Functions + Scope

- **What is a function in Python?**
- **Types of Functions**
- **Defining a Function:**
  - Defining a function without any parameters
  - Defining a function without parameters and return value
  - Defining a function with parameters
  - Defining a function with parameters and return value
- **Passing Arguments with Key and Value**
- **How to call a function in python?**
- **Docstrings**
- **Global, Local Variables**

Functions

Why functions?

- Maximize code reuse and minimize redundancy
- Procedural decomposition
  - Break bit task into smaller tasks
  - Makes code less buggy – can test each function individually

Types of Functions

Python support two types of functions

1. **Built-in** function
2. **User-defined** function

1.**Built-in function**

The functions which are come along with Python itself are called a built-in function or predefined function. Some of them are: `range()`, `print()`, `input()`, `type()`, `id()`, `eval()` etc.

Example: Python `range()` function generates the immutable sequence of numbers starting from the given start integer to the stop integer.

```
>>> for i in range(1, 10):
>>>     print(i, end=' ')
```

1 2 3 4 5 6 7 8 9

2. User-defined function

Functions which are created by programmer explicitly according to the requirement are called a user-defined function.

Syntax:

```
def function_name(parameter1, parameter2):
    """docstring"""
    # function body
    # write some action

    return value
```

Functions

Create function (required)

Function inputs: here there are two and order matters. (optional)

```
def times(x,y):

    product = x*y

    return product
```

Object that function gives back. (optional)

Functions

Create function (required)

Function inputs: here there are two and order matters. (optional)

```
def times(x,y):

    product = x*y

    return product
```

Inside the function (local scope)

Object that function gives back. (optional)

The code inside the function is never executed unless you call the function!

Functions

```
def times(x,y):

    product = x*y

    return product

times(5,6)
```

```
30
```

To call a function type the name and give the inputs required.

Functions

$$x = 5$$
$$y = 6$$

```
def times(x,y):

    product = x*y

    return product

times(5,6)
```

```
30
```

To call a function type the name and give the inputs required.

Functions

$$x = 5$$
$$y = 6$$
$$\text{product} = 30$$

```
def times(x,y):

    product = x*y

    return product

times(5,6)
```

```
30
```

To call a function type the name and give the inputs required.

Functions

$$x = 5$$
$$y = 6$$
$$\text{product} = 30$$

```
def times(x,y):

    product = x*y

    return product

times(5,6)
```

```
30
```

To call a function type the name and give the inputs required.

Functions

$$x = 5$$
$$y = 6$$
$$\text{product} = 30$$

```
def times(x,y):

    product = x*y

    return product

times(5,6)
```

```
30
```

To call a function type the name and give the inputs required.

Functions

```
def times(x,y):

    product = x*y

    return product

times(5,6)
print(product)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-aea8a152219b> in <module>()
      6
      7 times(5,6)
----> 8 print(product)

NameError: name 'product' is not defined
```

The variables product doesn't exist outside of the function…but we can store the object returned by the function!

Functions

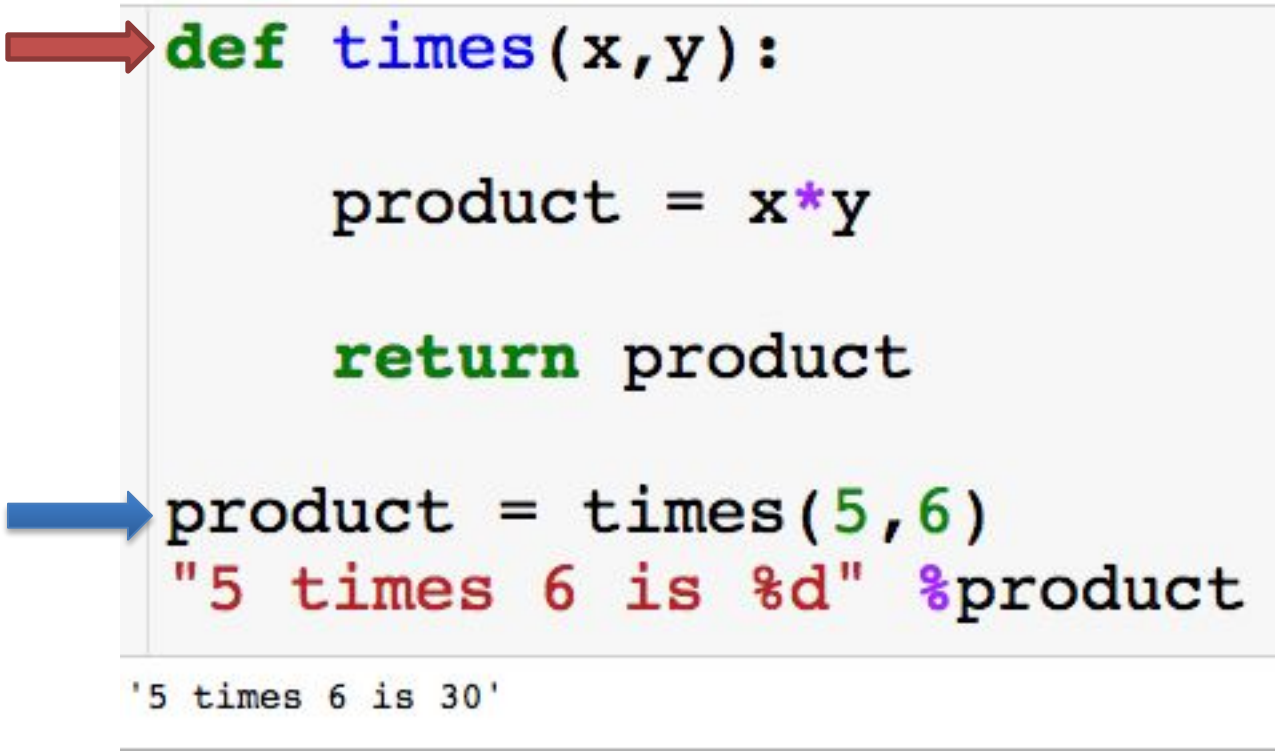
We can store the result returned from the function in a variables

```
def times(x,y):

    product = x*y

    return product

product = times(5,6)
"5 times 6 is %d" %product
```

```
'5 times 6 is 30'
```

