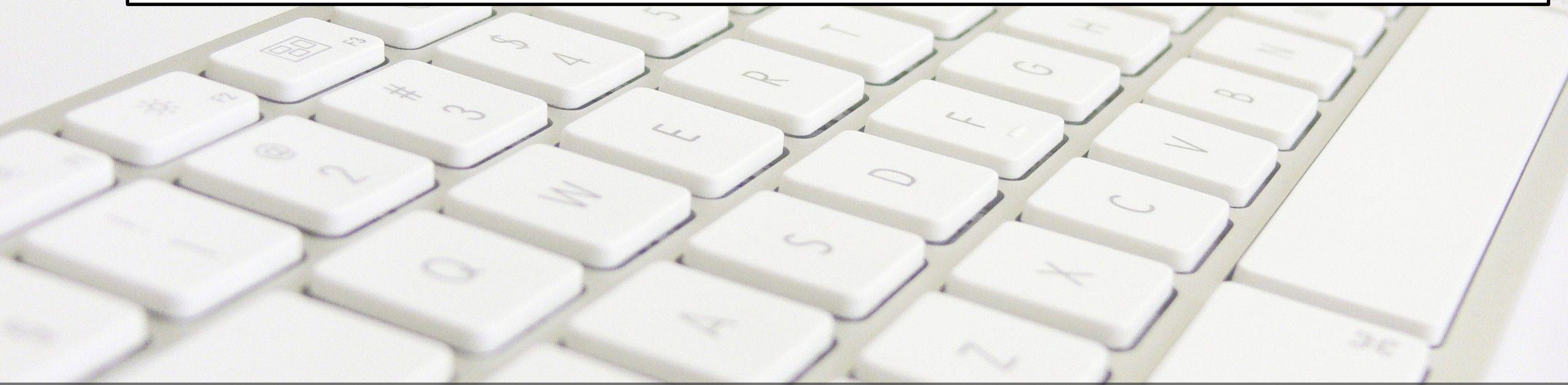
4.Functions and Module



General Guideline

Objective of Functions and Modules

- To describe the importance of functions in Python
- To explain the function definition in Python
- To develop python programs with functions
- To use the void functions and return statements
- To explain the difference between different function argument types and use them
- To select built-in functions in Python to write programs in Python.

Topics Covered

Day 1	Day 2	Day 3	Day 4
4.1 Introduction to Functions • 4.1.1 Arguments and parameters • 4.1.2 Return statement	4.1 Introduction to Functions • 4.1.3 Types of Function arguments • 4.1.4 Scope and Lifetime of variables	4.2 Anonymous and Higher Order Function • 4.2.1 Anonymous Function	4.2 Anonymous and Higher Order Function • 4.2.2 First Class Function and Higher Order Function

Topics Covered

Day 5	Day 6		
4.3 Modules • 4.3.1 Creation of module • 4.3.2 Importing module • 4.3.3 Standard Built-In Module	4.4 Iterative Built-in Functions		

Session Plan - Day 1

4.1 Introduction to functions

4.1.1 Argument and parameters

4.1.2 Return statement

4.1.3 Types of function arguments

4.1.4 Scope and lifetime of variables.

Introduction

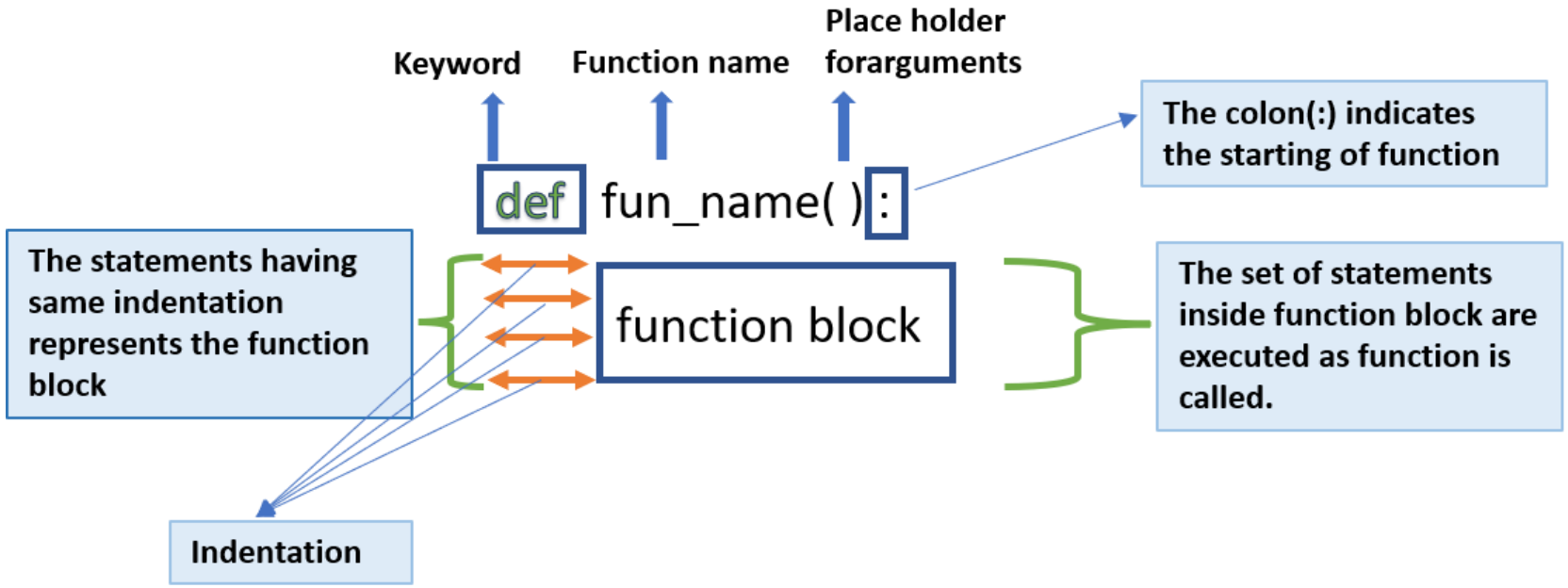
A large program is divided into small blocks of code called **functions**, to:

- Increase the reusability of code.
- Ease of programming.

Function is a group of all statement(activities) that perform a specific task.

