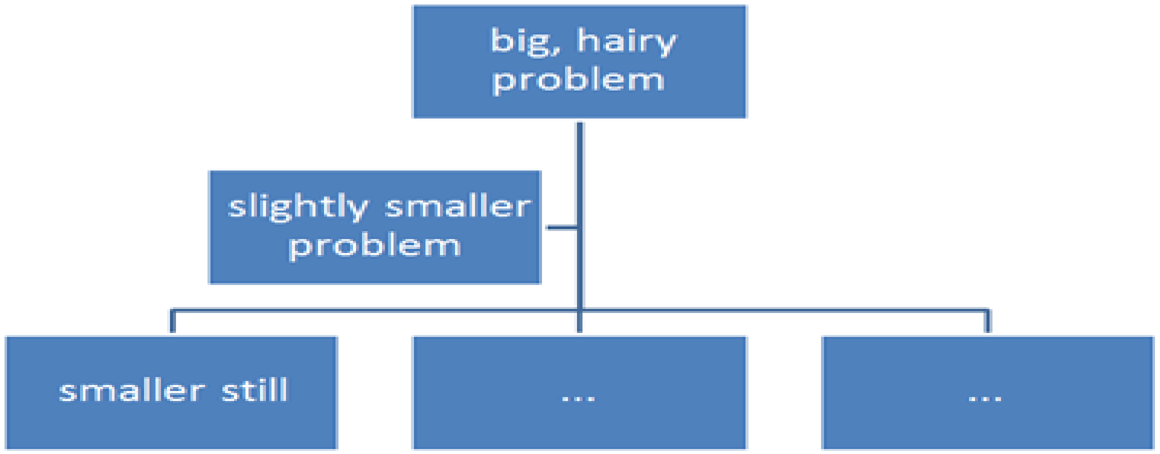
Functions



Python Functions

- A function is a block of organized, **reusable code** that is used to perform a single, related action.
- Functions provide better **modularity** for your application and a high degree of code reusing.
- As you already know, Python gives you many **built-in functions** like print(), etc. but you can also create your own functions.
- These functions are called **user-defined functions**.

Why Functions?

- To reduce code duplication and increase program modularity.
- A software cannot be implemented by any one person, it takes a team of programmers to develop such a project.

Functions Contd...

- In order to manage the complexity of a large problem, it is broken down into smaller sub problems. Then, each sub problem can be focused on and solved separately.

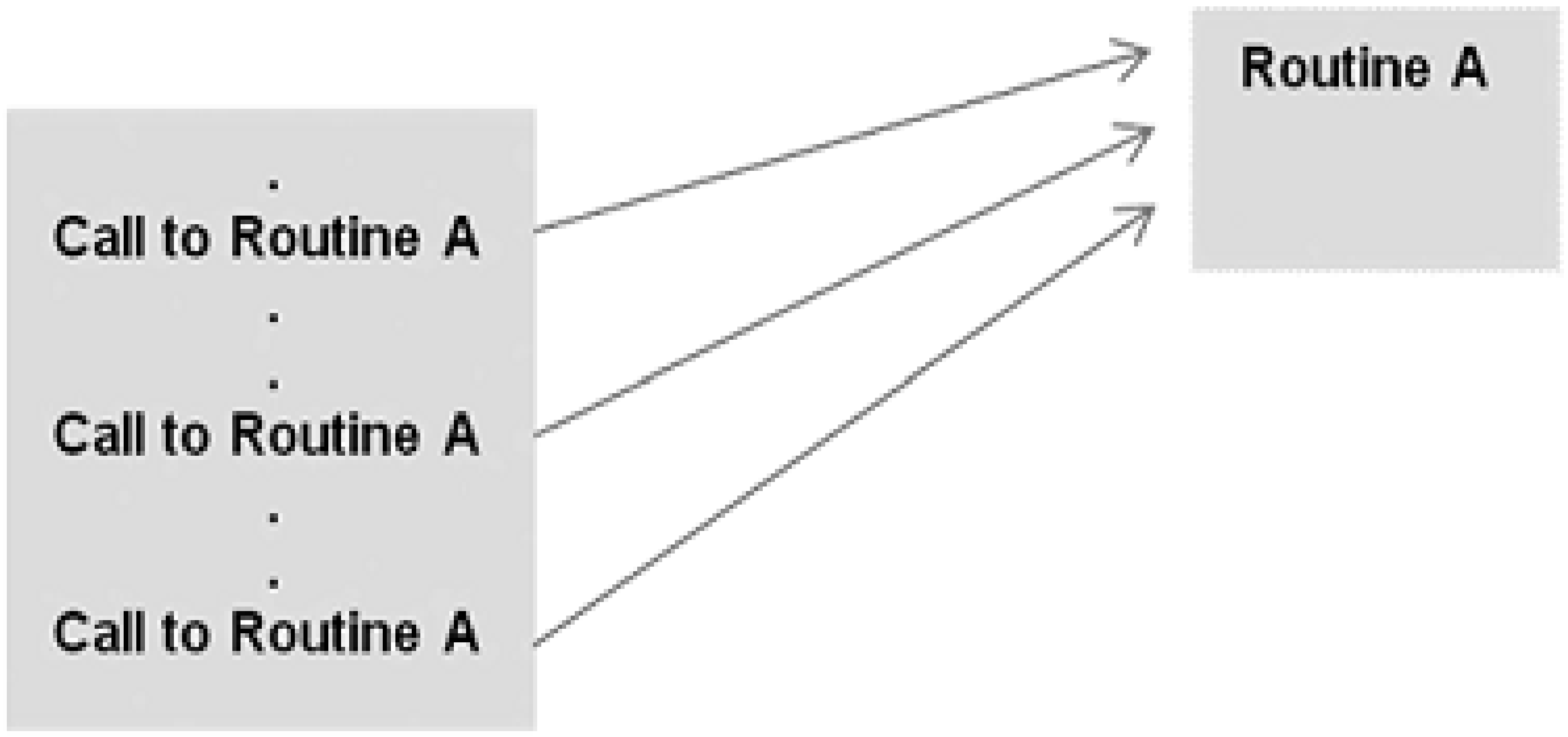
- Program routines provide the opportunity for code reuse, so that systems do not have to be created from “scratch.”

What Is a Function Routine?

- A **function or routine** is a named group of instructions performing some task.

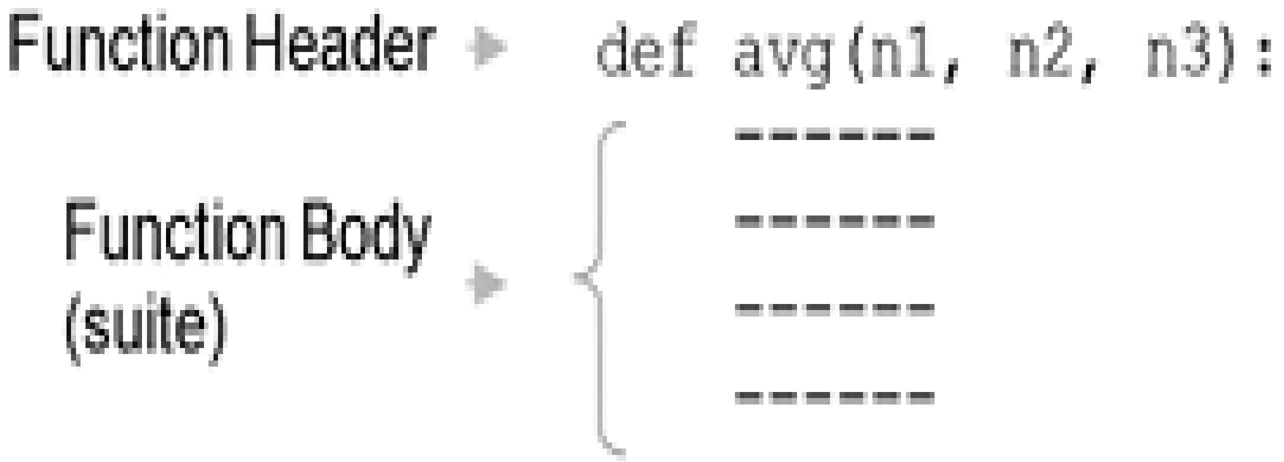
- A function can be **invoked** (*called*) as many times as needed in a given program.