Functions

We can store the result returned from the function in a variables

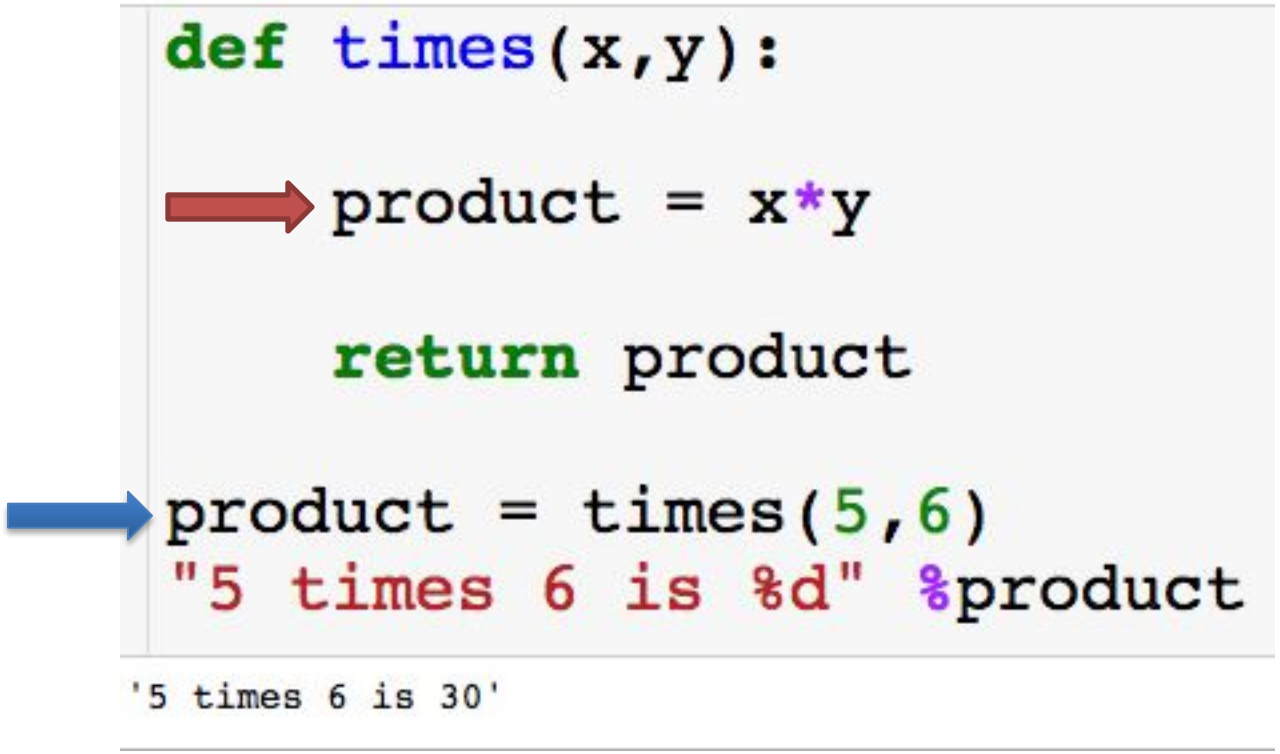


```
def times(x,y):

    product = x*y

    return product

product = times(5,6)
"5 times 6 is %d" %product
```

```
'5 times 6 is 30'
```