Syntax

The syntax of **function definition** is:



Benefits of Using Functions:

- Avoid duplication of the similar type of codes
- Increases Program readability and Improved Concept Clarity
- Divide the bigger problem into smaller chunks
- Reusability of code

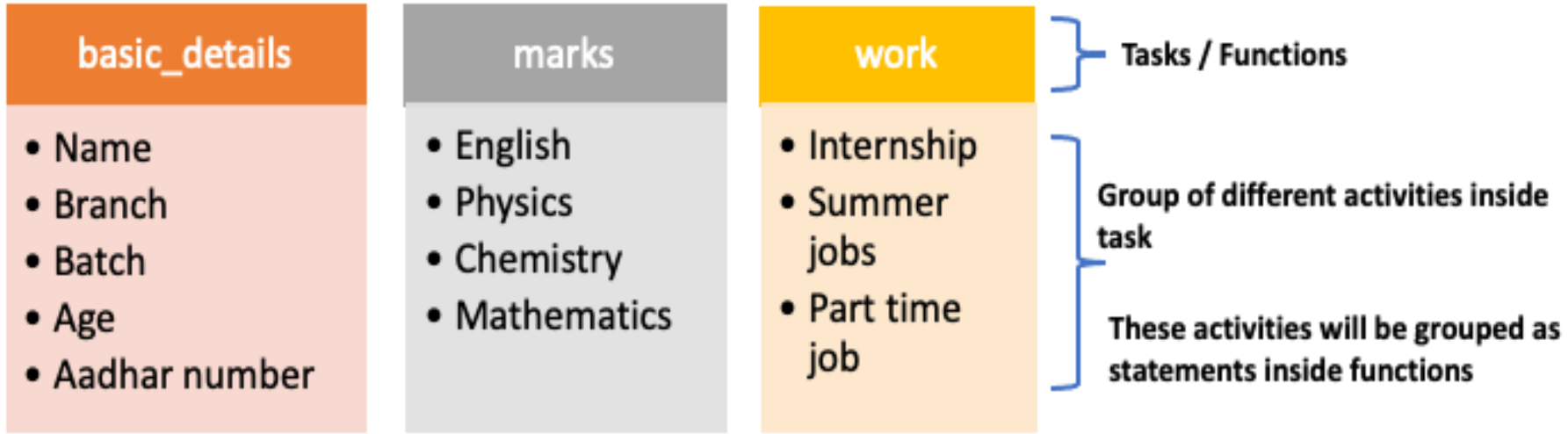
Example

In **real world scenario**, we need to group all the activities related to a specific task, like:

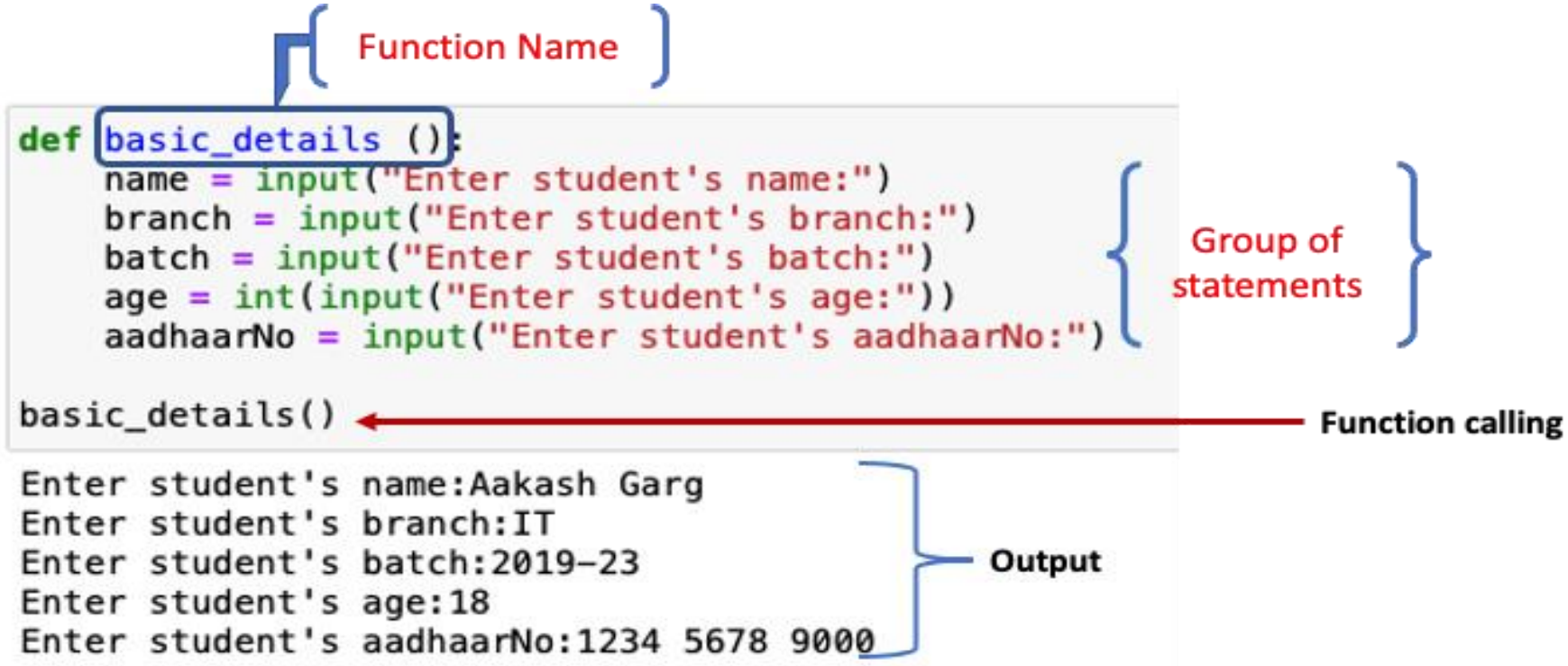
1. A task of maintaining student's basic details involves a group of different activities like: ***Collecting their name, branch, batch, age, aadhar number, etc.***
2. Similarly, the task of maintaining marks of students involves a group of different activities like: ***Collecting marks of English, Physics, Chemistry, Mathematics, etc.***

Contd..

- We can create three functions like:
  - basic_details ()
  - marks()
  - work()