- When a routine terminates, execution automatically returns to the point from which it was called.


.
Call to Routine A
.
.
Call to Routine A
.
.
Call to Routine A
Routine A

Defining Functions

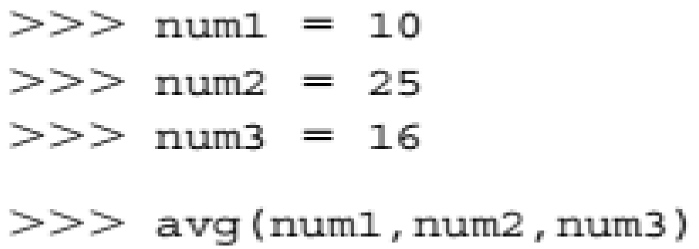
- Functions may be designed as per user's requirement.
- The elements of a function definition are given



```
def avg(n1, n2, n3):
    -------
    -------
    -------
    -------
```

Defining Functions   Contd…

- The number of items in a parameter list indicates the number of values that must be passed to the function, called **actual arguments** (or simply “arguments”), such as the variables num1, num2,and num3 below.



```
>>> num1 = 10
>>> num2 = 25
>>> num3 = 16

>>> avg(num1,num2,num3)
```

- *Every function must be defined before it is called.*

Creating a Function

#Ex 1:

```
def my_function():
  print("Hello VIT")
```

# Ex 2 – with one argument:

```
def printme( str ):
        print(str)
```

Calling a Function

```
# Ex 2: calling my_function function
my_function()

# Ex 2: calling printme function
printme("It’s the first call")
printme("Again the second call")
```

Output

- When the above code is executed, it produces the following result –

```
printme("It’s the first call")
printme("Again the second call")
```

O/p:

```
It‘s the first call
Again the second call
```

Parameters or Arguments

- You can call a function by using the following types of formal parameters –
  - Required parameters
  - Keyword parameters
  - Default parameters

Required parameters

- Required arguments are the arguments passed to a function in correct positional order.
- A function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.

Required arguments

- To call the function *printme()*, you definitely need to pass one argument, otherwise it gives a syntax error as follows –

```
# Function definition is here
def printme( str ):
    print(str)

# Now you can call printme function
printme()
```

Required arguments

- When the above code is executed, it produces the following result –

```
printme()
Traceback (most recent call last):
  File "test.py", line 11, in <module>
    printme();
TypeError: printme() takes exactly 1 argument (0 given)
```

Required arguments

```
printme(“Hello”, “VIT”)
```

```
Traceback (most recent call last):
  File "C:/Users/admin/AppData/Local/Programs/Python/Python37-32/ff1.py", line 3, in <module>
    printme("Hello", "VIT")
TypeError: printme() takes 1 positional argument but 2 were given
```

Keyword parameters

- This allows you to place arguments out of order because the Python interpreter is able to use the keywords provided to match the values with parameters.
- Keyword arguments are related to the function calls.
- When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.

Keyword parameters

- The following example gives more clear picture. Note that the order of parameters does not matter.

```
def printinfo( name, age ):
        print("Name: ", name)
        print ("Age ", age)
# Now you can call printinfo function
printinfo( age=50, name="miki" )
```

- Output:

```
Name:  miki
Age  50
```

Keyword parameters

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)
my_function(child1 = “Raj", child2 = “Sam", child3 = “Tim")
```

Output:

```
The youngest child is Tim
```

Default arguments

- A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

```
def printinfo( name, age = 35 ):
        print "Name: ", name
        print "Age ", age
# Now you can call printinfo function
printinfo( name=“Ram" )   #passing one parameter
printinfo( name=“Sam”, 55) #passing two parameter
```

Default arguments

- When the above code is executed, it produces the following result –

```
Name:  Ram
Age  35
Name:  Sam
Age  55
```

Example Functions

```
def printer(message):
	print('Hello ' + message)
```

Example Functions

```
>>> def times(x, y):
        # Create and assign function ...
                return x * y
        # Body executed when called ...
```

When Python reaches and runs this def, it creates a new function object that packages the function’s code and assigns the object to the name times

Calls