Functions

We can store the result returned from the function in a variables

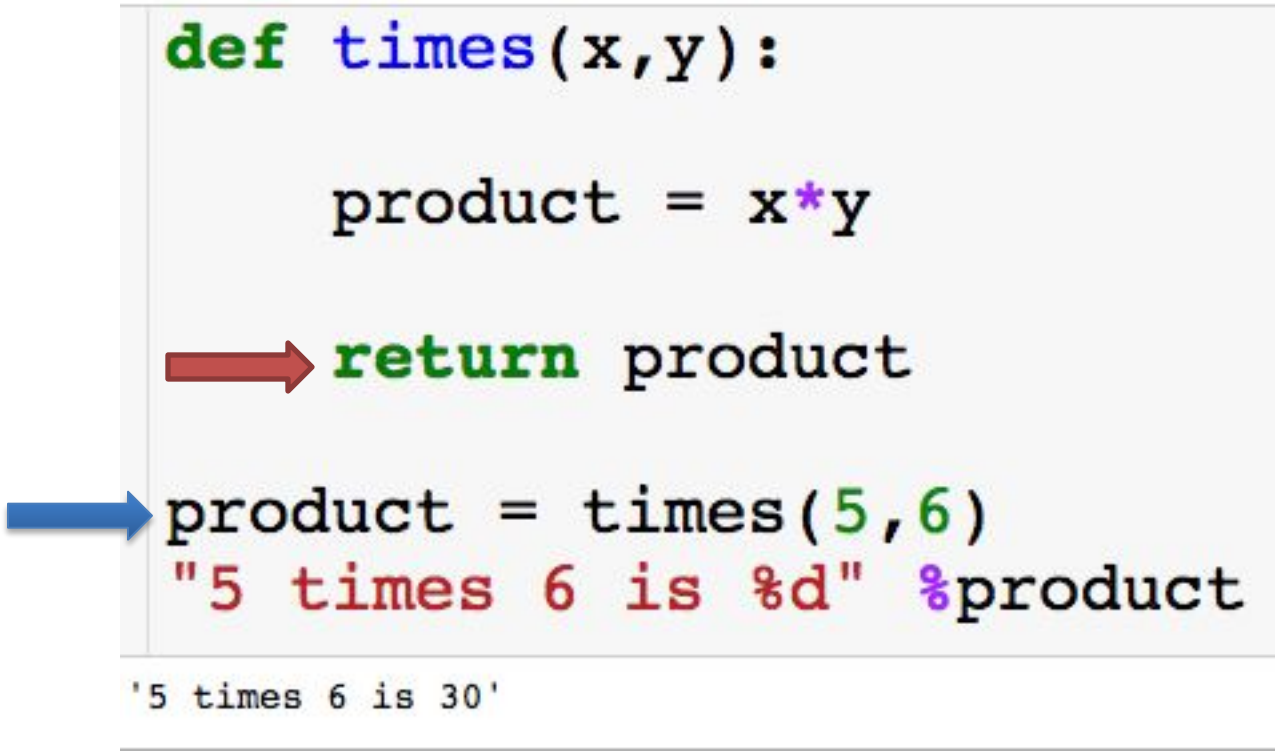


```
def times(x,y):

    product = x*y

    return product

product = times(5,6)
"5 times 6 is %d" %product
```

```
'5 times 6 is 30'
```

Functions

We can store the result returned from the function in a variables

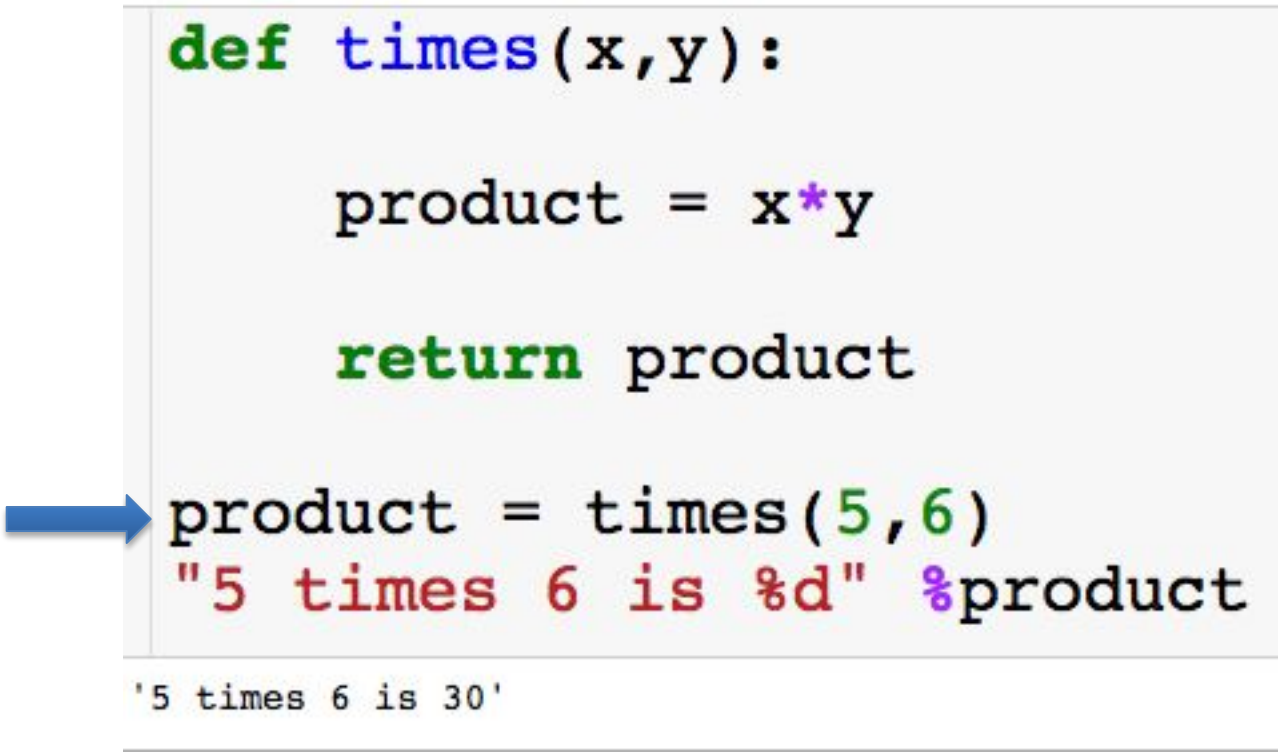


```
def times(x,y):

    product = x*y

    return product

product = times(5,6)
"5 times 6 is %d" %product
```

```
'5 times 6 is 30'
```

Functions

We can store the result returned from the function in a variables



```
def times(x,y):

    product = x*y

    return product

product = times(5,6)
"5 times 6 is %d" %product
```

```
'5 times 6 is 30'
```

We did not have to name this variable product, we could have names it anything.

Functions

We can store the result returned from the function in a variables

```
def times(x,y):

    product = x*y

    return product

product = times(5,6)
"5 times 6 is %d" %product
```

```
'5 times 6 is 30'
```

Functions

We can store the result returned from the function in a variables

```
def times(x,y):

    product = x*y

    return product

result = times(5,6)
print(result)
```

```
30
```

Functions

We can store the result returned from the function in a variables

```
def times(x,y):

    product = x*y

    return product

result = times(5,6)
print(result)
```

```
30
```

- What will happen if I try to print product? Or x?

Functions

We can store the result returned from the function in a variables

```
def times(x,y):

    product = x*y

    return product

result = times(5,6)
print(result)
```

```
30
```

- What will happen if I try to print product? Or x?
    - Where does each variables “live”?

Scope

- Python scopes are the places where variables are defined and looked up.

- Local Scope
  - Variables created with a function, i.e., inside of a def.
  - Variables inside of a def will not clash with variables outside even if they have the same name.

- Global Scope
  - Variable created outside of a function.

Scope

- Python scopes are the places where variables are defined and looked up.

- Local Scope
  - Variables created with a function, i.e., inside of a def.
  - Variables inside of a def will not clash with variables outside even if they have the same name.

- Global Scope
  - Variable created outside of a function.

When outside of a function, python **only sees variables in the global scope**.

Scope

- Python scopes are the places where variables are defined and looked up.

- Local Scope
  - Variables created with a function, i.e., inside of a def.
  - Variables inside of a def will not clash with variables outside even if they have the same name.

- Global Scope
  - Variable created outside of a function.

When inside of a function, python **first searches the local scope and then searches the global.**

