Intro to Computer Science CS-UH 1001, Spring 2022

Lecture 9 – Functions

Functions

- Functions are used to structure your code, improve code readability and make code reusable

- We have used functions already:
  - Built-in functions: like print(), len(), type(), etc
  - Module functions like random.randint(num1, num2), time.sleep(sec), etc
  - Methods are also functions

- We have seen that a all functions
  - have a name
  - have a set of arguments defined inside “( )”
  - can return a value (if not, None is returned)

Functions Types

- Two types of functions exist:
  - Void functions
  - Fruitful functions

- Void functions do not return a value
- Fruitful functions return a value

Functions Definition

- Syntax:

```
def function_name(arguments):
    # indented code block
    return value # return statement is optional
```

- Arguments:
  - A function can have as many arguments as desired, separated by commas
- The indented code block is executed when the function is called
- The function exits if the end of the indented code block is reached
  - or if a return statement is reached, if present

Function Definition

```
def print_hello_world():
    print("Hello", end=" ")
    print("World!")
```

← Function without arguments

Function body

Function Definition

```
def measure_distance(x1, y1, x2, y2):
    '''This function is used to measure the
    distance between two points (x1,y1) and
    (x2,y2)
    Example: measureDistance(0,0,5,5)

    Precondition: all input arguments must
    be either an int or a float'''

    distance = ((x2-x1)**2+(y2-y1)**2)**0.5

    return distance
```

function with arguments

Specification (docstring)

Function body

Optional return statement

Functions Definition

- Function definition specifies what a function does:
  - It **does not** cause the function to execute

- If you want the function to execute you need to **call** the function:
  - You can call the function by writing the function name followed by "**( )**"

Functions Flow

- When you write a program with functions, always add the function definition to the beginning of your program:
  - You cannot use functions before they are defined
- When you call a function, the Python interpreter
  - jumps to the function definition
  - executes the function body
  - jumps back to the line that called the function

Functions Flow

```
def main():
    print("This is the main function")
    print("Finished the main function")

def message():
    print ("This is a message")

main()
message()
print("Good bye")
```

1

2

3

4

Functions Flow

```
def main():
    print("This is the main function")
    message()
    print("Finished the main function")

def message():
    print("This is a message")

main()
print("Good bye")
```

1

2

3

4

Function Arguments

- Usually you want to send data into your function
  - So that the function can use them to do something
  - For example, if you want a function to check if a number is odd or even, you need to pass the number to the function
- Arguments are used to pass data to functions
  - You can define how many arguments the function takes
  - You also define the order of the arguments

Function Argument

```
def is_num_even_or_odd(number):
    if number%2 == 0:
        print("Number is even")
    else:
        print("Number is odd")

is_num_even_or_odd(3)
is_num_even_or_odd(40)
```

Number is odd
Number is even

Functions and Multiple Arguments

```
def power(number, exponent):
    print(number**exponent)

power(2, 4)  16
power(exponent=4, number=2)
```

Notice here the order of the arguments have changed by using explicitly the keyword argument

Hands-on Session

Using functions

Temperature Converter (ex_9.1.py)

Write a python function that takes a temperature value argument and converts it to the other scale. Then it prints the converted temperature value. The conversion should be done in a function.

Example conversions:

40 C => 104 F

or

104 F => 40 C

The function should have 2 arguments:

convert_temp(value, unit)

Example:

convert_temp(104, ‘F’)

Hint: use F = (C*9/5)+32

Default arguments

- Python allows function arguments to have default values
  - If the function is called without the argument, the argument gets its default value

- Example:

```
def power(number, exponent = 2):
    print(number**exponent)

power(2)   4
power(2, 4)  16
```

Temperature Converter (ex_9.1.py)

Modify the previous exercise so that the default value is Celsius

Example:

convert_temp(40) => 104 F

Note:

convert_temp(104, ‘F’) should still work!