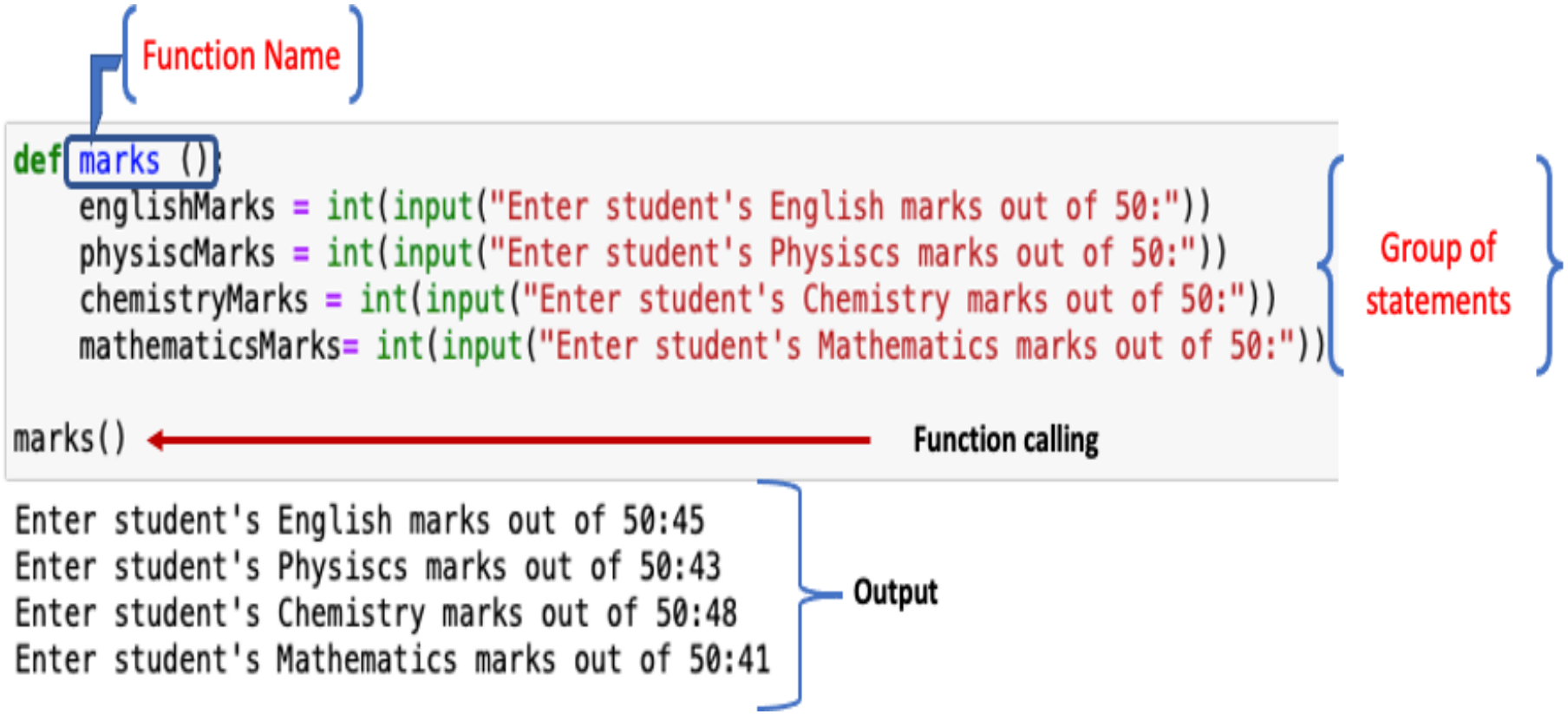


Function 1: basic_details ()

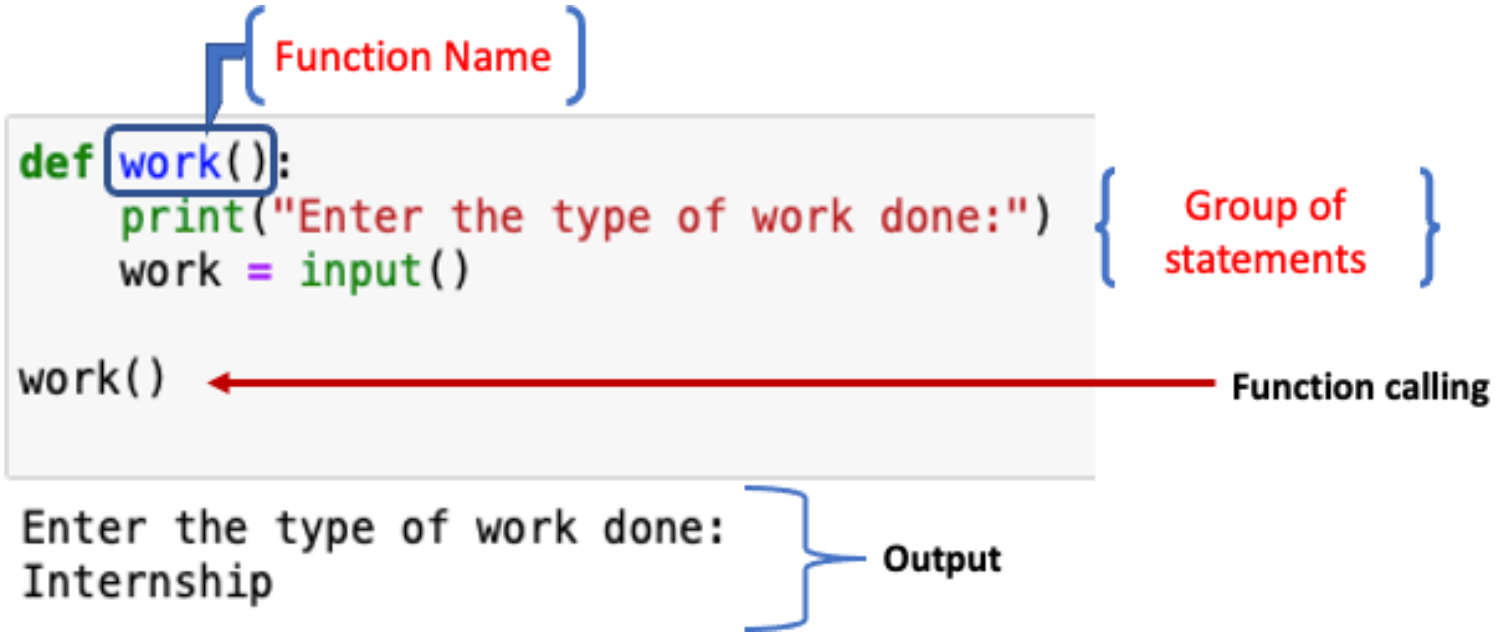


Contd..

