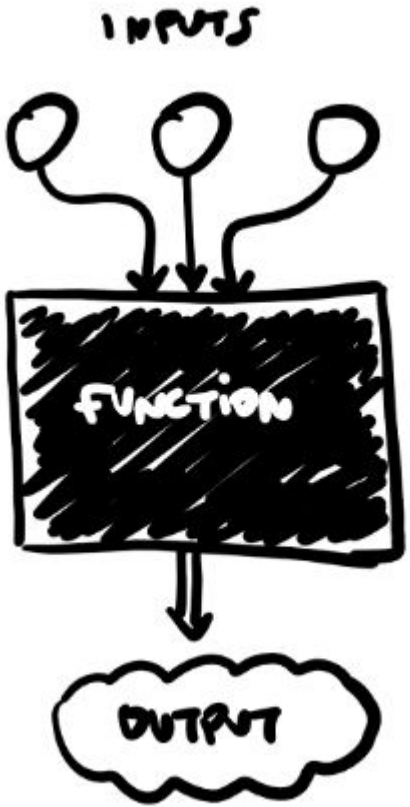
Programming Fundamentals

With Python

Chapter 2

01

Functions



Functions Overview

- A function takes n number of inputs, computes some logic with them and outputs 1 or more values
- Print() is actually a function that takes a string input and outputs that string on the terminal
- In Python, a function can return multiple values (but it is not recommended)

Basic Function Example

```
def calculate_rectangular_area(length,width):
    return length * width

area = calculate_rectangular_area(10,5)
print(f"The area of this rectangle is {area}")
```

Parameters

Arguments

Make sure the function is defined before it has been called

This is equivalent to saying:

length = 10
width = 5

This is called a function call. When it calls the function defined, it returns a value.

Void Functions

```python
def sayHello():
    print("Hello")

sayHello()
```

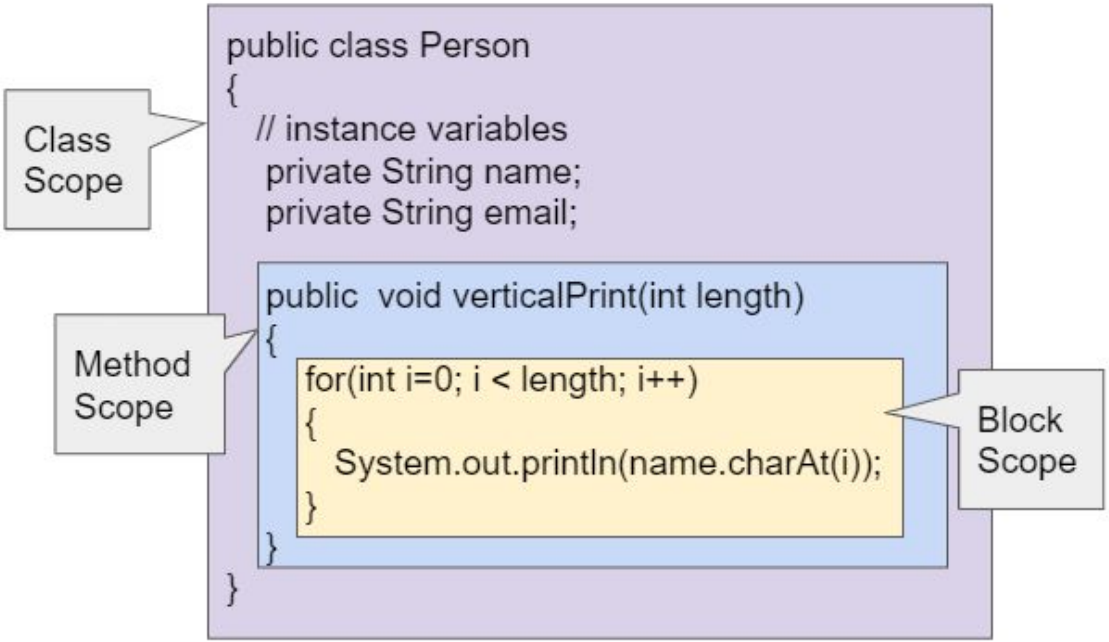
There are 2 main types:

1. A function that returns nothing, so you never use the keyword return
2. A function that has no parameters.

The example to the left shows a case where it is both. But, it can be any combination!

02

Variable Scoping



Local Variables

For now, we can define local Variables as variables that were instantiated within a function. So, you can only access them within that function.

```
def add():
    a = 5
    b = 10
    print(a+b)
add()
print(a+b)
```

In this example, I instantiated **a** and **b** in the add function. So, I can only print or change **a** and **b** within the add function.If you try printing outside the function, it will give an error saying the variables were not defined.

