STRINGS and FUNCTIONS IN PYTHON

Online One Week Short Term Training Programme on
Python Programming-Day 3

Ms.M.KAMALA MALAR,
Assistant Professor/Information Technology,
Dr.Sivanthi Aditanar College of Engineering
Tiruchendur.

Strings

- A string is a sequence of characters.
- In Python, strings start and end with single or double quotes or triple quotes.

Strings

- We can create strings simply by enclosing characters in quotes. Python treats single quotes the same as double quotes.
- Creating strings is as simple as assigning a value to a variable. For example:

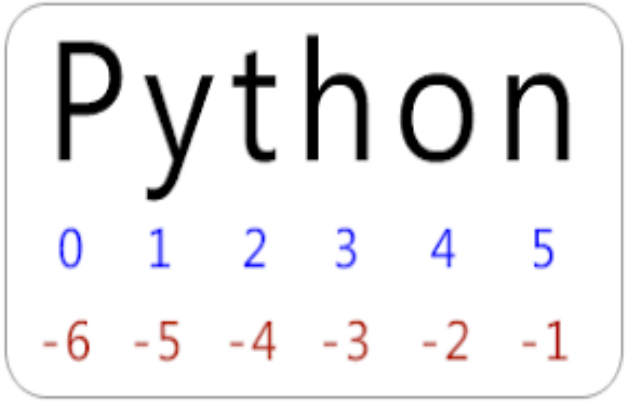
```
var1 = 'Hello World!'
var2 = "Python Programming"
var3 = '''Welcome to Python'''
```

Defining strings

- Each string is stored in the computer's memory as a list of characters.

```
>>> myString = "Python"
```

myString



String Operations

Concatenation and repetition

- Strings are concatenated with the + sign:

```
>>> 'abc'+'def'
'abcdef‘
```

- Strings are repeated with the * sign:

```
>>> 'abc'*3
'abcabcabc'
```

Indexing and slicing

- Python starts indexing at `0`.
- A string `s` will have indexes running from `0` to `len(s)-1` (where `len(s)` is the length of `s`) in integer quantities.
- `s[i]` fetches the i th element in s.

```
>>> s = 'string'
>>> s[1]
't'
```

Indexing and slicing

- `S='string'`
- `[i:j]` fetches elements i (inclusive) through j (not inclusive)

```
>>> s[1:4]
'tri'
```

- `s[:j]` fetches all elements up to j, but not including j

```
>>> s[:3]
'str'
```

- `s[i:]` fetches all elements from i onward (inclusive)

```
>>> s[2:]
'ring'
```

Indexing and slicing

- `S='string'`
- `s[i:j:k]` extracts every `k`th element starting with index `i` (inlcusive) and ending with index `j` (not inclusive)

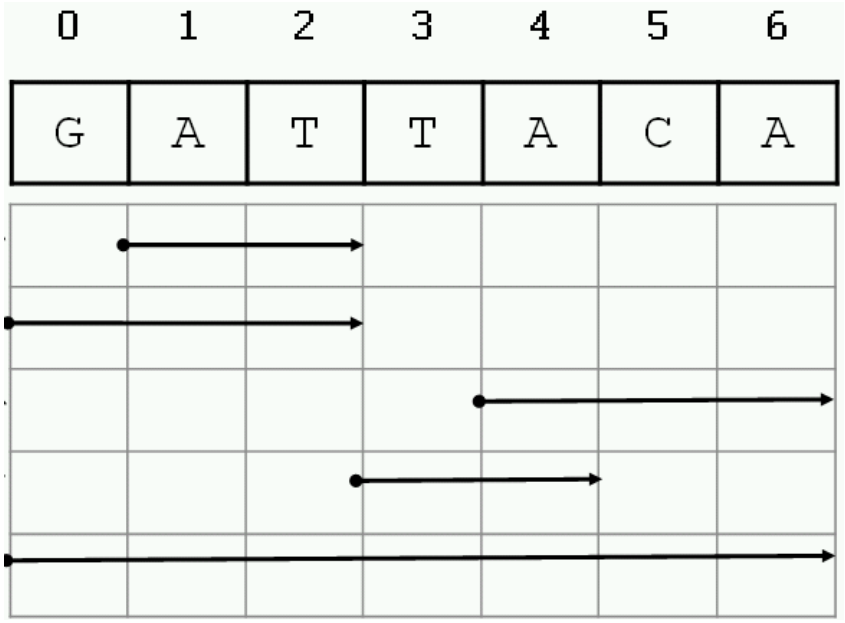
```
>>> s[0:5:2]
'srn'
```

- Python also supports negative indexes. For example, `s[-1]` means extract the first element of `s` from the end (same as `s[len(s)-1]`)

```
>>> s[-1]
'g'
>>> s[-2]
'n'
```

