Computational Thinking with Data Science

5. Programming Principles Functions & Variable scope

Turgut Batuhan Baturalp, (Dr. Batu)
Whitacre College of Engineering
Texas Tech University

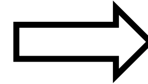
Topic Outline

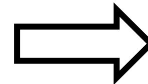
- Functions in Python
  - Creating a Function
  - Calling a Function
- Variable Scopes
  - Types of Scopes
- Recursive Functions

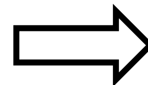
Objectives

- To understand the role, types and usage of Functions.
- To understand and implement Functions in Python under various configurations.
- To understand the role and types of variable scopes.
- To understand and implement variable scopes in Python under various configurations.

Computational Thinking Concepts

Functions 
- Decomposition
- Algorithm design

Variable Scopes 
- Decomposition
- Algorithm design

Recursive Functions 
- Pattern recognition

What is a Function in Coding?

Function: A piece of code that you can easily use multiple times by calling it in the code. Basically, functions are packages of codes that executes certain tasks.

https://youtu.be/0eo0ESEX9DE

How to use Functions?

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.
- They are useful for decomposition, since Functions can be used to modularize a program.

- Things you can do using functions:
  - Creating a function
  - Calling a function (I bet you did this before. For example, if you searched anything on Google, you called the search function; or when you press the plus button on your calculator, you are calling the sum function.)

Creating a function

- In Python a function is defined using the def keyword:
- Example

Begin definition with def

function name

```
def my_function_name1():
    <body statements>

<outside function statements>

def my_function_name2(arg1, arg2):
    <body statements>
    return <expression>
```

Arguments
Initialized corresponding to the values that were passed when calling the function

Return to indicate the values to be sent back to the caller

Calling a function

- To call a function, use the function name followed by parenthesis:
- Example

```
my_function_name1()
```

Function - Return

- All functions in Python have a return value, including the functions does not have the return statement.
- Functions without the return statement will return None.
  - None is a special constant.
  - None is equivalent to False.

Function Examples – 1

- Function to calculate sum of the first n numbers?

```
def sum_of_n_numbers(n) :
    my_sum = 0
    for i in range(n+1):
        my_sum = my_sum + I
    return my_sum
```

- Calling the defined function:

```
my_sum = sum_of_n_numbers(10)
print(my_sum)
```

What value will n be initialized?

```
print(sum_of_n_numbers(10))
```

What value will n be initialized?

```
m = 10
my_sum = sum_of_n_numbers(m)
print(my_sum)
```

What value will n be initialized?

Function Examples – 2

- Function to calculate sum of the first n numbers?

```
def sum_of_n_numbers(n) :
    my_sum = 0
    for i in range(n+1):
        my_sum = my_sum + I
```

NO return statement

- Calling the defined function:

```
my_sum = sum_of_n_numbers(10)
print(my_sum)
```

What value will be printed?

```
print(sum_of_n_numbers(10))
```

What value will be printed?

```
m = 10
my_sum = sum_of_n_numbers(m)
print(my_sum)
```

What value will be printed?

Function Arguments w/ Default Values

```
def sum_of_n_numbers(n=3) :
    my_sum = 0
    for i in range(n+1):
        my_sum = my_sum + I
    return my_sum


my_sum = sum_of_n_numbers()

print(my_sum)
```

What value will be printed?

- Does NOT allow argument with default value followed by arguments without default values.

bad

```
def my_function(a, n=3, b) :
    <statements>
```

Variable Scope & Types of Scopes

- A variable is only available from inside the region (a function for example) it is created. This is called scope.
- Types of Scopes:
  - Local Scope: A variable created inside a function belongs to the local scope of that function and can only be used inside that function.
  - Function Inside Function: In this case, the variable is not available outside the function, but it is available for any function inside the function (nested functions).
  - Global Scope: A variable created in the main body of the code is a global variable and belongs to the global scope. Global variables are available from within any scope, global and local.