Scope

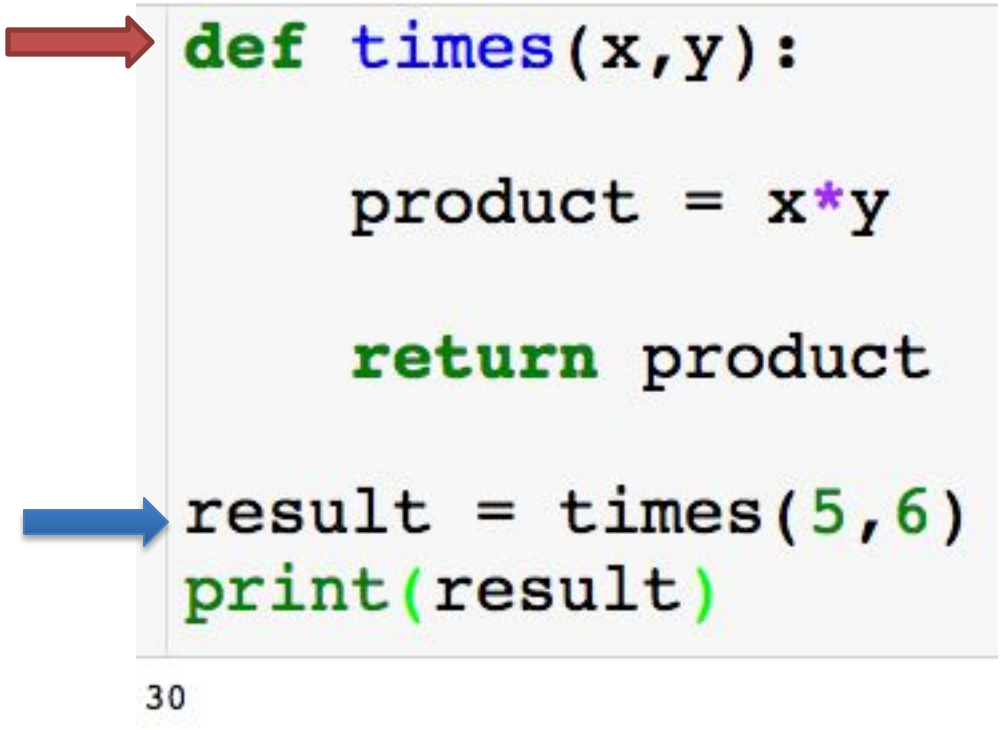
```
def times(x,y):

    product = x*y

    return product

result = times(5,6)
print(result)
```

```
30
```

Local scope:

Global scope:

Scope



```
def times(x,y):

    product = x*y

    return product

result = times(5,6)
print(result)
```

```
30
```

Local scope:

$$x = 5$$
$$y = 6$$

Global scope:

Scope

```
def times(x,y):

    product = x*y

    return product

result = times(5,6)
print(result)
```

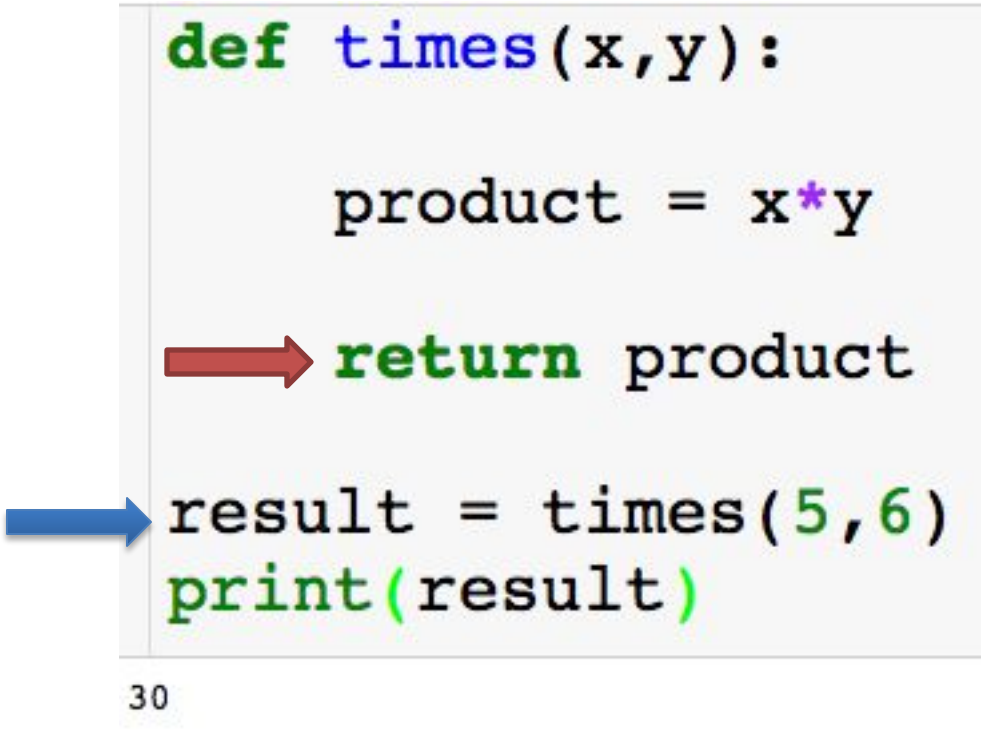
```
30
```

Local scope:

$$x = 5$$
$$y = 6$$
$$\text{product} = 30$$

Global scope:

Scope



Local scope:

$$x = 5$$
$$y = 6$$
$$\text{product} = 30$$

Global scope:

Scope

```
def times(x,y):

    product = x*y

    return product

result = times(5,6)
print(result)
```

```
30
```

Local scope:

$$x = 5$$
$$y = 6$$
$$\text{product} = 30$$

Global scope:

$$\text{result} = 30$$

Scope

```
def times(x,y):

    product = x*y

    return product

result = times(5,6)
print(result)
```

```
30
```

