|  |  |
| --- | --- |
| Lab 12 - Scope | Name: Lex Baker  Date: 11/8/22 |

A ***function call*** sends the flow of execution from one block of statements to another block of statements. So far we have seen the flow go from the main program to a function and back. In general, functions call other functions which call other functions, ..., so the flow of execution can get quite complicated. As you send your program from function to function it is critical to know which variables are available to be used in statements and expressions. The set of all locations where a **variable name** can be used without incurring an error is called the **scope** of that variable name.

**1.** Open the file exploring\_scope.py from Canvas, and run it to see what happens. After you have studied (and fiddled with) this program, answer the following questions.

1. Does changing the value of the parameter inside a function effect that value of the corresponding variable passed as an argument? How do you know?

**It affects the value outside the function only if the value passed is mutable, such as an array, and it is modified, not directly replaced. An immutable value such as a string is not, even if the value is changed within the function. Proven by:**

**def my\_test(a, b):**

**a = 5**

**b += 5**

**return**

**def my\_array\_test(a, b):**

**a = [5]**

**b += [5]**

**return**

**first = 1**

**second = 2**

**my\_test(first, second)**

**print(first, second) # Returns: 1 2**

**arr = [1, 2, 3, 4]**

**arr2 = [1, 2, 3, 4]**

**my\_array\_test(arr, arr2)**

**print(arr, arr2) # Returns: [1, 2, 3, 4] [1, 2, 3, 4, 5]**

1. Are variable names defined in the main program accessible in the function? How do you know?

**Yes, because they are variables with a global scope. For instance:**

**def more\_testing():**

**if is\_global:**

**print("Globally accessible variable")# This prints**

**print(is\_global)# Returns: True**

**is\_global = True**

**more\_testing()**

1. After completion of a function call, are names of parameters used in the function still accessible in the main program? How do you know?

**No, they are removed after the program closes, in addition to only being accessible within the function**

**def param\_exists(check\_var):**

**check\_var += 10**

**return**

**example = 5**

**param\_exists(example)**

**print(check\_var)# Returns: NameError: name ‘check\_var’ is not defined**

1. After completion of a function call, are names of variables defined inside the function still accessible in the main program? Make a conjecture and then add to the program to test it. Describe what you did and what you found

**No they are not. Tested by defining a variable within a function and attempting to call it outside of the function.**

**def param\_exists(check\_var):**

**check\_var += 10**

**a\_new\_var = 17**

**return**

**example = 5**

**param\_exists(example)**

**# print(check\_var)**

**print(a\_new\_var)# Returns: NameError: name ‘a\_new\_var’ is not defined**

.

1. Can the value of a parameter used in a function be altered by a statement inside of the function? How do you know?

**Yes:**

**def param\_mod(param):**

**print(param) # Returns 32**

**param += 10**

**print(param) # Returns 42**

**return**

**param\_mod(32)**

**f.** Can the value of a variable defined in the main program be altered by a statement inside of a function? Make a conjecture and then add to the program to test it. Describe what you did and what you found in the space below. Then add the line "global message\_in\_main" at the top of the function and try running the code again. What do you notice?

**Originally, no, the global variable can be accessed within the function but any attempt to modify it or set it to something new either throws an UnboundLocalError or creates a local variable by the same name, respectively. Adding “global message\_in\_main” allows the global variable to be modified and completely changed from within the function with no errors. However, it’s important to note that this affects the variable outside of the function as well.**

**2. Duplicating names.** Create a program with a function called shadowing to test the following questions:

1. What happens if the parameter value in the function has the same name as a variable within main?
2. Do you change the value in the main part of the program if you alter it within the function?
3. What if you create a variable in the function with the same name as a variable in main?
4. Which value takes precedence within the function and outside of it?

Write both your code and the answers to the questions below.

**def shadowing(same):**

**same2 = "world"**

**print("First from function: ", same, same2) # Returns: First from function: 21 world**

**same += 5**

**same2 = "helloW"**

**print("Second from function: ", same, same2) # Returns: Second from function: 26 helloW**

**same = 10**

**same2 = "hello"**

**print("First from main: ", same, same2) # Returns: First from main: 10 hello**

**shadowing(21)**

**print("Second from main: ", same, same2) # Returns: Second from main: 10 hello**

1. **Nothing, the function will use the value passed to the parameter.**
2. **No, it will only affect the value of the parameter.**
3. **The function will use the local variable, and the global variable will remain unchanged.**
4. **Within the function, the local variable takes precedence. Outside of the function, only the global variable is accessible, so that’s the one that takes precedence.**