namespace and scope

```
>>> name = 1
... def func():
...     name = func.__name__
...     print(locals())
...     print(globals()
...
... func()

{'name': 'func'}

{'__name__': '__main__', '__doc__': None, '__package__':
None, ..., '__builtins__': <module 'builtins' (built-in)>,
'name': 1, 'func': <function func at 0x7fbe556c79e0>}
```

LEGB Rule

- **L**ocal - contains the names that you define inside the function.

- **E**nclosing (nonlocal) scope is a special scope that only exists for nested functions.

- **G**lobal (or module) scope is the top-most scope in a Python program, script, or module.

- **B**uilt-in scope is a special Python scope that’s created or loaded whenever you run a script or open an interactive session.

LEGB Rule

Local - contains the names that you define inside the function.

These names will only be visible from the code of the function.

It's created at function call, *not* at function definition, so you'll have as many different local scopes as function calls.

This is true even if you call the same function multiple times, or recursively.

Each call will result in a new local scope being created.

LEGB Rule

Enclosing (nonlocal) scope is a special scope that only exists for nested functions.

This scope contains the names that you define in the enclosing function.

The names in the enclosing scope are visible from the code of the inner and enclosing functions.

LEGB Rule

Global scope is the top-most scope in a Python program, script, or module.

This Python scope contains all of the names that you define at the top level of a program or a module.

Names in this Python scope are visible from everywhere in your module’s code.

LEGB Rule

Built-in scope is a special Python scope that’s created or loaded whenever you run a script or open an interactive session.

This scope contains names such as keywords, functions, exceptions, and other attributes that are built into Python.

Names in this Python scope are available from everywhere in your code.

It’s automatically loaded by Python when you run a program or script.

Scopes

```
>>> name = "Global"                          # global
>>> def outer_function():
...     name = "Outer function scope"       # enclosing
...     def inner_fuction():
...         name = "Inner function scope" # local
...         print(name)
...     return inner_fuction

>>> func = outer_function()

>>> func()
Inner function scope
```

Closures

```
>>> name = "Global"
>>> def outer_function():
...     name = "Outer function scope"
...     def inner_fuction():
...         print(name)
...     return inner_fuction

>>> func = outer_function()

>>> func()
Outer function scope
```

Closures

```
>>> name = "Global"
>>> def outer_function():
...     def inner_fuction():
...         print(name)
...     return inner_fuction

>>> func = outer_function()

>>> func()
Global
```

Closures

```
>>> name = "Global"
>>> def outer_function():
...     def inner_fuction():
...         print(name)
...     return inner_fuction

>>> func = outer_function()

>>> func()
Global
```

global

```
>>> num = 1
...
...
... def func():
...     num = num + 1
...     return num
...
...
... print(func())
Traceback (most recent call last):
  File "<stdin>", line 9, in <module>
  File "<stdin>", line 5, in func
UnboundLocalError: local variable 'num' referenced before
assignment

local variable 'num' referenced before assignment
```

global

global allows to modify variable value from global scope

global

```
>>> num = 1
...
...
... def func():
...     global num
...     num = num + 1
...     return num
...
...
... print(func())
2
```

nonlocal

```
>>> def find_number(seq, num):
...     found = False
...
...     def helper():
...         indexes = []
...         for index, item in enumerate(seq):
...             if item == num:
...                 indexes.append(index)
...                 found = True
...         return indexes
...
...     indexes = helper()
...     return indexes, found

>>> find_number([1, 2, 3, 4, 4, 4, 4], 4)
([3, 4, 5, 6], False)
```

nonlocal

nonlocal allows to modify variable value from enclosing scope

nonlocal

```
>>> def find_number(seq, num):
...     found = False
...
...     def helper():
...         nonlocal found
...         indexes = []
...         for index, item in enumerate(seq):
...             if item == num:
...                 indexes.append(index)
...                 found = True
...         return indexes
...
...     indexes = helper()
...     return indexes, found

>>> find_number([1, 2, 3, 4, 4, 4, 4], 4)
([3, 4, 5, 6], True)
```

Namespaces and scopes

- There are four scopes in Python: Built-In, Global, Enclosing, Local
- LEGB Rule: name search is processed from Local to Built-In
- Assignment operations create name in current scope unless name defined as **global** or **nonlocal**