Local scope:

$$x = 5$$
$$y = 6$$
$$\text{product} = 30$$

Global scope:

$$\text{result} = 30$$

Scope

```
def intersect(seq1, seq2):
    res = []
    for x in seq1:
        if x in seq2:
            res.append(x)
    return res

S1="Spam"
S2 = "Scan"
intersect(S1,S2)
```

```
['S', 'a']
```

Local scope:

Global scope:

$$S1 = \text{“Spam”}$$

Scope

```
def intersect(seq1, seq2):
    res = []
    for x in seq1:
        if x in seq2:
            res.append(x)
    return res

S1="Spam"
S2 = "Scan"
intersect(S1,S2)
```

```
['S', 'a']
```

Local scope:

Global scope:

$$S1 = \text{“Spam”}$$
$$S2 = \text{“Scan”}$$

Scope

```
def intersect(seq1, seq2):
    res = []
    for x in seq1:
        if x in seq2:
            res.append(x)
    return res

S1="Spam"
S2 = "Scan"
intersect(S1,S2)
```

```
['S', 'a']
```

Local scope:

Global scope:

$$S1 = \text{“Spam”}$$
$$S2 = \text{“Scan”}$$

Scope

```
def intersect(seq1, seq2):
    res = []
    for x in seq1:
        if x in seq2:
            res.append(x)
    return res

S1="Spam"
S2 = "Scan"
intersect(S1,S2)
```

```
['S', 'a']
```

Local scope:

$$\text{seq1} = \text{“Spam”}$$
$$\text{seq2} = \text{“Scan”}$$

Global scope:

$$\text{S1} = \text{“Spam”}$$
$$\text{S2} = \text{“Scan”}$$

Scope

```
def intersect(seq1, seq2):
    res = []
    for x in seq1:
        if x in seq2:
            res.append(x)
    return res

S1="Spam"
S2 = "Scan"
intersect(S1,S2)
```

```
['S', 'a']
```

Local scope:

$$\text{seq1} = \text{“Spam”}$$
$$\text{seq2} = \text{“Scan”}$$
$$\text{res} = [\,]$$

Global scope:

$$\text{S1} = \text{“Spam”}$$
$$\text{S2} = \text{“Scan”}$$

Scope

```
def intersect(seq1, seq2):
    res = []
    for x in seq1:
        if x in seq2:
            res.append(x)
    return res

S1="Spam"
S2 = "Scan"
intersect(S1,S2)
```

```
['S', 'a']
```

Local scope:

seq1 = "Spam"
seq2 = "Scan"
res = [ ]
x= "S"

Global scope:

S1 = "Spam"
S2 = "Scan"

Scope

```
def intersect(seq1, seq2):
    res = []
    for x in seq1:
        if x in seq2:
            res.append(x)
    return res

S1="Spam"
S2 = "Scan"
intersect(S1,S2)
```

```
['S', 'a']
```

Local scope:

seq1 = "Spam"
seq2 = "Scan"
res = [ ]
x= "S"

Global scope:

S1 = "Spam"
S2 = "Scan"

Scope

```
def intersect(seq1, seq2):
    res = []
    for x in seq1:
        if x in seq2:
            res.append(x)
    return res

S1="Spam"
S2 = "Scan"
intersect(S1,S2)
```

```
['S', 'a']
```

Local scope:

seq1 = "Spam"
seq2 = "Scan"
res = ["S"]
x= "S"

Global scope:

S1 = "Spam"
S2 = "Scan"

Scope

```
def intersect(seq1, seq2):
    res = []
    for x in seq1:
        if x in seq2:
            res.append(x)
    return res

S1="Spam"
S2 = "Scan"
intersect(S1,S2)
```

```
['S', 'a']
```

Local scope:

seq1 = "Spam"
seq2 = "Scan"
res = ["S"]
x= "p"

Global scope:

S1 = "Spam"
S2 = "Scan"

And so on …

Stringing Functions Together

```
def Add(x,y):

    return x+y
```

```
def Check_Mult_Two(x):

    if x %2==0:

        print("Yes, %d is a mulitple of 2" %x)

    else:

        print("No, %d is not mulitple of 2" %x)
```

Stringing Functions Together

```
def Add(x,y):

    return x+y
```

```
def Check_Mult_Two(x):

    if x %2==0:

        print("Yes, %d is a mulitple of 2" %x)

    else:

        print("No, %d is not mulitple of 2" %x)
```

```
num_one = 10
num_two = 15

sum_one_two = Add(num_one,num_two)
Check_Mult_Two(sum_one_two)
```

```
No, 25 is not mulitple of 2
```

