

CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

Topics

- Introduction to Functions
- Defining and Calling a Function
- Local Variables
- Passing Arguments to Functions
- Global Variables
- Value Returning Functions
- Recap: PythonTutor

Introduction to Functions

Function: group of statements within a program that perform as specific task

- Usually one task of a large program
 - Functions can be executed in order to perform overall program task
- Avoids code repetition and makes the code reusable

Figure 3-1 Using functions to divide and conquer a large task

This program is one long, complex sequence of statements.

[illegible]

In this program the task has been divided into smaller tasks, each of which is performed by a separate function.

```
def function1():
    statement
    statement
    statement
```

```
def function2():
    statement
    statement
    statement
```

```
def function3():
    statement
    statement
    statement
```

```
def function4():
    statement
    statement
    statement
```

Benefits of Modularizing a Program with Functions

The benefits of using functions include:

- Simpler code
- Code reuse
 - write the code once and call it multiple times
- Better testing and debugging
 - Can test and debug each function individually
- Faster development
- Easier facilitation of teamwork
 - Different team members can write different functions

Defining and Calling a Function

- **Functions are given names**
 - Function naming rules:
 - ✓ Cannot use key words as a function name
 - ✓ Cannot contain spaces
 - ✓ First character must be a letter or underscore
 - ✓ All other characters must be a letter, number or underscore
 - ✓ Uppercase and lowercase characters are distinct

Defining and Calling a Function (cont'd.)

- Function names should be descriptive of the task carried out by the function

Function definition:

```
def function_name() :  
    statement  
    statement
```

Defining and Calling a Function (cont'd.)

- **Function header**: first line of function
 - Includes keyword `def` and function name, followed by parentheses and colon

```
def function_name() :
```

- **Block**: set of statements that belong together as a group
 - Example: the statements included in a function

```
def function_name() :
```



```
    statement  
    statement
```


Defining and Calling a Function (cont'd.)

- **Call a function to execute it**

- When a function is called:
 - Interpreter jumps to the function and executes statements in the block
 - Interpreter jumps back to part of program that called the function

Indentation in Python

Each block must be indented

- Lines in block must begin with the same number of spaces
 - Use tabs or spaces to indent lines in a block, but not both as this can confuse the Python interpreter
 - IDLE automatically indents the lines in a block

"Hello, World!" with Functions

```
#Name:  your name here  
#Date:  October 2017  
#This program, uses functions,  
#      says hello to the world!
```

```
def main():  
    print("Hello, World!")
```

```
if __name__ == "__main__":  
    main()
```

Python Tutor

```
#Name: your name here  
#Date: October 2017  
#This program, uses functions,  
#    says hello to the world!
```

```
def main():  
    print("Hello, World!")  
  
if __name__ == "__main__":  
    main()
```

(Demo with pythonTutor)

Local Variables

Local variable: variable that is assigned a value inside a function

- Belongs to the function in which it was created
 - Only statements inside that function can access it, error will occur if another function tries to access the variable
 - variables that are defined inside of a function are considered “local” to that function
 - This means that they only exist within that function. Objects outside the “scope” of the function will not be able to access that variable

Scope: the part of a program in which a variable may be accessed

- For local variable: function in which created

Local Variables (cont'd.)

- **Different functions may have local variables with the same name**
 - Each function does not see the other function's local variables, so no confusion
 - local variables will not overwrite one another since they exist in different "scopes"

Python 3.6

([known limitations](#))

```
1 def func1():  
2     value = 5  
3  
4  
→ 5 print(value)
```



NameError: name 'value' is not defined


Passing Arguments to Functions

Argument: piece of data that is sent into a function

- Function can use argument in calculations
- When calling the function, the argument is placed in parentheses following the function name
- This process is identical to what you've been doing with the built-in functions we have studied so far (i.e len(x))

Figure 3-13 The value variable is passed as an argument

```
def main():  
    value = 5  
    show_double(value)  
  
def show_double(number):  
    result = number * 2  
    print(result)
```

A diagram consisting of a horizontal line extending from the right side of the 'show_double(value)' call in the first function, followed by a vertical line pointing downwards to the 'show_double(number)' function definition, ending in an arrowhead. This illustrates the flow of the argument 'value' to the parameter 'number'.

Passing Arguments to Functions (cont'd.)

Parameter variable: variable that is assigned the value of an argument when the function is called

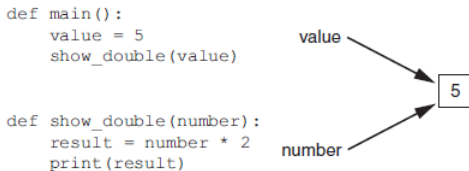
- The parameter and the argument reference the same value
- General format:

```
def function_name(parameter) :
```

- Scope of a parameter: the function in which the parameter is used

Passing Arguments to Functions (cont'd.)