Rules & tips about variable scopes

- If you operate with the same variable name inside and outside of a function, they will be treated as two separate variables, one available in the global scope (outside the function) and one available in the local scope (inside the function).
- If you need to create a global variable, but are stuck in the local scope, you can use the global keyword. The global keyword makes the variable global.

Function Arguments & Variable Scope

Class	Description	Immutable
bool	Boolean value	Yes
int	Integer	Yes
float	Floating-point number	Yes
str	Character string	Yes
tuple	Immutable sequence of objects	Yes
list	Mutable sequence of objects	No
set	Unordered set of distinct objects	No
dict	Associative mapping	No

- Immutable objects: can not change after it is created. Passed to function by value.
- Mutable objects: can change after it is created. Passed to function by reference.

Function Scope Example

- Possible to have function definition inside another function definition.

```
def my_function_name1():
    <body statements>
    def my_function_name2(arg1, arg2):
        <inner function body statements>
    return <expression>
    my_function_name2(1, 2)
    <other body statements>
<outside function statements>
my_function_name2(1, 2)
```

Global function

Local function

Bad function call ?

Function in Built-in Modules

```
import math

print(math.sqrt(100))
```

Use import to use functions defined in that module.

Invoke function: Module name + dot + function name

```
import math as m

print(m.sqrt(100))
```

Use import/as to use functions defined in that module.

Invoke function: alias name + dot + function name

- How to know how to use a function?
- https://docs.python.org/2/library/math.html

Function in Built-in Modules

```
from math import sqrt, sin


print(sqrt(100))
print(sin(3.14))
```

Use from/import to use functions needed.

Invoke function: function name

- How to know how to use a function?
- https://docs.python.org/2/library/math.html

Live Demo: Variable Scopes in Function

```
1   x = 1
2   def a():
3       x = 25
4       print("\nlocal x in a is", x, "after entering a")
5       x += 1
6       print ("local x in a is", x, "before exiting a")
7
8   def b():
9       global x
10      print("\nglobal x is", x, "on entering b")
11      x *= 10
12      print("global x is", x, "on exiting b")
13
14  print("global x is", x)
15  x = 7
16  print ("global x is", x)
17  a()
18  b()
19  a()
20  b()
```

local

global

What is the output?

- Live demo 2: variable scope and pass by reference

Recursive Function

- Recursive Function: Function that invokes itself.

```
def my_func():
    <statements>
    my_func()
    <other statements>
```

Can I invoke the function in its body?

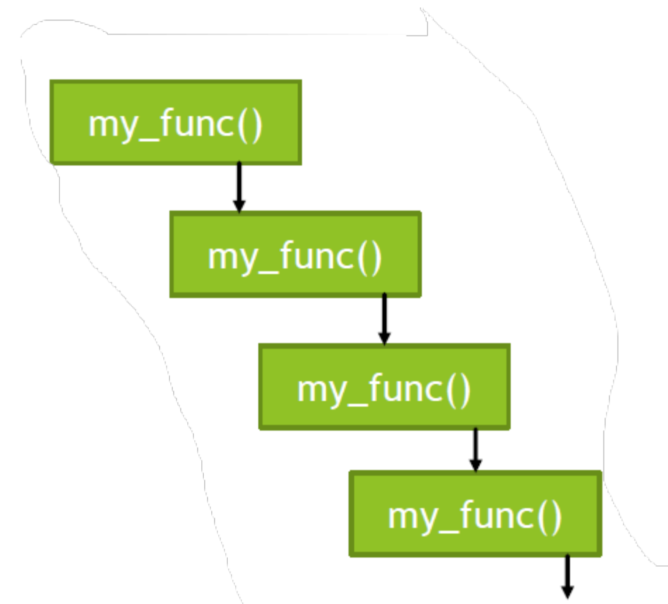
Does it work?

```
def my_func():
    print("hello")
    my_func()
my_func()
```

Does it work without any error?