Stringing Functions Together

```
def Add(x,y):

    return x+y
```

```
def Check_Mult_Two(x):

    if x %2==0:

        print("Yes, %d is a mulitple of 2" %x)

    else:

        print("No, %d is not mulitple of 2" %x)
```

```
num_one = 10
num_two = 15

sum_one_two = Add(num_one,num_two)
Check_Mult_Two(sum_one_two)
```

```
No, 25 is not mulitple of 2
```

Stringing Functions Together

```
def Add(x,y):

    return x+y
```

```
def Check_Mult_Two(x):

    if x %2==0:

        print("Yes, %d is a mulitple of 2" %x)

    else:

        print("No, %d is not mulitple of 2" %x)
```

```
num_one = 10
num_two = 15

sum_one_two = Add(num_one,num_two)
Check_Mult_Two(sum_one_two)
```

```
No, 25 is not mulitple of 2
```

Stringing Functions Together

```
def Add(x,y):

    return x+y
```

```
def Check_Mult_Two(x):

    if x %2==0:

        print("Yes, %d is a mulitple of 2" %x)

    else:

        print("No, %d is not mulitple of 2" %x)
```

```
num_one = 10
num_two = 15

sum_one_two = Add(num_one,num_two)
Check_Mult_Two(sum_one_two)
```

```
No, 25 is not mulitple of 2
```

Stringing Functions Together

```
def Add(x,y):

    return x+y
```

```
def Check_Mult_Two(x):

    if x %2==0:

        print("Yes, %d is a mulitple of 2" %x)

    else:

        print("No, %d is not mulitple of 2" %x)
```

```
num_one = 10
num_two = 15

sum_one_two = Add(num_one,num_two)
Check_Mult_Two(sum_one_two)
```

```
No, 25 is not mulitple of 2
```

Stringing Functions Together

```
def Add(x,y):

    return x+y
```

```
def Check_Mult_Two(x):

    if x %2==0:

        print("Yes, %d is a mulitple of 2" %x)

    else:

        print("No, %d is not mulitple of 2" %x)
```

```
num_one = 10
num_two = 15

sum_one_two = Add(num_one,num_two)
Check_Mult_Two(sum_one_two)
```

```
No, 25 is not mulitple of 2
```

Stringing Functions Together

```
def Add(x,y):

    return x+y
```

```
def Check_Mult_Two(x):

    if x %2==0:

        print("Yes, %d is a mulitple of 2" %x)

    else:

        print("No, %d is not mulitple of 2" %x)
```

```
num_one = 10
num_two = 15

sum_one_two = Add(num_one,num_two)
Check_Mult_Two(sum_one_two)
```

```
No, 25 is not mulitple of 2
```

Stringing Functions Together

```
def Add(x,y):

    return x+y
```

```
def Check_Mult_Two(x):

    if x %2==0:

        print("Yes, %d is a mulitple of 2" %x)

    else:

        print("No, %d is not mulitple of 2" %x)
```

```
num_one = 10
num_two = 15

sum_one_two = Add(num_one,num_two)
Check_Mult_Two(sum_one_two)
```

```
No, 25 is not mulitple of 2
```

Nested Functions

Functions can call other functions!

Nested Functions

```
def Count_Vowels(name):
    total=0
    vowels = ['a','e','i','o','u']
    for i in name.lower():
        if i in vowels:
            total+=1
    return total
```

Count number of vowels in name

```
def Percent_Vowels(name):

    num_vowels = Count_Vowels(name)
    percent = num_vowels/len(name)
    return percent
```

Computes percentage of vowels in name

Nested Functions

```
def Count_Vowels(name):
    total=0
    vowels = ['a','e','i','o','u']
    for i in name.lower():
        if i in vowels:
            total+=1
    return total
```

Count number of vowels in name

```
def Percent_Vowels(name):

    num_vowels = Count_Vowels(name)
    percent = num_vowels/len(name)
    return percent
```

Computes percentage of vowels in name

```
Percent_Vowels("jake")
```

Nested Functions

```
def Count_Vowels(name):
    total=0
    vowels = ['a','e','i','o','u']
    for i in name.lower():
        if i in vowels:
            total+=1
    return total
```

Count number of vowels in name

```
def Percent_Vowels(name):

    num_vowels = Count_Vowels(name)
    percent = num_vowels/len(name)
    return percent
```

Computes percentage of vowels in name

```
Percent_Vowels("jake")
```

Nested Functions

```
def Count_Vowels(name):
    total=0
    vowels = ['a','e','i','o','u']
    for i in name.lower():
        if i in vowels:
            total+=1
    return total
```

Count number of vowels in name

```
def Percent_Vowels(name):

    num_vowels = Count_Vowels(name)
    percent = num_vowels/len(name)
    return percent
```

Computes percentage of vowels in name

```
Percent_Vowels("jake")
```

Nested Functions

```
def Count_Vowels(name):
    total=0
    vowels = ['a','e','i','o','u']
    for i in name.lower():
        if i in vowels:
            total+=1
    return total
```