```
def times(x,y):
    return x*y
x=times(2,3)
print(x)
y=times(3.14,4)
print(y)
z=times('Ni',4)    # Functions are "typeless"
print(z)
```

```
Output:
6
12.56
NiNiNiNi
```

More Example

```
def intersect(seq1, seq2):
 res=[]
 for x in seq1:
  if x in seq2:
   res.append(x)
 return(res)
print(intersect("spamy","spamx"))
s1="Ramu"
s2="Raju"
s3=intersect(s1,s2)
print(s3)
```

```
Output:
['s', 'p', 'a', 'm']
['R', 'a', 'u']
```

Function definition in selection statement Example

```
a=4
if a%2==0:
    def func():
        print('even')
else:
    def func():
        print("odd")
func()
```

Output:

```
even
```

Scope of Variables

- Enclosing module is a global scope
- Global scope spans a single file only
- Assigned names are local unless declared global or nonlocal
- Each call to a function creates a new local scope

Scope Example

```
# Global scope
X = 99
# X and func assigned in module: global
def func(Y):
        # Y and Z assigned in function: locals
        # Local scope
        Z = X + Y        # X is a global
        print(Z)
func(1)                  # func in module: result=100
```

Scope Example

- Global names: X, func
- Local names: Y, Z

Scope Example