Indexing and slicing

```
>>> myString = “GATTACA”
>>> myString[1:3]
‘AT’
>>> myString[:3]
‘GAT’
>>> myString[4:]
‘ACA’
>>> myString[3:5]
‘TA’
>>> myString[:]
‘GATTACA’
```



Accessing Values in Strings

- Python does not support a character type; these are treated as strings of length one.
- **Example:**

```
var1 = "Python Programming"
print("var1[0]: ", var1[0])
print("var1[1:5]: ",var1[1:5])
```

print("var[8:4:-1]:",var[8:4:-1]) # print(var[4:8:-1]) As going backward i>j

print("var[4::-1]:",var[4::-1]) # j defaults to beginning of the string

print("var[:4:-1]:",var[:4:-1])

This will produce following result:

```
var1[0]: P
var1[1:5]: ytho
var[8:4:-1]: rP n
 var[4::-1]: ohtyP     var[:4:-1]: gnimmargorP n
```

slicing

P	y	t	h	o	n		P	r	o	g	r	a	m	m	i	n	g
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

String Immutability

Strings are immutable.

Example:

```
>>> greeting = 'Hello Python'
>>> greeting[0] = 'J'
```

Error will be displayed

(TypeError: 'str' object does not support item assignment)

String Immutability

One solution is to create a new string that is a variation on the original.

Example

```
>>> greeting = 'Hello Python'
>>> new_greeting = 'J' + greeting[1:]
>>> new_greeting
'Jello Python'
```

String Concatenation

- When the + operator is applied to strings, it means "concatenation"

```
>>> a = 'Hello'
>>> b = a + 'There'
>>> print (b)
HelloThere
>>> c = a + '  ' + 'There'
>>> print c
Hello There
>>>
```

Using in as an Operator

- The in keyword is used to check to see if one string is in another string.
- The in expression is a logical expression and returns True or False and can be used in an if statement

```
>>> fruit = 'banana’
>>> 'n' in fruit
True
>>> 'm' in fruit
False
>>> 'nan' in fruit
True
>>> if 'a' in fruit :
...     print ('Found it!’)
...
Found it!
>>>
```

STRING METHODS

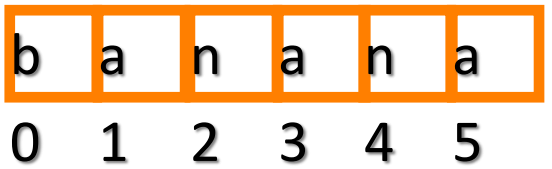
- In Python, a method is a function that is defined with respect to a particular object.

- The syntax is
  `<object>.<method>(<parameters>)`

```
>>> var = “Dr.SACOE”
>>> var.find(“S”)
3
```

Strings Have Length

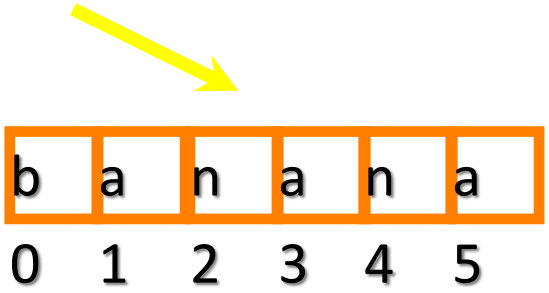
- There is a built-in function len that gives us the length of a string




```
>>> fruit = 'banana'
>>> print (len(fruit))
6
```

Searching a String

- We use the find() function to search for a substring within another string
- find() finds the first occurance of the substring
- If the substring is not found, find() returns -1
- Remember that string position starts at zero




```
>>> fruit = 'banana'
>>> pos = fruit.find('na')
>>> print (pos)
2
>>> aa = fruit.find('z')
>>> print (aa)
-1
```

UPPER CASE / LOWER CASE

- You can make a copy of a string in lower case or upper case
- Often when we are searching for a string using find() - we first convert the string to lower case so we can search a string regardless of case

```
>>> v1 = 'Hello Malar'
>>> v2= v1.upper()
>>> v2
HELLO MALAR
>>> v3 = v2.lower()
>>> v3
hello malar
>>>
```

Search and Replace

- The replace() function is like a "search and replace" operation in a word processor
- It replaces all occurrences of the search string with the replacement string

```
>>> v1 = 'Hello Sibi'
>>> v2=v1.replace('Sibi','Nila')
>>> print(v2)
Hello Nila
>>> v3=v2.replace('l','o')
Heooo Nila
```

Stripping Whitespace

- Sometimes we want to take a string and remove whitespace at the beginning and/or end.
- lstrip() and rstrip() to the left and right only
- strip() Removes both begin and ending whitespace

```
>>> greet = '   Hello SACOE  '
>>> greet.lstrip()
'Hello SACOE  '
>>> greet.rstrip()
'   Hello SACOE'
>>> greet.strip()
'Hello SACOE'
>>>
```

startswith() method & endswith() method

```
>>>var='Playing is good for health'
>>>var.startswith("P")
True
>>> var.startswith("p")
False

>>>name='Information technology'
>>>name.endswith('technology')
True
>>>name.endswith('Technology')
False
```

find() / rfind() Method

```
>>> var=“Welcome”
>>> var.find(“e")
1
>>> var.rfind(‘e’)
6
>>> var1=‘How are you’
>>> var1.find(‘are’)
4
```

count() Method

```
>>>var='welcome'
>>>var.count('e')
2
>>>var1='how are you? Are you there'
>>>var1.count('are')
1
>>>var1.count('you')
2
```

capitalize() Method

```
>>>var='welcome'
>>>var.capitalize()
Welcome
>>> var1='wELCOME'
>>>var1.capitalize()
Welcome
>>>var2='WELCOME'
>>>var2.capitalize()
Welcome
```

title() method

```
>>>var=‘hello how are you?’
>>>var.title()
Hello How Are You?

>>>var1=‘HELLO HOW ARE YOU?’
>>>var1.title()
Hello How Are You?
```

swapcase() method

```
>>>var='python IS A powerful language'
>>>var.swapcase()
PYTHON is a POWERFUL LANGUAGE

>>>var1='wELcOME'
>>>var1.swapcase()
WelCome
```

islower() / isupper() Methods

```
>>>var='python'
>>>var.islower()
True
>>>var.isupper()
False
>>>var1='PYTHON'
>>>var1.isupper()
True
```

istitle() method

```
>>>var='Welcome'
>>>var.istitle()
True

>>>var1='Welcome TO INDIA'
>>>var1.istitle()
False
```

isalpha() Method

```
>>>var1='STTP'
>>>var1.isalpha()
True

>>>var2='STTP on Python Programming'
>>>var2.isalpha()
False
```

isdigit() Method

```
>>>mobile=input('Enter your mobile no:')
9988776655
>>>mobile.isdigit()
True
>>>mobile=input('Enter your mobile no:')
>>>mobile2=999922113o
>>>mobile2.isdigit()
False
```

isalnum() method

```
>>>password='abcde1234zx'
>>>password.isalnum()
True

>>>pw='ab 12 zx 78'
>>>pw.isalnum()
False
```

Split() and Join Method

```
>>> str1="C C++ Java Python"
>>> lis=str1.split()
>>> lis
['C', 'C++', 'Java', 'Python']
>>>l1=str1.split('a')
>>> l1
['C C++ J', 'v', ' Python']
str2=' '.join(lis)
>>> str2
'C C++ Java Python'
```

String.format() Method()

```
>>> '{} plus {} equals{}'.format(4,5,'Nine')
 '4 plus 5 equalsNine‘
>>> '{1} plus {0} equals{2}'.format(4,5,'Nine')
'5 plus 4 equalsNine‘
>>> '{0} plus {1} equals {res}'.format(4,5,res='Nine')
'4 plus 5 equals Nine'
```

String Comparison

- Operators such as ==,<,>,<=,>= and != are used to compare the strings.
- Python compares strings by comparing their corresponding characters.

```
>>> S1="abcd"
>>> S2="ABCD“
>>> S1>S2
True
```

FUNCTIONS

- A function is a block of code which only runs when it is called.
- **function** is a block of program statements which can be used repetitively in a program.
- It saves the time of a developer.
- There are two basic types of functions:
    1. built-in functions and
    2. user defined functions