Count number of vowels in name

```
def Percent_Vowels(name):

    num_vowels = Count_Vowels(name)
    percent = num_vowels/len(name)
    return percent
```

Computes percentage of vowels in name

```
Percent_Vowels("jake")
```

Nested Functions

```
def Count_Vowels(name):
    total=0
    vowels = ['a','e','i','o','u']
    for i in name.lower():
        if i in vowels:
            total+=1
    return total
```

Count number of vowels in name

```
def Percent_Vowels(name):

    num_vowels = Count_Vowels(name)
    percent = num_vowels/len(name)
    return percent
```

Computes percentage of vowels in name

```
Percent_Vowels("jake")
```

Nested Functions

```
def Count_Vowels(name):
    total=0
    vowels = ['a','e','i','o','u']
    for i in name.lower():
        if i in vowels:
            total+=1
    return total
```

Count number of vowels in name

```
def Percent_Vowels(name):

    num_vowels = Count_Vowels(name)
    percent = num_vowels/len(name)
    return percent
```

Computes percentage of vowels in name

```
Percent_Vowels("jake")
```

Nested Functions

```
def Count_Vowels(name):
    total=0
    vowels = ['a','e','i','o','u']
    for i in name.lower():
        if i in vowels:
            total+=1
    return total
```

Count number of vowels in name

```
def Percent_Vowels(name):

    num_vowels = Count_Vowels(name)
    percent = num_vowels/len(name)
    return percent
```

Computes percentage of vowels in name

```
Percent_Vowels("jake")
```

Nested Functions

```
def Count_Vowels(name):
    total=0
    vowels = ['a','e','i','o','u']
    for i in name.lower():
        if i in vowels:
            total+=1
    return total
```

Count number of vowels in name

```
def Percent_Vowels(name):

    num_vowels = Count_Vowels(name)
    percent = num_vowels/len(name)
    return percent
```

Computes percentage of vowels in name

```
Percent_Vowels("jake")
```

Nested Functions

```
def Count_Vowels(name):
    total=0
    vowels = ['a','e','i','o','u']
    for i in name.lower():
        if i in vowels:
            total+=1
    return total
```

Count number of vowels in name

```
def Percent_Vowels(name):

    num_vowels = Count_Vowels(name)
    percent = num_vowels/len(name)
    return percent
```

Computes percentage of vowels in name

```
Percent_Vowels("jake")
```

```
0.5
```

Nested Functions

```
def Count_Vowels(name):
    total=0
    vowels = ['a','e','i','o','u']
    for i in name.lower():
        if i in vowels:
            total+=1
    return total
```

Count number of vowels in name

```
def Percent_Vowels(name):

    num_vowels = Count_Vowels(name)
    percent = num_vowels/len(name)
    return percent
```

Computes percentage of vowels in name

```
first_names = ["Jake", "Joe", "Tarik"]
D = {}
for name in first_names:
    D[name] = Percent_Vowels(name)

D
```

```
{'Jake': 0.5, 'Joe': 0.6666666666666666, 'Tarik': 0.4}
```

Function Exercises

Function Exercises

1. Area of a circle is calculated as follows: **area = π x r x r** and **perimeter = 2 x π x r**. Write a function that calculates `area_of_circle` and `perimeter_of_circle`.

```
[ ]:
```

2. Declare a function named `sum_of_numbers`. It takes a number parameter and it adds all the numbers in that range.

```
print(sum_of_numbers(5))  # 15
print(sum_all_numbers(10)) # 55
print(sum_all_numbers(100)) # 5050
```

```
[ ]:
```

3. Declare a function named `evens_and_odds`. It takes a positive integer as parameter and it counts number of evens and odds in the number.

```
print(evens_and_odds(100))
#The number of odds are 50.
#The number of evens are 51.
```

Function Exercises

4. Write a function calculate_mean() that takes a list of numbers and returns the mean (average).

[ ]:

5. Write a function called sum_all() that accepts any number of arguments and returns their total.

[ ]:

6. Write a function called get_sum_avg() that returns both the sum and the average of any number of arguments passed.

[ ]:

7. Write a function analyze_list(lst) that returns below as as a tuple:
   - the average of the list
   - the maximum number
   - and the length of the list

[ ]:

Function Exercises

8: `calculate_median(list)`

Write a function called `calculate_median()` that takes a list of numbers and returns the **median** value.

The **median** is the middle number in a sorted list:

- If the list has **odd** length → return the middle number.
- If the list has **even** length → return the average of the two middle numbers.

💡 **Hint:**

- Use the `sorted()` function to sort the list.
- Use `len()` to get the list length.
- Use integer division (`//`) to find the middle index.

✅ **Example Output:**

```
calculate_median([4, 1, 9])         # Output: 4
calculate_median([1, 2, 3, 4])      # Output: 2.5
```