Figure 3-14 The `value` variable and the `number` parameter reference the same value

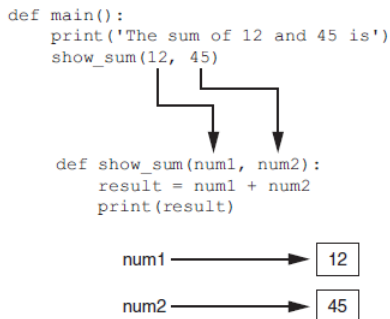


Passing Multiple Arguments

- **Python allows writing a function that accepts multiple arguments**
 - Parameter list replaces single parameter
 - Parameter list items separated by comma
- **Arguments are passed *by position* to corresponding parameters**
 - First parameter receives value of first argument, second parameter receives value of second argument, etc.

Passing Multiple Arguments (cont'd.)

Figure 3-16 Two arguments passed to two parameters

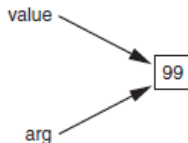


Changes made to a parameter value within the function do not affect the argument*

Making Changes to Parameters (cont'd.)

Figure 3-17 The value variable is passed to the change_me function

```
def main():  
    value = 99  
    print('The value is', value)  
    change_me(value)  
    print('Back in main the value is', value)  
  
def change_me(arg):  
    print('I am changing the value.')  
    arg = 0  
    print('Now the value is', arg)
```



Making Changes to Parameters (cont'd.)

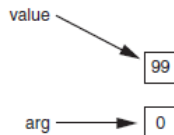
Figure 3-18

The `value` variable passed to the `change_me` function cannot be changed by it

Figure 3-18 The `value` variable is passed to the `change_me` function

```
def main():  
    value = 99  
    print('The value is', value)  
    change_me(value)  
    print('Back in main the value is', value)
```

```
def change_me(arg):  
    print('I am changing the value.')  
    arg = 0  
    print('Now the value is', arg)
```



Keyword Arguments

- **Keyword argument: argument that specifies which parameter the value should be passed to**
 - Position when calling function is irrelevant
 - General Format:

`function_name(parameter=value)`

- **Possible to mix keyword and positional arguments when calling a function**
 - Positional arguments must appear first

Global Variables and Global Constants

Global variable: created by assignment statement written outside all the functions

- Can be accessed by any statement in the program file, including from within a function
- If a function needs to assign a value to the global variable, the global variable must be redeclared within the function
- If you want to be able to change a global variable inside of a function you must first tell Python that you wish to do this using the “global” keyword inside your function
 - General format: `global variable_name`

Value Returning Functions

- Value returning functions are functions that return a value to the part of the program that initiated the function call
- They are almost identical to the type of functions we have been writing so far, but they have the added ability to send back information at the end of the function call
- The only difference is that you need to include a “return” statement in your function to tell Python that you intend to return a value to the calling program
- The return statement causes a function to end immediately
- A function will not proceed past its return statement once encountered.

```
i.e Somestring =input("Tell me your name")
```


Value Returning Functions

```
def sampleFunction(arg1, arg2):  
    statement  
    statement  
  
    ...  
    statement  
    return expression  
  
returnValue = sampleFunction(1, 5)
```

In Pairs or Triples:

1. Predict what the code will do:

```
def totalWithTax(food,tip):  
    total = 0  
    tax = 0.0875  
    total = food + food * tax  
    total = total + tip  
    return(total)  
  
lunch = float(input('Enter lunch total: '))  
lTip = float(input('Enter lunch tip: ' ))  
lTotal = totalWithTax(lunch, lTip)  
print('Lunch total is', lTotal)  
  
dinner= float(input('Enter dinner total: '))  
dTip = float(input('Enter dinner tip: ' ))  
dTotal = totalWithTax(dinner, dTip)  
print('Dinner total is', dTotal)
```

Python Tutor

```
def totalWithTax(food,tip):  
    total = 0  
    tax = 0.0875  
    total = food + food * tax  
    total = total + tip  
    return(total)  
  
lunch = float(input('Enter lunch total: '))  
lTip = float(input('Enter lunch tip: '))  
lTotal = totalWithTax(lunch, lTip)  
print('Lunch total is', lTotal)  
  
dinner= float(input('Enter dinner total: '))  
dTip = float(input('Enter dinner tip: '))  
dTotal = totalWithTax(dinner, dTip)  
print('Dinner total is', dTotal)
```

(Demo with pythonTutor)

In Pairs or Triples:

Predict what the code will do:

```
motto = "Mihi Cura Futuri"  
l = len(motto)  
for i in range(l):  
    print(motto[i])  
for j in range(l-1,-1,-1):  
    print(motto[j])
```

Python Tutor

```
motto = "Mihi Cura Futuri"  
l = len(motto)  
for i in range(l):  
    print(motto[i])  
for j in range(l-1,-1,-1):  
    print(motto[j])
```

(Demo with pythonTutor)

Recap: Functions

```
#Name:  your name here
#Date:  October 2017
#This program, uses functions,
#      says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:
Example: `print("Hello", "World")`
- Can write, or **define** your own functions, which are stored, until invoked or called.