Built-in Functions

- **Print()**

```
print ("Hello, Python!")
```

- **Input()**

```
a=input("Enter First Number")
b=input("Enter Second Number")
print("The First Number is:",a)
print("The Second Number is:",b)
```

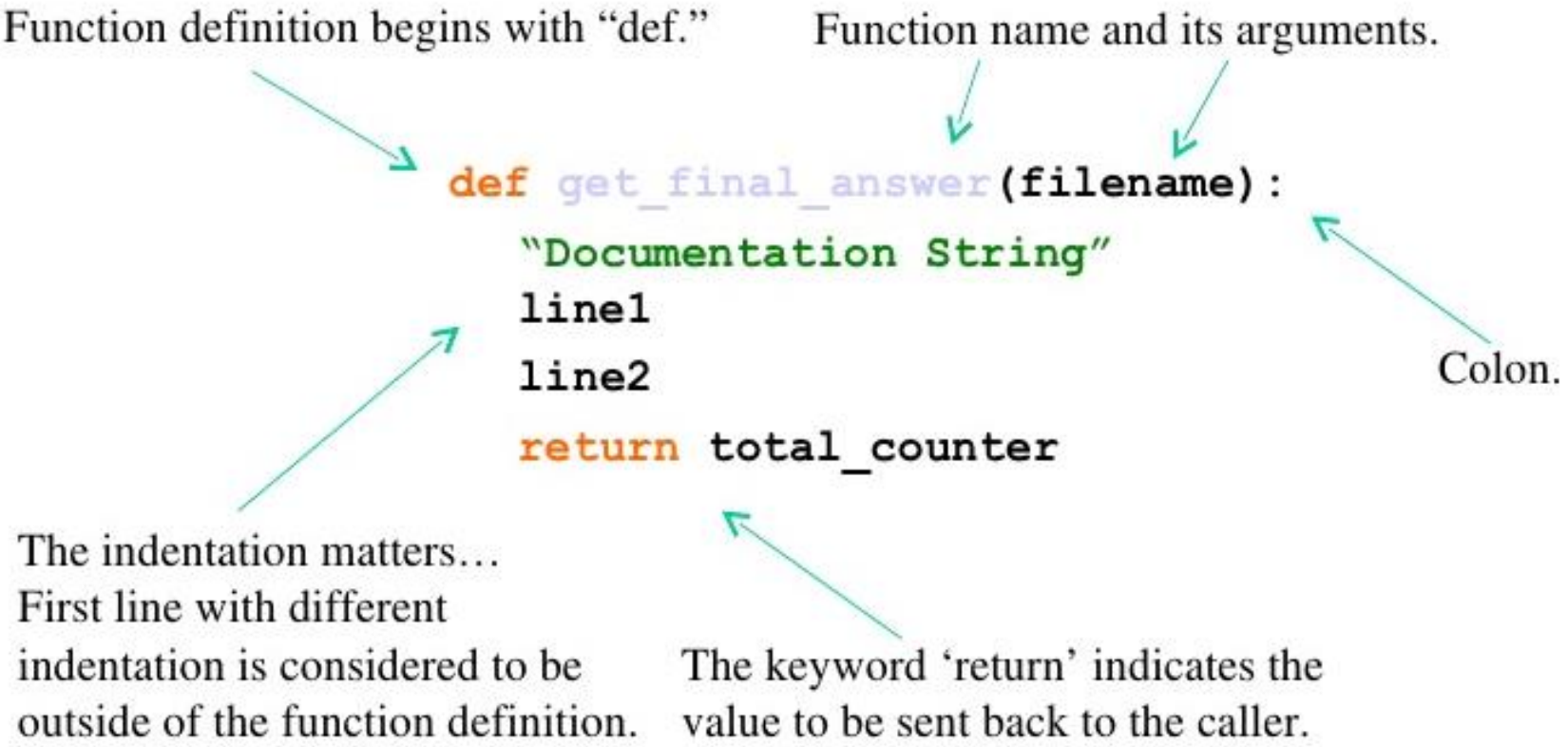
Built-in numeric functions - examples

```
>>> range (6, 18, 3)      # returns: [6, 9, 12, 15]
>>> range (10)      # returns: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> abs (-17)       # returns: 17

>>> max (12, 89)  # returns: 89

>>> round (12.7)   # returns: 13.0
```

