pass_variables_to_functions

November 11, 2023

[1]:
```
def try_to_modify(x, y, z):
    x = 23
    y.append(42)
    z = [99] # new reference
    print(x)
    print(y)
    print(z)

a = 77    # immutable variable
b = [99]  # mutable variable
c = [28]
try_to_modify(a, b, c)
# Immutable types (e.g., integers, strings, tuples) are effectively pass by␣
 ↪value in Python because you can't
#                                                                    modify␣
 ↪them in-place within a function.


# Mutable types (e.g., lists, dictionaries) are effectively pass by reference␣
 ↪because changes made to them
#                                                        within a function will␣
 ↪affect the original object.

# Parameters to functions are references to objects/values, which are passed by␣
 ↪value. When you pass a variable to
#                 a function, python passes the reference to the object to␣
 ↪which the variable refers (the value).
#                 Not the variable itself.

# If the value passed in a function is immutable, the function does not modify␣
 ↪the caller's variable. If the
#                                  value is mutable, the function may modify␣
 ↪the caller's variable in-place.
```

```
23
[99, 42]
[99]
```

[2]:
```
print(a) # not changed
print(b)
print(c)
```

```
77
[99, 42]
[28]
```

[14]:
```
##########################################################################
x = 5

# "global" variables cannot be modified within the function, unless declared␣
 ↪global in the function.
def addx(y):
    return x + y
addx(10)
```

[14]: 15

[18]:
```
def setx(y):
    x = y
    print('x is %d' %x)
setx(10)
x
```

```
x is 10
```

[18]: 5

[19]:
```
def setx_(y):
    global x
    x = y
    print('x is %d' %x)
setx_(10)
x
```

```
x is 10
```

[19]: 10

[26]:
```
##########################################################################3

# function definition
def calculateTotalSum(*arguments):
    # type(arguments) => <class 'tuple'>
    # arguments => (5, 4, 3, 2, 1)
    totalSum = 0
    for number in arguments:
```

```
        totalSum += number
    print(totalSum)

# function call
calculateTotalSum(5, 4, 3, 2, 1)
```

```
15
```

[24]:
```
# function definition
def displayArgument(**arguments):
    # type(arguments) => <class 'dict'>
    # arguments => {'argument1': 'Geeks', 'argument2': 4, 'argument3': 'Geeks'}
    for arg in arguments.items():
        print(arg)

# function call
displayArgument(argument1 ="Geeks", argument2 = 4, argument3 ="Geeks")
```

```
('argument1', 'Geeks')
('argument2', 4)
('argument3', 'Geeks')
```

[34]:
```
##############################################################################
z=10
def setx_v(z):
    print(z)
    y = z*20
    print('x is %d' %y)
setx_v(15)
z
```

```
15
x is 300
```

[34]: 10

[33]:
```
z=16
def setx_v_(z):
    print(z)
    z = z*100
    print('x is %d' %z)
setx_v_(z)
z
```

```
16
x is 1600
```

[33]: 16

[35]:
```
################################################################################
var = 100  # A global variable
def increment():
    var = var + 1  # Try to update a global variable

increment()
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
Cell In[35], line 6
      3 def increment():
      4     var = var + 1  # Try to update a global variable
----> 6 increment()

Cell In[35], line 4, in increment()
      3 def increment():
----> 4     var = var + 1

UnboundLocalError: local variable 'var' referenced before assignment
```

[73]:
```
var = 100  # A global variable
def increment_():
    var = 0
    var = var + 1  # Try to update a global variable
    return var

increment_()
```

[73]:
```
1
```

[42]:
```
def func():
    var_ = 100  # a nonlocal variable - nonlocal to nested()             #␣
 ↪immutable
    def nested():
        #  nonlocal statement causes the listed identifiers to refer to␣
 ↪previously bound variables in the nearest
        #                                                                 ␣
 ↪enclosing scope excluding globals.
        nonlocal var_  # declare var as nonlocal
        var_ += 100

    nested()
    print(var_)

func()
```

```
200
```

[44]:
```
# Unlike global, you can't use nonlocal outside of a enclosed function.
nonlocal my_var
```

```
  Cell In[44], line 2
    nonlocal my_var
    ^
SyntaxError: nonlocal declaration not allowed at module level
```

[49]:
```
# Unlike global, you can't use nonlocal outside of a nested function.
def func_():
    nonlocal var  # Try to use nonlocal in a local scope
    print(var)
```

```
  Cell In[49], line 3
    nonlocal var  # Try to use nonlocal in a local scope
    ^
SyntaxError: no binding for nonlocal 'var' found
```

[50]:
```
def func_1():
    def nested():
        nonlocal lazy_var  # Try to create a nonlocal lazy name
```

```
  Cell In[50], line 3
    nonlocal lazy_var  # Try to create a nonlocal lazy name
    ^
SyntaxError: no binding for nonlocal 'lazy_var' found
```

[51]:
```
def func_2(arg):
    var = 100
    print(locals())
    another = 200

func_2(300)
```

```
{'arg': 300, 'var': 100}
```

[ ]:
```

```

[60]:
```
print( list(locals().items())[:5] )
locals() is globals()
```

```
[('__name__', '__main__'), ('__doc__', 'Automatically created module for IPython
interactive environment'), ('__package__', None), ('__loader__', None),
('__spec__', None)]
```

[60]:
```
True
```

[53]:
```
# locals() is only useful for read operations since updates to the locals␣
 ↪dictionary are ignored by Python.
def func_3():
    var = 100
    locals()['var'] = 200
    print(var)

func_3()
```

```
100
```

[61]:
```
list(globals().items())[:5]
```

[61]:
```
[('__name__', '__main__'),
 ('__doc__',
  'Automatically created module for IPython interactive environment'),
 ('__package__', None),
 ('__loader__', None),
 ('__spec__', None)]
```

[63]:
```
globals()['__doc__'] = """Docstring for __main__ ."""
__doc__
```

[63]:
```
'Docstring for __main__ .'
```

[66]:
```
def power_factory(exp):
    # returns closures (an inner function).
    def power(base):
        return base ** exp
    return power

square = power_factory(2)
print(square(10))
square(20)
```

```
100
```

[66]:
```
400
```

[65]:
```
power_factory(4)
```

[65]:
```
<function __main__.power_factory.<locals>.power(base)>
```

[67]:
```
cube = power_factory(3)
print(cube(3))
cube(4)
```

```
27
```

[67]: 64

[68]:
```
########################################################################
def mean():
    sample = []
    def _mean(number):
        sample.append(number)
        return sum(sample) / len(sample)
    return _mean

current_mean = mean()
print(current_mean(10))
print(current_mean(15))
print(current_mean(12))
print(current_mean(11))
current_mean(13)
```

```
10.0
12.5
12.333333333333334
12.0
```

[68]: 12.2

[72]:
```
#####################################################3
def mean_():
    total = 0
    length = 0
    def _mean(number):
        nonlocal total, length
        total += number
        length += 1
        return total / length
    return _mean

current_mean = mean_()
print(current_mean(10))
print(current_mean(15))
print(current_mean(12))
print(current_mean(11))
current_mean(13)
```

```
10.0
12.5
12.333333333333334
12.0
```

[72]: 12.2

[ ]: