Inner_test

September 17, 2024

1 Python inner functions (aka nested functions)

1.1 Basic case

[ ]:

[1]:
```
def f(v1):
    v2 = 1
    def g(v3=2):
        return v1 + v2 + v3 + 4
    def h():
        return 16
    return g() + h() + 32
```

1.2 Function inside a method

[2]:
```
class C(object):
    def foo(self):
        def k(x):
            return [ self, x ]
        return k(3)
```

1.3 Deeper nesting

[3]:
```
def m():
    vm = 1
    def n(an=2):
        vn = 4
        def o(ao=8):
            vo = 16
            return vm + an + vn + ao + vo
        return o()
    return n()
```

2 How do we test inner fuctions?

2.0.1 After all, they are in the inner scope because we don't want anyone else to depend on them - they are implementation detail

[4]:
```
import pytest
from nested import nested
from tested_code import f, C, m

def test_all():
    # Basic case
    nested_g = nested(f, 'g', v1=8, v2=1)
    assert nested_g(2) == 15

    nested_h = nested(f, 'h')
    assert nested_h() == 16

    # Method case
    nested_k = nested(C.foo, 'k', self='mock')
    assert nested_k(5) == ['mock', 5]

    # Deeper nesting
    nested_n = nested(m, 'n', vm=1)
    nested_o = nested(nested_n, 'o', vm=1, an=2, vn=4)
    assert nested_o(8) == 31
```

3 The magic

3.0.1 nested.py

[5]:
```
'''Gain access to the code objects of inner functions for testing purposes.'''
import types

def free_var(val):
    '''A function that wraps free variables.'''
    def nested():
        return val

    # The __closure__ attribute of a closure function returns a tuple of cell␣
 ↪objects.
    # This cell object also has an attribute called cell_contents,
    # which returns the contents of the cell
    return nested.__closure__[0]

def nested(outer, inner_name, **free_vars):
    '''Find the code object of an inner function and return it as a callable␣
 ↪object.
```

```
    Arguments:
        outer (function or method): A function object with an inner function.
        inner_name (str): The name of the inner function we want access to
        **free_vars (dict(str: any)): A dictionary with values for the free
            variables in the context of the inner function.
    Returns:
        A function object for the inner function, with context variables set.
    '''

    # 1. Check that the passed object is actually a callable
    # 2. If it is, fetch the code object
    if isinstance(outer, (types.FunctionType, types.MethodType)):
        outer = outer.__code__

    # 3. Loop through the constants of the code object
    # -  mix of objects tied to the code obj
    for const in outer.co_consts:

        # 4. Check if each one is callable and the one we are looking for
        if isinstance(const, types.CodeType) and const.co_name == inner_name:

            # 5. Return a function using the code object we found with the␣
↪global
            #    environment set to the original globals and all the
            #    free variables bound to the vallues passed in the call
            return types.FunctionType(const, globals(), None, None, tuple(
                free_var(free_vars[name]) for name in const.co_freevars))
```

4 https://www.openend.se/~jacob/src/nested.py