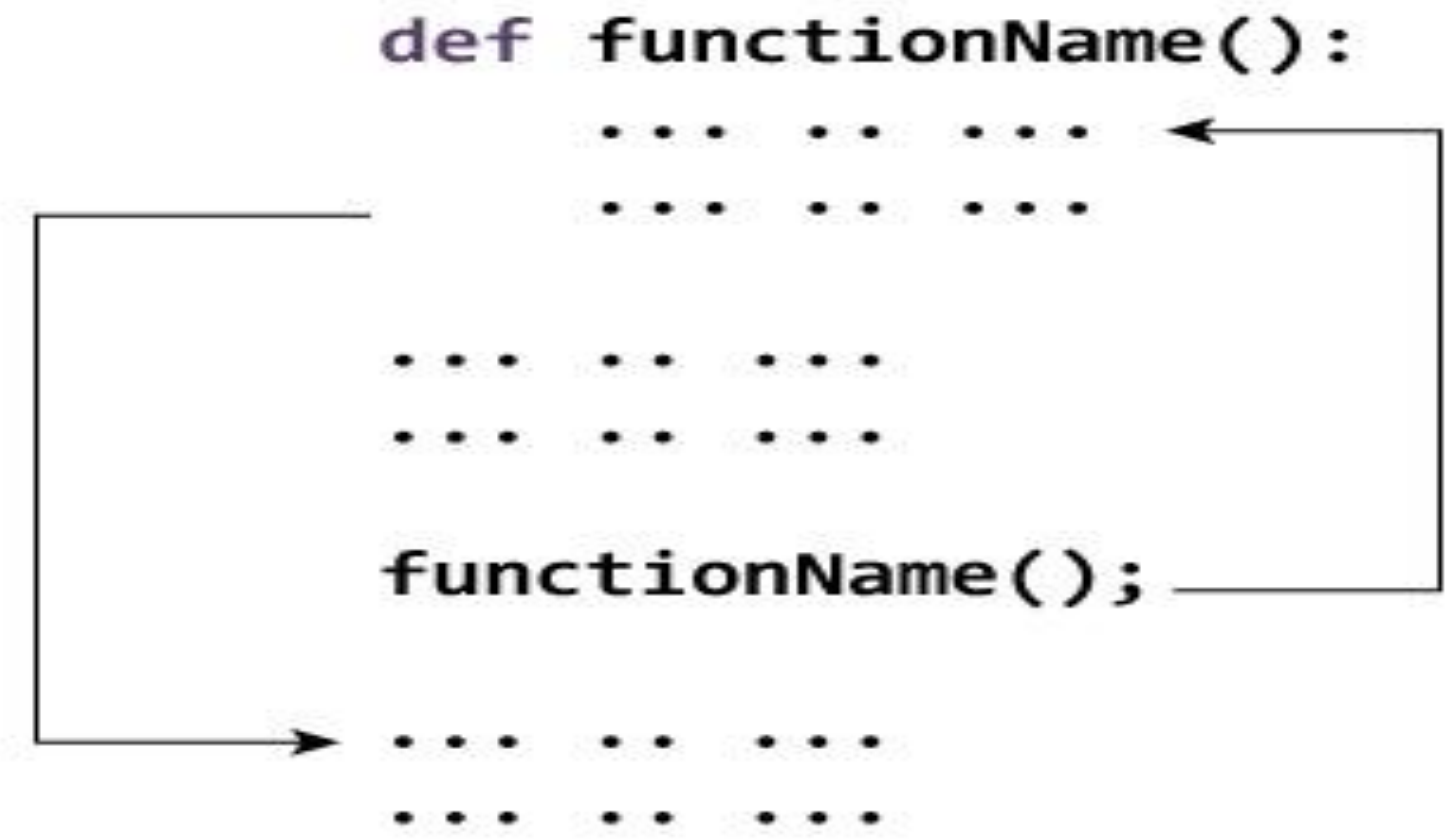
User Defined Functions

- You can also create your own functions.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.
- You can call the function using the function name.

Defining Functions



No header file or declaration of types of function or arguments.


def functionName():
... .
... .
... .
... .
functionName();
... .
... .

Example

```
def add(a,b):
    c=a+b
    return c
a=2
b=3
Sum1=add(a,b)
print (Sum1)
```

Return Statement

- The statement return [expression] exits a function.
- A return statement with no arguments is the same as return None.

```
def maximum(x, y):
  if x > y:
    return x
  else:
    return y


print maximum(2, 3)
```

Types of Arguments

- Required arguments
- Keyword arguments
- Default arguments
- Variable length arguments

Positional/Required arguments

- ✓ The arguments passed to a function in correct positional order.
- ✓ The number of arguments in the function call should match exactly with the function definition.
- ✓ Python will show an error when an incorrect number of arguments are passed to the function call.
- ✓ The arguments must match the parameters in order, number and type as defined in the function

```
# Function definition is here
def printinfo( name, age ):
   print ("Name: ", name)
   print ("Age ", age)
    return
# Now you can call printinfo function
printinfo( “Sibi”,5 )
```

OUTPUT:

Name: Sibi Age 5

Keyword arguments

- ✓ When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.
- ✓ A positional argument cannot follow a keyword argument.

```
def printinfo( name, age ):
   print ("Name: ", name)
   print ("Age ", age)
    return
printinfo( age = 7, name = “Nila" )
```

OUTPUT:

Name: Nila Age 7

#printinfo(age=7,"Nila") Error

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 printname(name,no=123):
        print(“Name:”,name)
        print(“Number:”,no)
        return;
printname(name=“raj”)
printname(name=“raju”,no=456)
```

Returning Multiple values from function

```
def compute(num1):
    print("Number=",num1)
    return num1*num1, num1*num1*num1
square, cube=compute(4)
print("Square=",square,"Cube=",cube)
```

Output:

Number= 4

Square= 16 Cube= 64

Variable length arguments

- ✓ If you need to process a function for more arguments than you specified while defining the function. We go for variable length arguments.

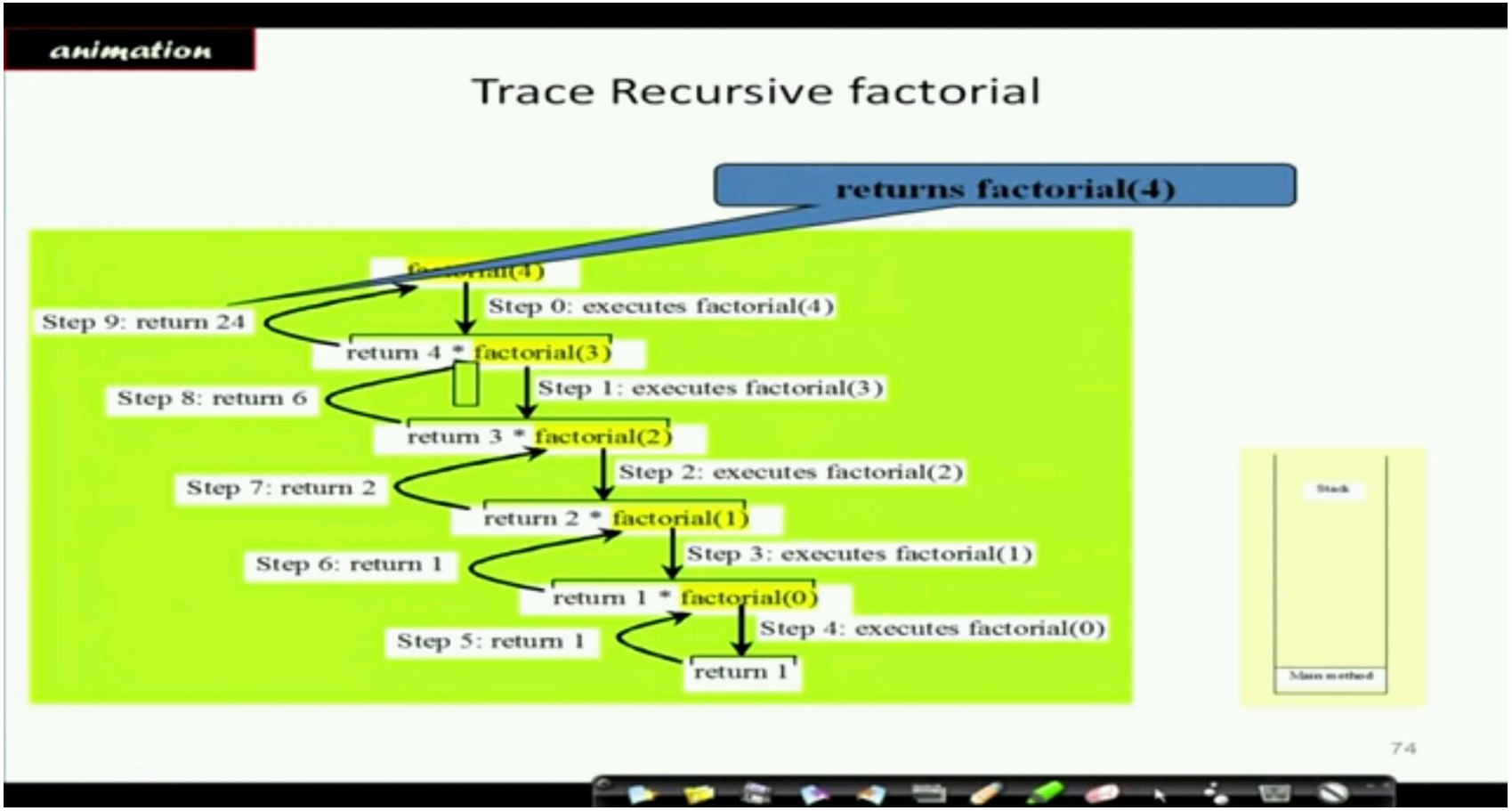
```
def info(arg1,*var_arg):
        print("Output is:")
        print(arg1)
        for var in var_arg:
                print(var)
        return;
info( 10)
info(10,20,30,40,50)
info('a','b','c')
```

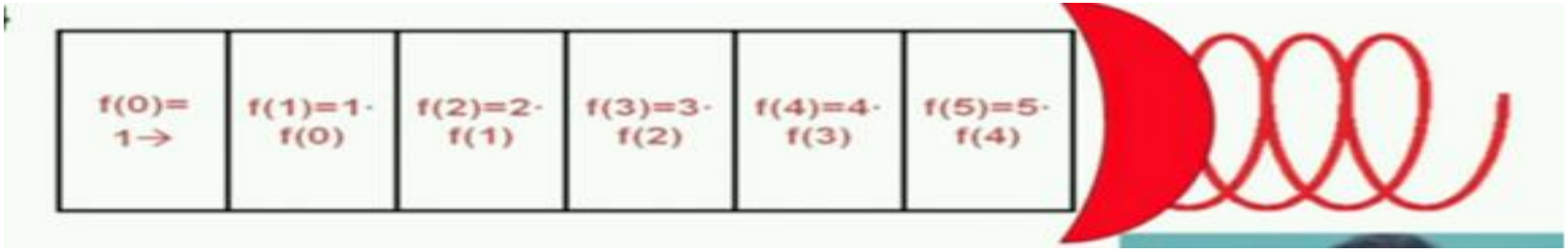
- The arbitrary number of arguments passed to the function basically forms a tuple before being passed into the function.
- The Variable-length arguments if present in the function should be the last in the list of formal parameters.
- Any formal parameters written after the variable-length arguments must be keyword-only arguments.

Recursion

- function calls itself one or more times in its body.