Global Variables

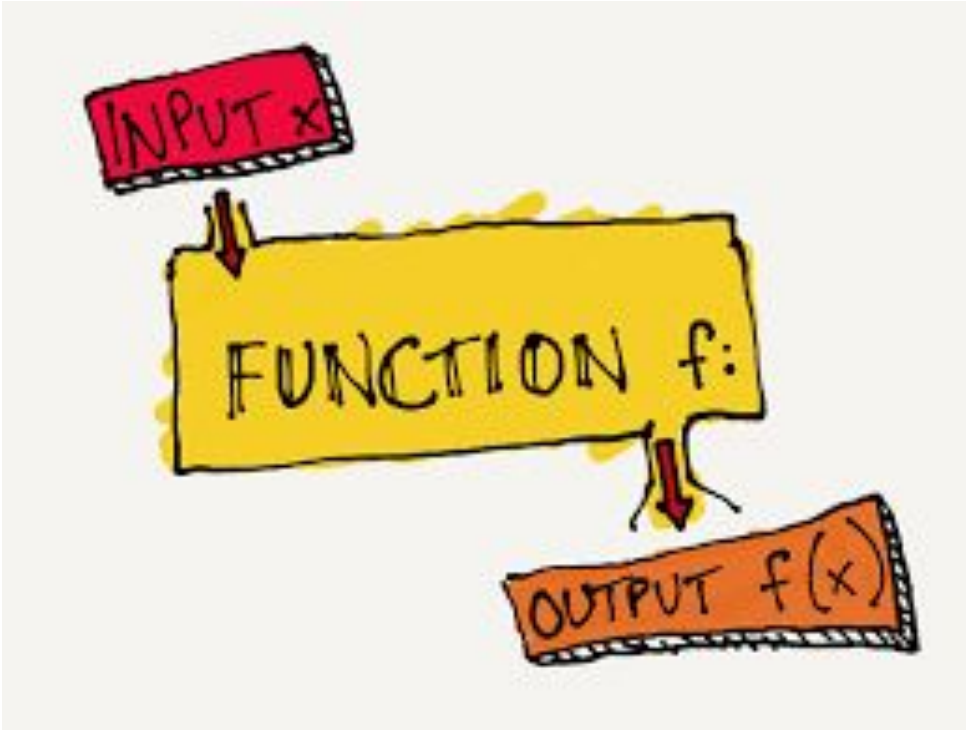
For now, we can define global Variables as variables that were instantiated outside. So, you can access them anywhere, even in any function.

```
1  a = 5
2  b = 10
3
4  def add():
5      print(a+b)
6
7  add()
8  print(a+b)
```

In this case, both line 5 and line 8 will print

03

Functions In Depth

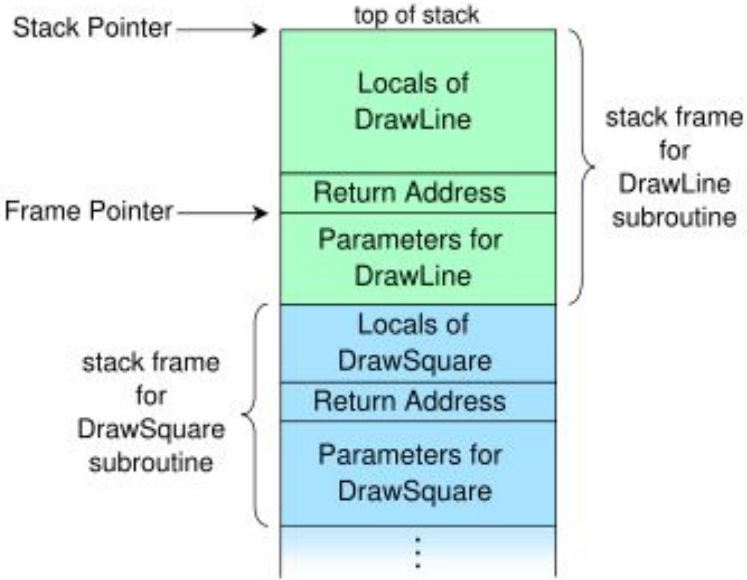


How computer manages functions

Think of it like stacking dishes. But, we are stacking memory in a computer.

When a function call is made, stack pointer goes down and compiler pushes one chunk of memory into the stack containing:

1. Local Variables
2. Return address
3. Arguments for the function



When a function is over or the return keyword is read, the function call is pulled from the stack and compiler goes to the most recent return address.

```
1  def addC(a,b):
2      c = 3
3
4      return a + b + c
5
6  def addB(a):
7      b = 2
8
9      return 1 + addC(a,b)
10
11 a = 1
12 print(addB(a))
13 print("Program Over")
```

Status: line 11

Abstract view of Stack

Stack Pointer

```
1  def addC(a,b):
2      c = 3
3
4      return a + b + c
5
6  def addB(a):
7      b = 2
8
9      return 1 + addC(a,b)
10
11 a = 1
12 print(addB(a))
13 print("Program Over")
```