Function 2: marks ()



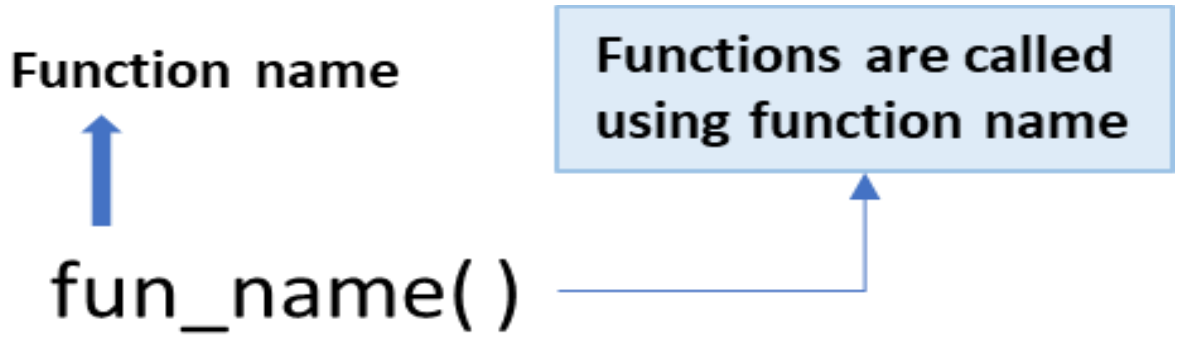
Function 3: work ()



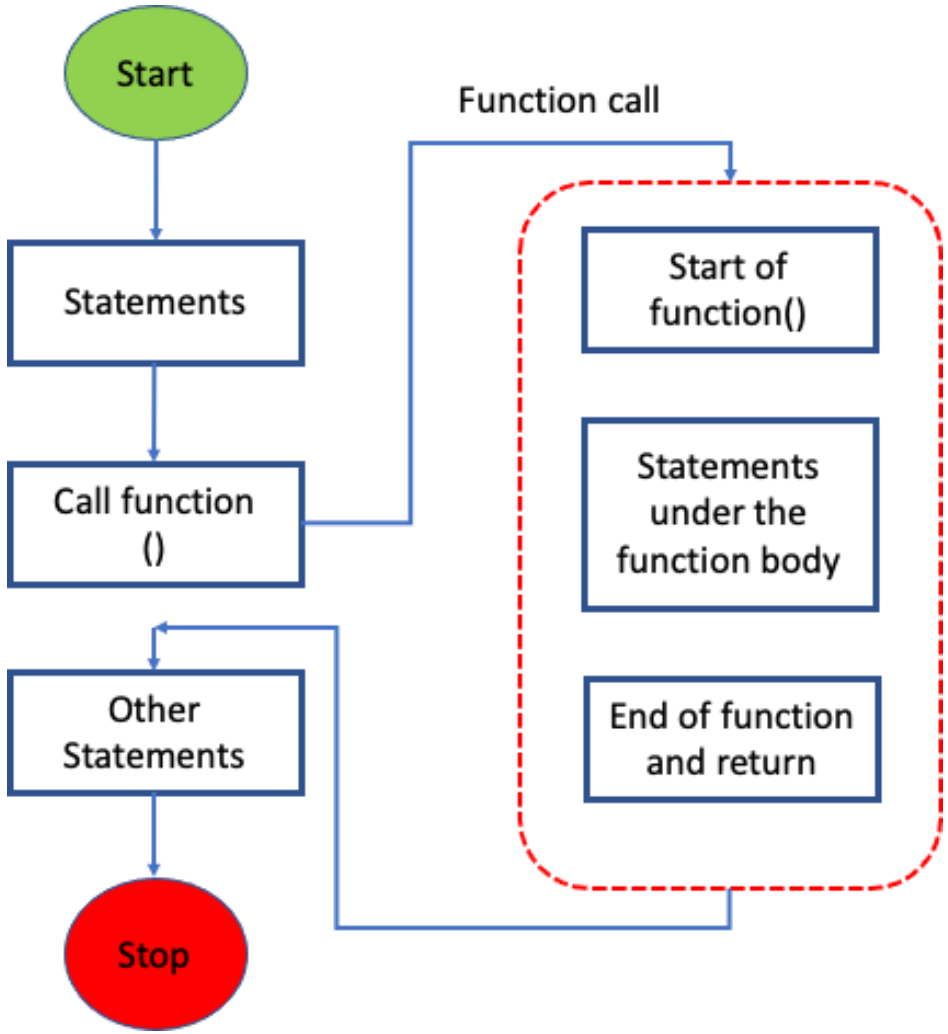
Function Call

A function is called using function name through calling environment.

The syntax of function call is:



Flow chart



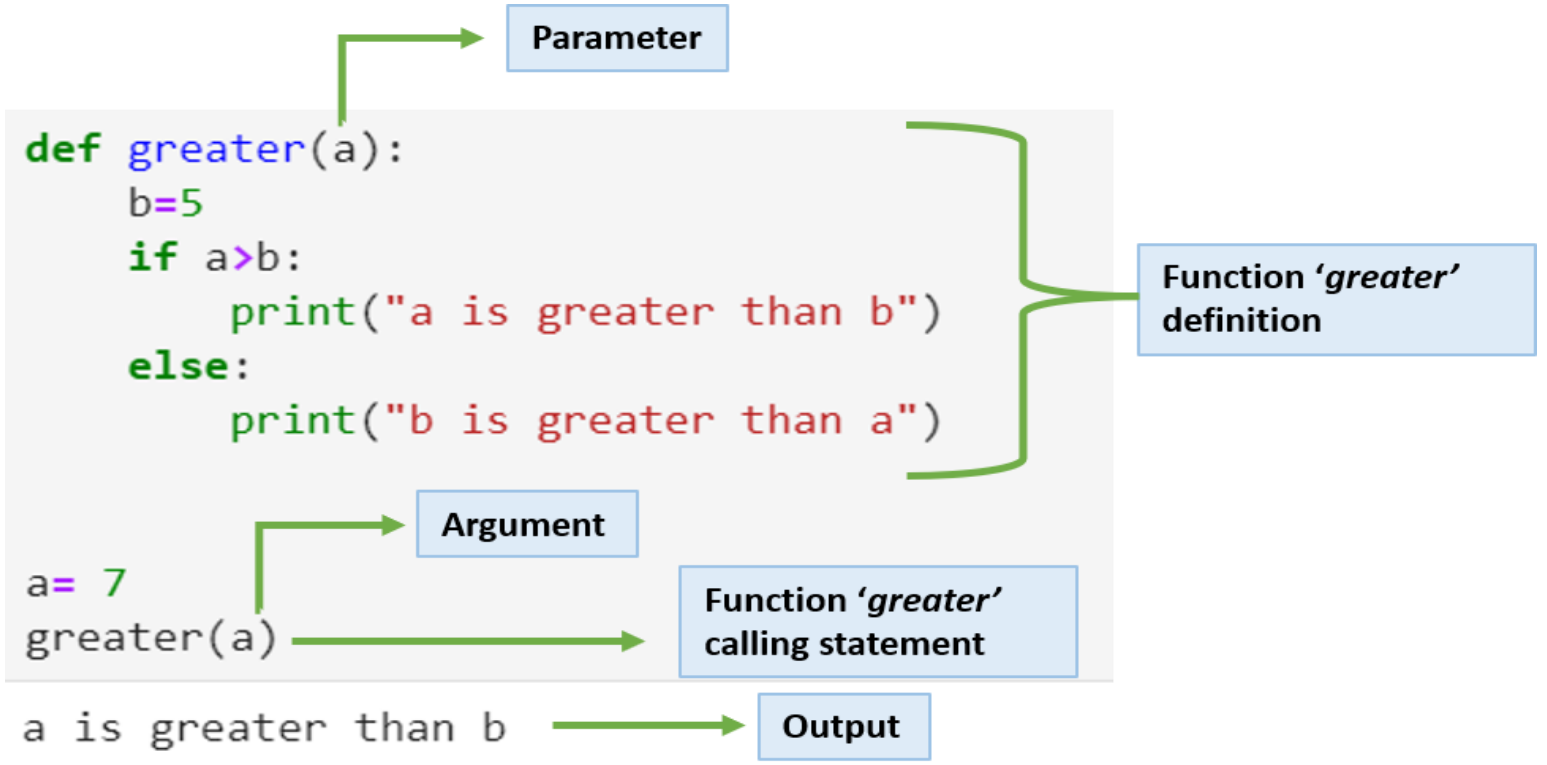
Arguments and parameters

- Arguments are used to pass the information to the functions.
- They are mentioned after function name inside the parentheses in function calling statement.
- Any number of arguments can be added by separating them by commas.

Note: The number of parameters in function definition must be same as the number of arguments in function calling statement.

Example 1

Write a python program to determine the greater number among two numbers using function *greater(a)*, by passing one number as argument in function calling.



Example 2

Write a python program to determine the greater number among two numbers by passing both the number as argument in function calling.

Two parameters *'x'* and *'y'*. The x will be equal to a and y will be equal to b.

```
def greater(x, y):
    if x>y:
        print("x is greater than y")
    else:
        print("y is greater than x")

a= 7
b=10
greater(a, b)
```

Two arguments *'a'* and *'y'*

```
y is greater than x
```

Output

Two parameters *'x'* and *'y'*. The x will be equal to 7 and y will be equal to 10.

```
def greater(x, y):
    if x>y:
        print("x is greater than y")
    else:
        print("y is greater than x")

greater(7, 10)
```

Two arguments *'7'* and *'10'*

```
y is greater than x
```

Output

Return statement

- Return statement indicate the end of function execution.
- It is also used to return the values or results to the function calling statement.

Example 1

Write a python program to demonstrate the return statement of function for the addition of two numbers.

```
def addition (a, b):
    sum = a + b
    return sum

x = addition(9, 10)
print("The value of two number addition = ", x)
```

Return statement of function '*addition*'

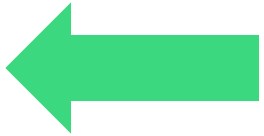
Value returned by function is assigned to variable x, of function calling statement.

```
The value of two number addition =  19
```

Output

Review Questions

1. Which keyword is used for function?
   a) Fun
   b) Define
   c) Def
   d) Function



2. What will be the output of the following Python code?

```
def f(x, y, z): return x + y + z
f(2, 30, 400)
```

a) 432

b) 24000

c) 430

d) No output

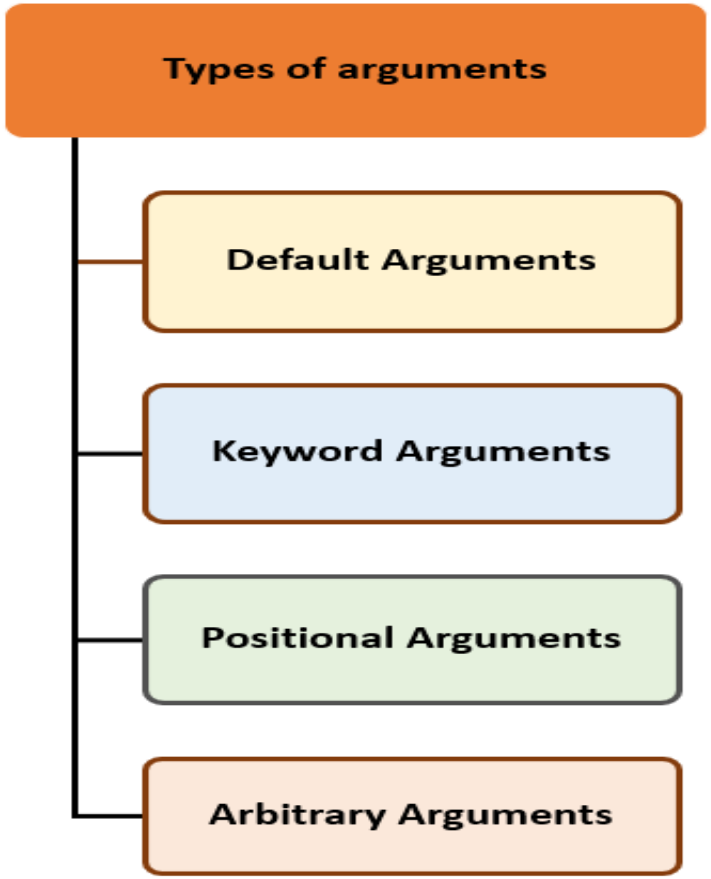
Session Plan - Day 2

4.1 Introduction to functions

- 4.1.3 Types of Function arguments
- 4.1.4 Scope and Lifetime of variables

Types of Function arguments

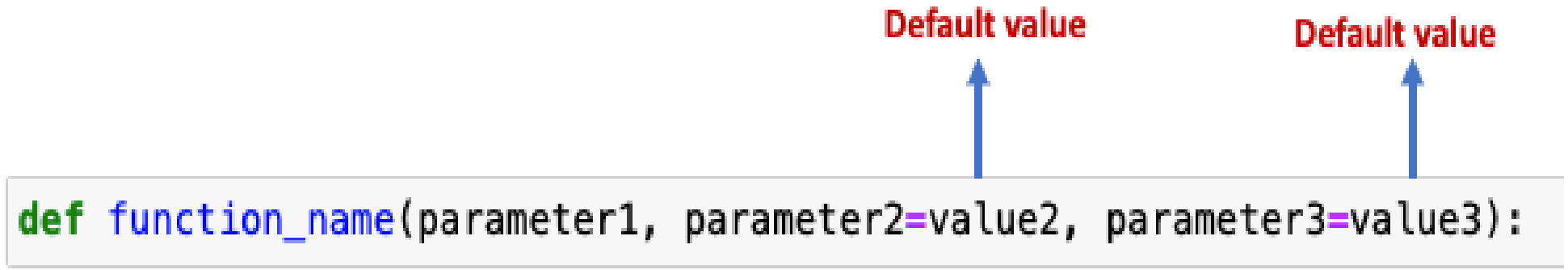
There are five types of arguments in Python function:



Default arguments :

Default arguments are values that are provided in the function definition.

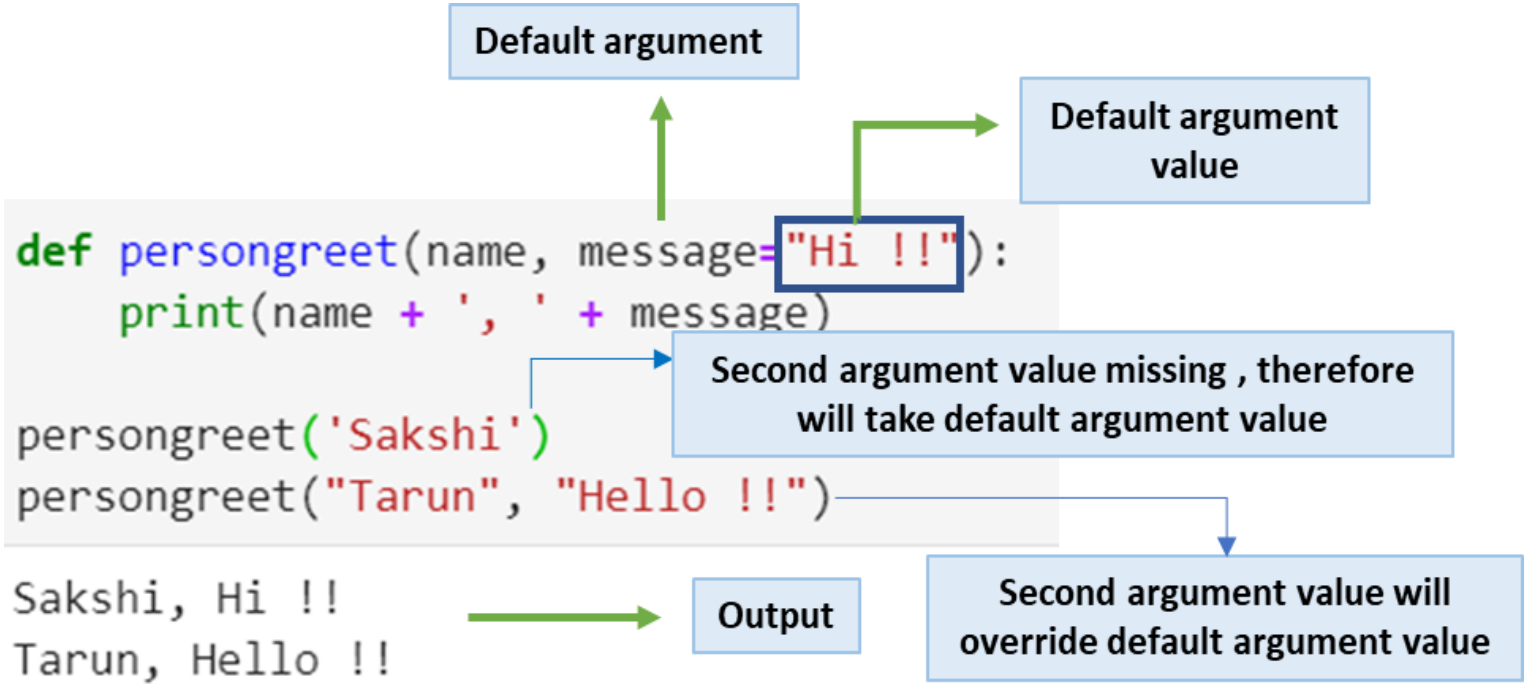
Syntax:



Note: Function can have any number of default arguments. If the values are provided to the default arguments during the function calling, it overrides the default value.

Example 1

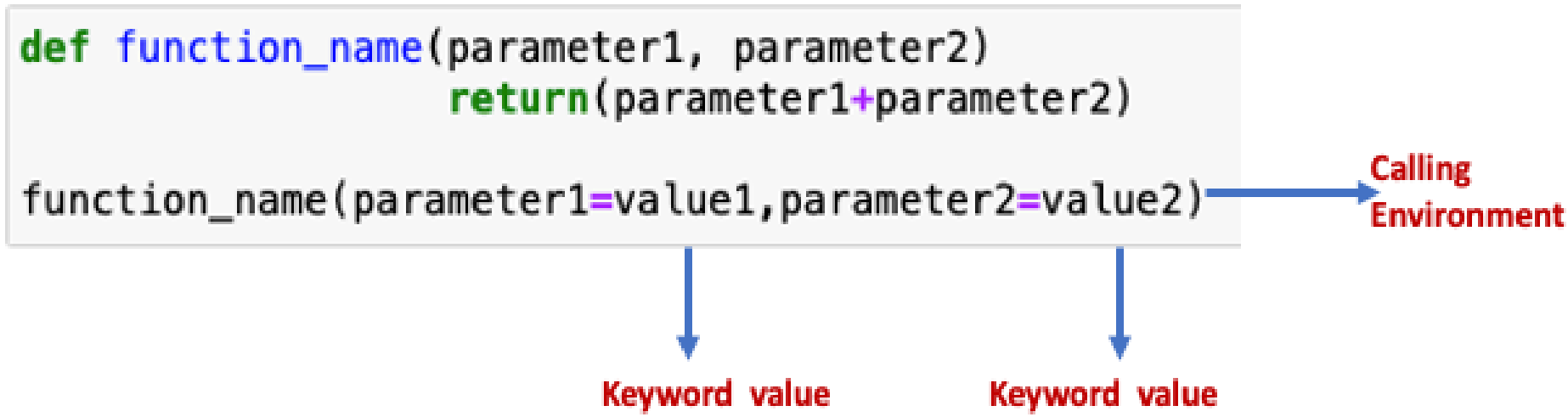
Write a python program to greet a person with person name and message provided in function calling statement. If message not provided in function call, then print a default message “Hi !!”.



Keyword arguments :

Default arguments are keyword arguments whose values are assigned at the time of function definition.

Syntax:



Example 1

Write a program to demonstrate keyword arguments.

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

add(b=10,c=15,a=20)
```

b, c, a are keyword arguments

```
45
```

Output

```
def add(a=20,b,c):
    return(a+b+c)