```
def factorial(n):
    if n == 0:
        return 1
    else:
        res=n*factorial(n-1)
        return res
f=int(input("Enter a number:"))
result=factorial(f)
print("The result is:",result)
```


animation
Trace Recursive factorial
returns factorial(4)
Step 9: return 24
Step 0: executes factorial(4)
return 4 * factorial(3)
Step 8: return 6
Step 1: executes factorial(3)
return 3 * factorial(2)
Step 7: return 2
Step 2: executes factorial(2)
return 2 * factorial(1)
Step 6: return 1
Step 3: executes factorial(1)
return 1 * factorial(0)
Step 5: return 1
Step 4: executes factorial(0)
return 1
Stack
Main method
74


f(0)=
1→
f(1)=1·
f(0)
f(2)=2·
f(1)
f(3)=3·
f(2)
f(4)=4·
f(3)
f(5)=5·
f(4)

Lambda Function (Anonymous function)

Syntax:

Name=lambda(variables):code

```
#cube of a number
cube=lambda x: x*x*x
print(cube(2))
```

Output

8

```
#Using Function
def func(x):
    return x*x*x
Print(func(3))
```

Output:

27

Scope of Variables

- Where you can access the variable.
- Two basic scopes:
  - Global Scope
  - Local Scope
- Inside a function(Local) – can be accessed only inside the function.
- Outside a function(Global)- can be accessed throughout the program.

```
x=0
def outer():
    def inner():
        x=2 #Local
        print("inner:", x)
    inner()
    print("outer:",x)
outer()
print("global:",x)
```

OUTPUT:

inner: 2

outer: 0

global: 0

THANK YOU ALL

Queries?