Functions and Local Variables

- If an immutable variable is assigned a value anywhere within the function's body, it is a local function variable
- A local function variable cannot be accessed outside the function they are defined in
- Consequently, you can have the same variable name in different functions

Local Variable Examples

```
def calc_square(value):
    number = value**2

calc_square(5)
print(number)
```

NameError: name 'number' is not defined

```
def calc_square(value):
    number = value**2

number = 5
calc_square(number)
print(number)
```

5

Variables Outside Functions

- Immutable variables that are defined outside functions
  - can be accessed inside a function
  - can not be changed inside a function

- If you use an assignment statement inside the function with the same variable name you are creating a local variable

Variables Outside Functions

```
def calc_square():
    print(number**2)   25

number = 5
calc_square()
print(number)   5
```

```
def calc_square():
    number = 10
    print(number**2)   100

number = 5
calc_square()
print(number)   5
```

Global Variables

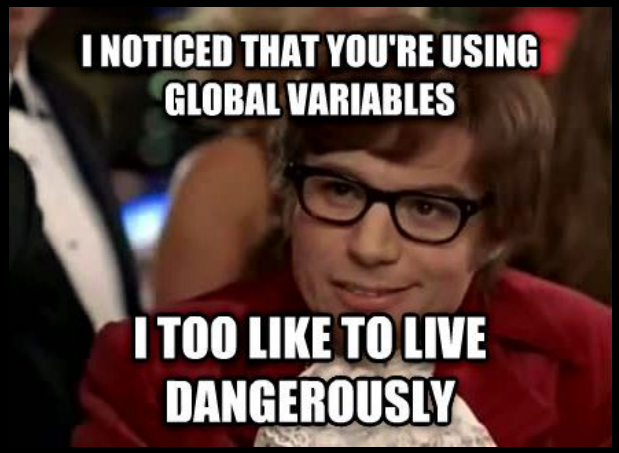
- The global keyword allows to modify a variable outside of the current scope
- It is used to create/modify a global variable from a non-global scope (inside a function)
- Note: There is no need to use the global keyword outside functions!

Global Variables Example

```
def calc_square():
    global number
    number = 10
    print(number**2)    100

number = 5
calc_square()
print(number)       10
```

Just a Minute: Global Variables are Evil! ☺



- As few as possible, as many as necessary
- Lots of global variables lead to ”Spaghetti” code
- Remember:
  global variables != global constant variables

Fruitful Functions

- There is a more elegant way:
  - Fruitful functions
- Fruitful functions are functions that return a value back to the caller using the return keyword
- Note: Functions can have multiple return statements, but the function exits after one is reached!

Local Variable vs. Fruitful Function

```
def calc_square(value):
    number = value**2
    return number

number = 5
result = calc_square(number)
print(result)        25
print(number)        5
```

Local Variable vs. Fruitful Function

```
def calc_square(value):
    number = value**2
    return number

number = 5
number = calc_square(number)
print(number)
```

25

Multiple Return Values

- What if the function should return multiple variables?

- Option 1:
  - return a, b, c
  - x, y, z = function()

- Option 2:
  - return [a, b, c]
  - values = function()

Example: Multiple Return Values

```
import random

def three_random_numbers():
    a = random.randint(0,10)
    b = random.randint(0,10)
    c = random.randint(0,10)
    return a,  b,  c

num1, num2, num3 = three_random_numbers()
print(num1)
print(num2)
print(num3)
```

Example: Multiple Return Values

```
import random

def three_random_numbers():
    a = random.randint(0,10)
    b = random.randint(0,10)
    c = random.randint(0,10)
    return [a,  b,  c]

numbers = three_random_numbers()
print(numbers[0])
print(numbers[1])
print(numbers[2])
```

Hands-on Session

Using functions II

Check for Even Number (ex_9.2.py)

Write a function that checks whether a number is even.

The function should return True if the number is even.

For example:

```
result = check_even(3)
print(result)  => False
# or
print(check_even(20))  => True
```

Reverse String (ex_9.3.py)

Write a function to reverse a string.

For example:

```
my_string = “abcdef”
my_string = reverse_string(my_string)
print(my_string)  => fedcba
```

Unique List (ex_9.4.py)

Write a function that takes a list as an argument and returns a new list with unique elements of the list.

For example:

```
my_list = [1,2,2,3,3,3,4,4,4,4]
print(unique_list(my_list))  => [1,2,3,4]
```