add(b=10,c=15,a)
```

An **error** because it uses the default argument after a keyword argument

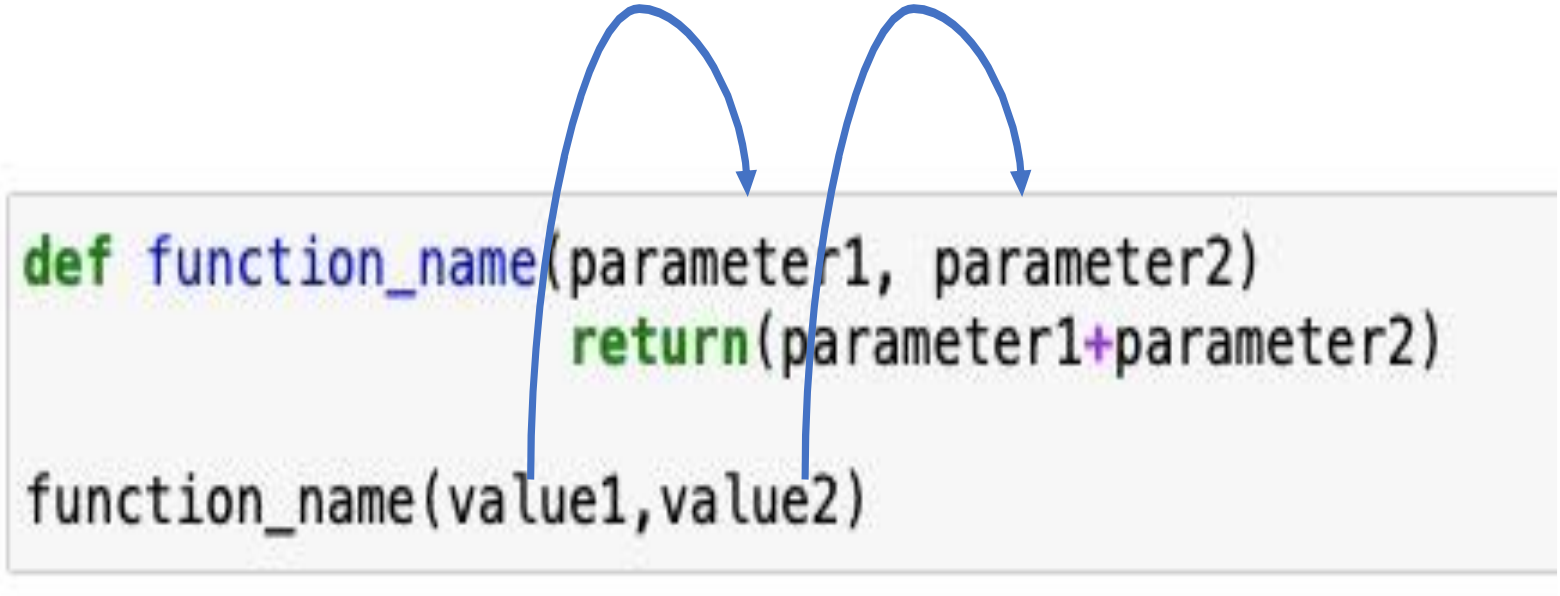
```
  File "<ipython-input-10-8a31cfeb86a3>", line 1
    def add(a=20,b,c):
                  ^
SyntaxError: non-default argument follows default argument
```

Output

Positional arguments :

Positional arguments are arguments that need to be included in the proper position or order while calling the function.

Syntax:



Example 1

Write a program to demonstrate positional arguments.

```
def add(a,b,c):
    return (a+b+c)
print (add(10,20,30))
```

10,20,30 are the values for Positional Arguments a,b,c

```
60
```

Output

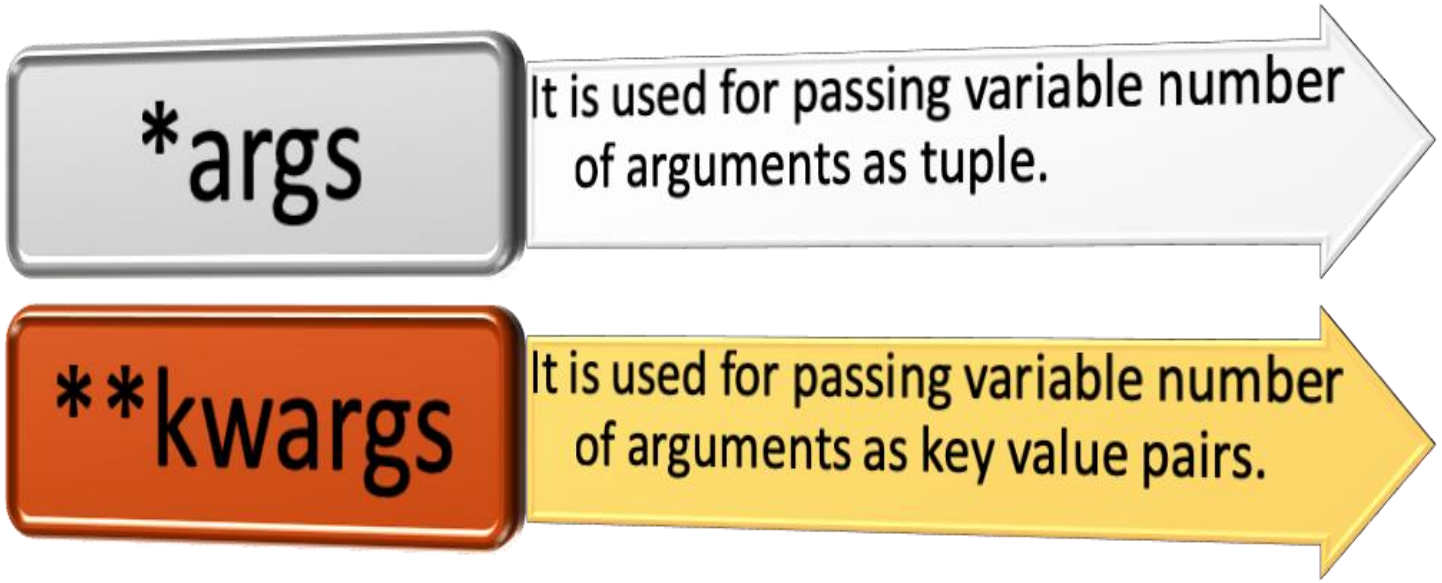
Arbitrary Arguments

- When we are not sure in the advance that how many arguments our function would require.
- This kind of situation is handled through function calls with an arbitrary number of arguments.

Note: We define the arbitrary arguments while defining a function using the asterisk (*).

Types of Arbitrary Arguments

There are **two** types of arbitrary arguments as illustrated in the given figure below.



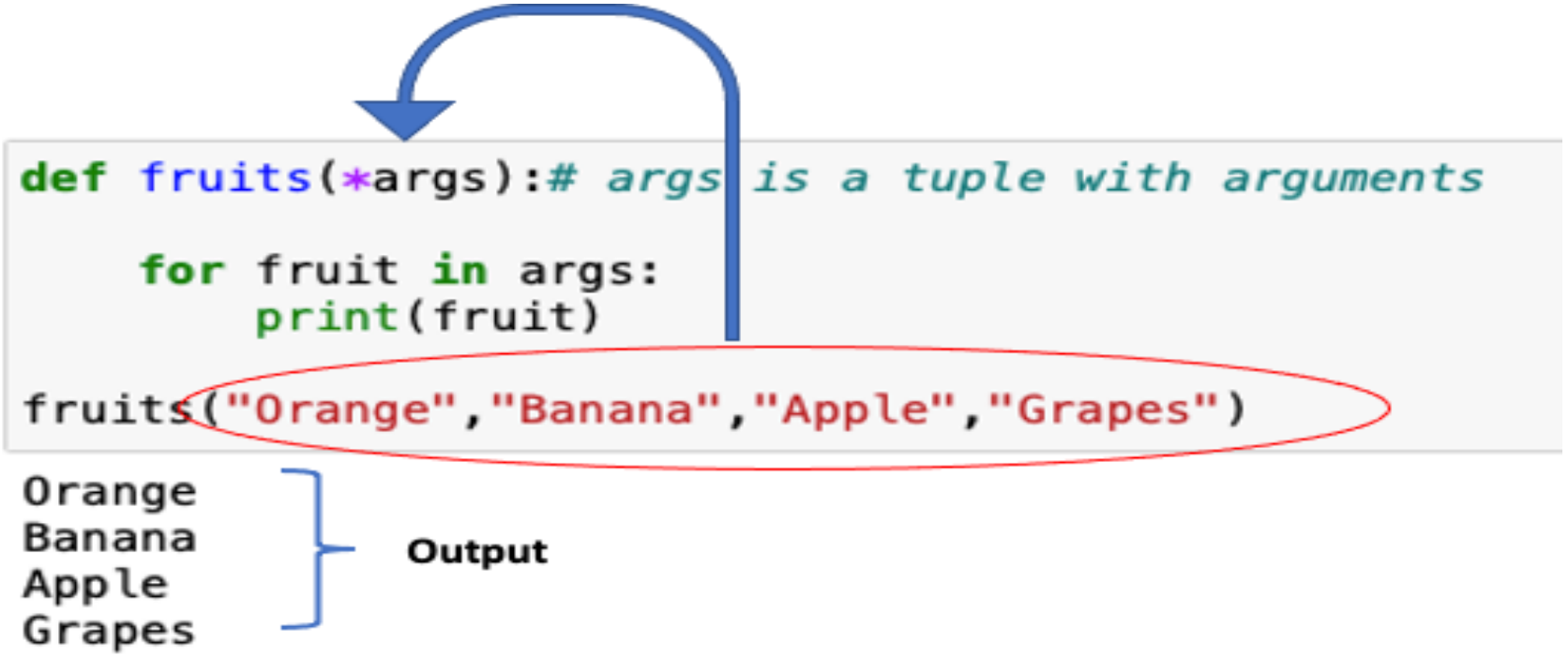
*args

The special syntax *args in function definitions is used to pass the variable number of arguments to a function.

Note: Generally, * args is used by convention but you can take any variable for example ***abc** will work in same way as * args.

Example 1

Write a program to demonstrate *args arbitrary arguments.



**kwargs

The **kwargs function in python is used to pass an arbitrary number of keyword arguments called kwargs.

Note: ** (Double star allows us to pass variable number of keyword arguments). This turns the identifier-keyword pairs into a dictionary within the function body.

**kwargs

Write a program to demonstrate **kwargs arbitrary arguments.