```
X = 88                          # Global X
def func():
        X = 99
        print(X)
# Local X: hides global
func()                       # Prints 99
print(X)                        # Prints 88
```

Accessing Global Variables

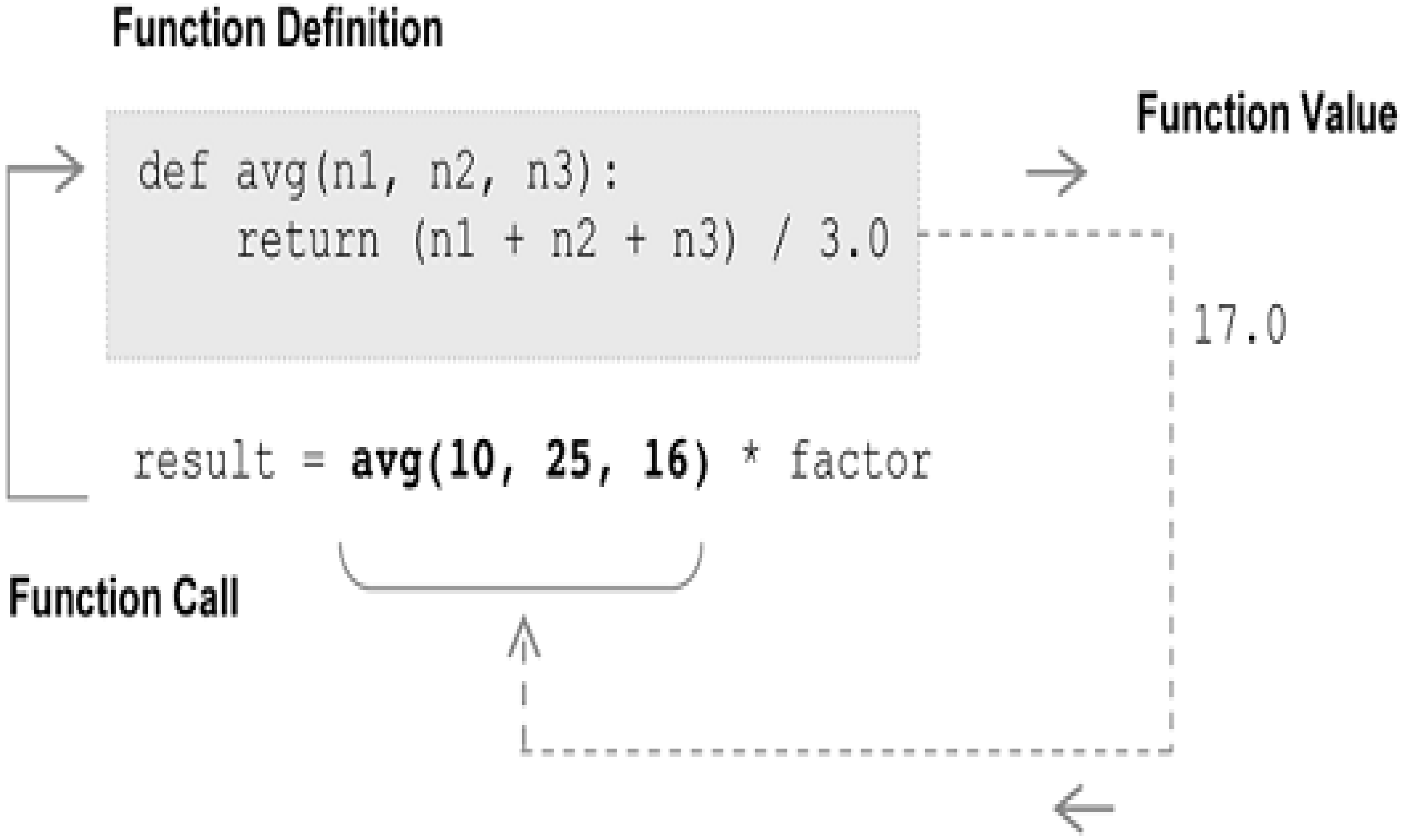
```
X = 88                      # Global X
def func():
    global X                #Global X accessed
    X = 99                  # Global X: outside def
func()                       # Prints 99
print(X)                    # Prints 99
```

Global Variables and Global Scope

- The use of global variables is generally considered to be bad programming style.

Value-Returning Functions

- Program routine called for its return value, and is therefore similar to a mathematical function.
- Function avg takes three arguments (n1, n2, and n3) and returns the average of the three.
- The *function call* avg(10, 25, 16), therefore, is an expression that evaluates to the returned function value.
- This is indicated in the function's *return statement* of the form return *expr*, where *expr* may be any expression.

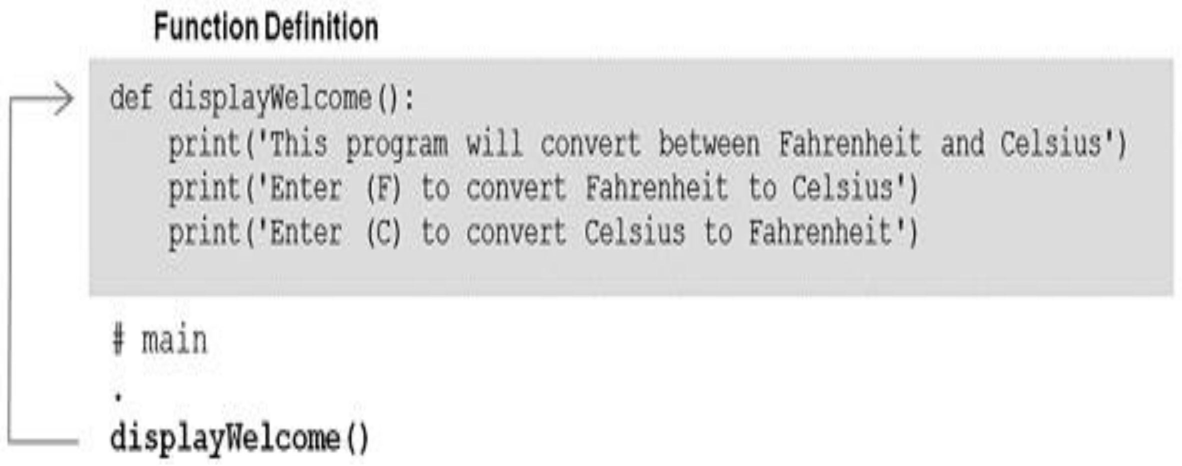


```
def avg(n1, n2, n3):
    return (n1 + n2 + n3) / 3.0
```

```
result = avg(10, 25, 16) * factor
```

Non-Value-Returning Functions

- A **non-value-returning function** is called not for a returned value, but for its *side effects*.
- A **side effect** is an action other than returning a function value, such as displaying output on the screen.


Function Definition

```
def displayWelcome():
    print('This program will convert between Fahrenheit and Celsius')
    print('Enter (F) to convert Fahrenheit to Celsius')
    print('Enter (C) to convert Celsius to Fahrenheit')