Recursive Function - 2

```
def my_func():
    print("hello")
    my_func()
my_func()
```



Infinite recursive calls

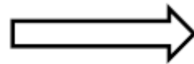
- Need termination case: called base case
- Invocation of the function: call recursion step

Recursive Function Example

- For a given value of n, calculate n!
  - n = 4
  - n! = 4x3x2x1

n! = n*(n-1)!

Recursive?

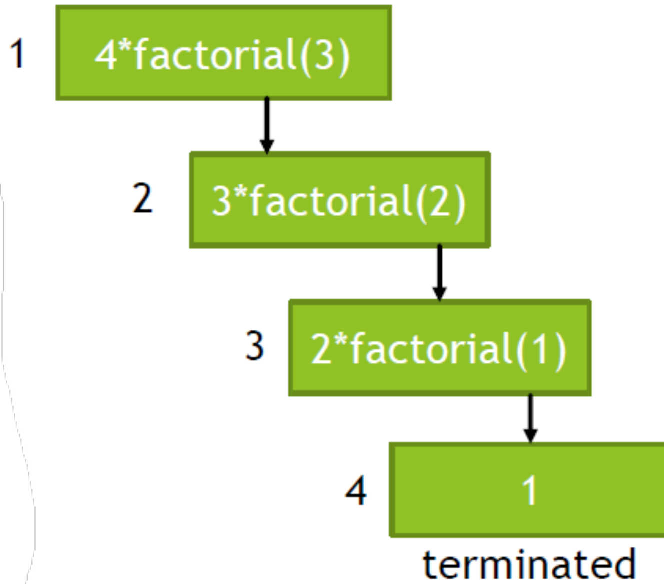


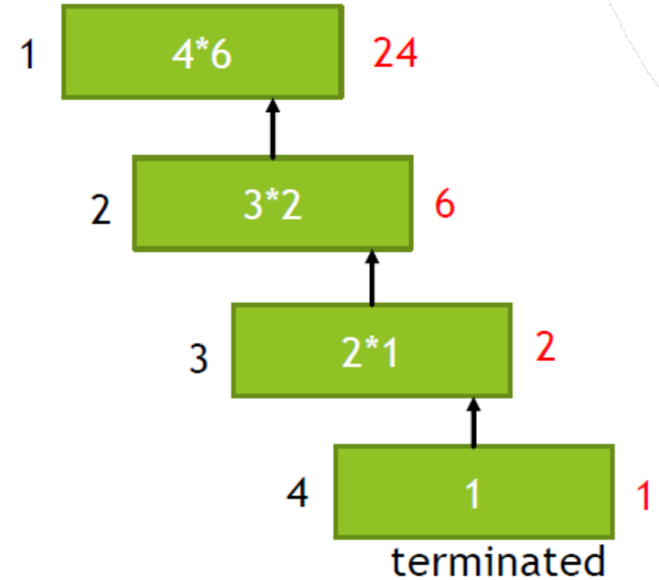
What is the base case?

```
def factorial(n):
    <<base case>>
    return n*factorial(n-1)

print(factorial(4))
```

Recursive Function Example






```
def factorial(n):
    if n <= 1:
        return 1
    return n*factorial(n-1)
```

Live demo: recursive function

Built-in String Functions - 1

- There are very useful built-in functions of Python for String applications.
  - The following method generates an uppercased version of a string.
  - The following method generates an lowercased version of a string.

Built-in String Functions - 2

- One of the most important method is replace, which replaces all instances of a substring within the string. The replace method takes two arguments, the text to be replaced and its replacement.

```
'hitchhiker'.replace('hi', 'ma')
```

```
'matchmaker'
```

- String methods can also be invoked using variable names, as long as those names are bound to strings.

```
s = "train"
t = s.replace('t', 'ing')
u = t.replace('in', 'de')
u
```

```
'degrade'
```

Built-in String Functions - 3

- Note that the line t = s.replace('t', 'ing') doesn't change the string s, which is still "train". The method call s.replace('t', 'ing') just has a value, which is the string "ingrain".

```
s
```

```
'train'
```

- The replace function is not unique to strings, can be applicable to other types of objects.