Status: line 12

Abstract view of Stack

Local variables: a = 1 (This is also a global variable) Return Address: line 12 Arguments: a = 1
Stack Pointer

```
1  def addC(a,b):
2      c = 3
3
4      return a + b + c
5
6  def addB(a):
7      b = 2
8
9      return 1 + addC(a,b)
10
11 a = 1
12 print(addB(a))
13 print("Program Over")
```

Status: line 6

Abstract view of Stack

Local variables: a = 1 (This is also a global variable) Return Address: line 12 Arguments: a = 1
Stack Pointer

```
1  def addC(a,b):
2      c = 3
3
4      return a + b + c
5
6  def addB(a):
7      b = 2
8
9      return 1 + addC(a,b)
10
11 a = 1
12 print(addB(a))
13 print("Program Over")
```

Status: line 9

Abstract view of Stack

Local variables: a = 1 (This is also a global variable) Return Address: line 12 Arguments: a = 1
Local variables: b = 2 Return Address: line 9 Arguments: a = 1, b = 2
Stack Pointer

```
1  def addC(a,b):
2      c = 3
3
4      return a + b + c
5
6  def addB(a):
7      b = 2
8
9      return 1 + addC(a,b)
10
11 a = 1
12 print(addB(a))
13 print("Program Over")
```

Status: line 1

Abstract view of Stack

Local variables: a = 1 (This is also a global variable) Return Address: line 12 Arguments: a = 1
Local variables: b = 2 Return Address: line 9 Arguments: a = 1, b = 2
Stack Pointer

```
1  def addC(a,b):
2      c = 3
3
4      return a + b + c
5
6  def addB(a):
7      b = 2
8
9      return 1 + addC(a,b)
10
11 a = 1
12 print(addB(a))
13 print("Program Over")
```

Status: line 4 returns 6 (no more function calls so stack pointer starts going back up)

Abstract view of Stack

Local variables: a = 1 (This is also a global variable) Return Address: line 12 Arguments: a = 1
Local variables: b = 2 Return Address: line 9 Arguments: a = 1, b = 2
Stack Pointer

```
1  def addC(a,b):
2      c = 3
3
4      return a + b + c
5
6  def addB(a):
7      b = 2
8
9      return 1 + addC(a,b)
10
11 a = 1
12 print(addB(a))
13 print("Program Over")
```

Status: line 9 returns 7

Abstract view of Stack

Local variables: a = 1 (This is also a global variable) Return Address: line 12 Arguments: a = 1
Local variables: b = 2 Return Address: line 9 Arguments: a = 1, b = 2 **Stack Pointer**

```
1  def addC(a,b):
2      c = 3
3
4      return a + b + c
5
6  def addB(a):
7      b = 2
8
9      return 1 + addC(a,b)
10
11 a = 1
12 print(addB(a))
13 print("Program Over")
```

Status: line 12 prints 7

Abstract view of Stack

Local variables: a = 1 (This is also a global variable)
Return Address: line 12
Arguments: a = 1

Stack Pointer

```
1  def addC(a,b):
2      c = 3
3
4      return a + b + c
5
6  def addB(a):
7      b = 2
8
9      return 1 + addC(a,b)
10
11 a = 1
12 print(addB(a))
13 print("Program Over")
```

Status: line 13 prints “Program Over”

Abstract view of Stack

Stack Pointer

What is the point of a function?

- Separate the logic in the program into smaller pieces to make the code more readable and maintainable. In other words, functions help you to be able to follow the code better, make changes to it more easily, and to debug faster.
- Allows you to hide the complex details of performing some action. This allows people to create a library where you can just call a method to perform something complex you do not want to do yourself.
- If you have to perform the same logic multiple times, it is easier to make a function so that you can reuse that code.

Exercise 1

```python
def addC(a,b):
    c = 3
    return a * b + c
    print("Hey")

def addB(a):
    b = 2 - a
    print(addC(a,b) - 1)
    print("Hello")

a = 1
addB(a)
print("Hi")
```

Output

```
3
Hello
Hi
```

Exercise 2

1. Ask user for first number
2. Ask user for second number
3. Call an add function that takes the 2 numbers and returns the addition and print the result
4. Call a sub function that takes the 2 numbers and returns the subtraction and print the result
5. Call a multiply function that takes the 2 numbers and returns the multiplication and print the result

Output

```
Enter value of a: 5
Enter value of b: 1
The result of addition is 6
The result of subtraction is 4
The result of multiplication is 5
```

Exercise 2

```python
def add(a, b):
    return a + b

def sub(a, b):
    return a - b

def multiply(a,b):
    return a*b

a = int(input("Enter value of a: "))
b = int(input("Enter value of b: "))

print(f"The result of addition is {add(a,b)}")
print(f"The result of subtraction is {sub(a,b)}")
print(f"The result of multiplication is {multiply(a,b)}")
```