# main
.
displayWelcome()
```

- In this example, function display Welcome is called only for the side-effect of the screen output produced.

Returning Multiple Values

```
def multiple(a,b):
    a=2
    b=[3,4,5]
    return a,b
x,y=multiple(3,[1,2])
print(x)
print(y)
# Return multiple new values in a tuple
```

```
Output:
2
[3, 4, 5]
```

More Examples

```
def my_function(fname):
  print(fname + " Team India")

my_function("Kholi")
my_function("Rohit")
my_function("Jadu")
```

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Virat", "Kohli")
```

```
def my_function(child3, child2, child1):
  print("The youngest player is " + child3)

my_function(child1 = "Jadeja", child2 = "Virat
   kholi", child3 = "Anukul Roy")

The youngest player is Anukul Roy
```

Default arguments Ex..

```
def my_function(country = "Norway"):
    print("I am from " + country)

  my_function("Sweden")
  my_function("India")
  my_function()
  my_function("Brazil")
```

```
  I am from Sweden
  I am from India
  I am from Norway
  I am from Brazil
```

Default arguments Ex..

```
def printinfo( name, age = 35 ):
   print "Name: ", name
   print "Age ", age

# Now you can call printinfo function
printinfo( age=50, name="miki" )
printinfo( name="miki" )
```

Passing a List as an Argument

```
def my_function(food):
    for i in food:
        print(i)
fruits = ["apple","banana","cherry"]
my_function(fruits)
```

Output:

```
    apple
    banana
    cherry
```

```
def my_function(x):
    return 5 * x

  print(my_function(3))
  print(my_function(5))
  print(my_function(9))
```

Calculator

```
# This function adds two numbers
def add(x, y):
   return x + y

# This function subtracts two numbers
def subtract(x, y):
   return x - y

# This function multiplies two numbers
def multiply(x, y):
   return x * y

# This function divides two numbers
def divide(x, y):
   return x / y
```

```
print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")
# Take input from the user
choice = input("Enter choice(1/2/3/4):")
num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))
if choice == '1':
   print(num1,"+",num2,"=", add(num1,num2))
elif choice == '2':
   print(num1,"-",num2,"=", subtract(num1,num2))
elif choice == '3':
   print(num1,"*",num2,"=", multiply(num1,num2))
elif choice == '4':
   print(num1,"/",num2,"=", divide(num1,num2))
else:
   print("Invalid input")
```

Function Call through variable

```
def one():
    print('one')

def two():
    print('two')

def three():
    print('three')
```

Function Call through variable

```
a = 3
 if a == 1:
    call_Func=one
elif a == 2:
    call_Func=two
else:
    call_Func=three
call_Func()
```

Triangle Formation Problem

Given three points, write an algorithm and the subsequent Python code to check if they can form a triangle. Three points can form a triangle, if they do not fall in a straight line and length of a side of triangle is less than the sum of length of other two sides of the triangle.

Triangle Formation Problem

For example, the points (5,10), (20,10) and (15,15) can form a triangle as they do not fall in a straight line and length of any side is less than sum of the length of the other two sides

Pseudocode for Triangle Formation

Read the three points

If the three points fall on a straight line then print **“No Triangle”** and break

Otherwise **find length of all three sides**

If length of one side is greater than the sum of length of the other two sides then print **“Triangle”** and print “No Triangle” otherwise

Pseudocode for Fall in Straight Line

input : X and Y coordinates of the three points

```
IF (pt1.x==pt2.x==pt3.x) THEN
        RETURN true
ELIF (pt1.y==pt2.y==pt3.y) THEN
        RETURN true
ELSE
        RETURN false
```

Pseudocode for Distance between Two Points (Length of a side in a triangle)

input : X and Y coordinates of the two points

Distance = sqrt((pt1.x-pt2.x)2 – (pt1.y-pt2.y)**2)**

Return distance

Pseudocode for Checking Length Constraint

```
**input :** Length of three sides l1, l2, and l3
   if l1<l2+l3 or l2<l1+l3 or l3<l1+l2:
return false
else:
return true
```