```
def fruits(**kwargs):# kwargs is a dictionary with key value pairs

    print(kwargs)
    for fruit in kwargs.values()
        print(fruit)

fruits(fruit1= "Orange",fruit2= "Banana",fruit3= "Apple",fruit4= "Grapes")
```

```
{'fruit1': 'Orange', 'fruit2': 'Banana', 'fruit3': 'Apple', 'fruit4': 'Grapes'}
Orange
Banana
Apple
Grapes
```

Output

**kwargs

Write a program using this function to generate perfect numbers from 1 to 1000. [An integer number is said to be “perfect number” if its factors, including 1(but not the number itself), sum to the number. E.g., 6 is a perfect number because 6=1+2+3].

```
def perfect_num(n):
    sum = 0
    for i in range(1,n):
        if n%i == 0:
            sum = sum + i
    if(sum == n):
        return True
    else:
        return False
for i in range(1,1001):
    if(perfect_num(i)):
        print(i)
```

Function returning true or false based on perfect number condition

```
6
28
496
```

Output

Scope and Lifetime of variables

- Let's consider a situation that you book a train ticket in sleeper coach and you got berth in S1 coach and 48 seat number.
- Now your ticket is valid for mentioned train and coach only this is called **scope** of your ticket.
- Your ticket would expire after mentioned journey date and that is called **lifetime**.

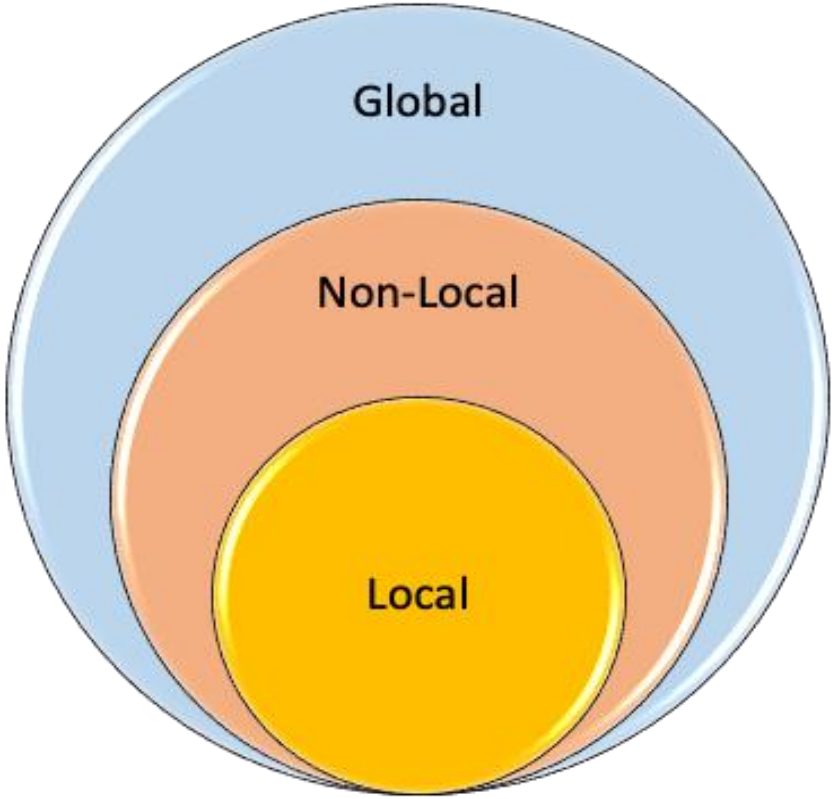
Scope and Lifetime of variables

- **Scope:** The scope of a variable determines its **accessibility and availability** in different portions of a program. Their availability depends on where they are defined.
- **Lifetime:** The lifetime of a variable is the time during which the variable **stays in memory**.

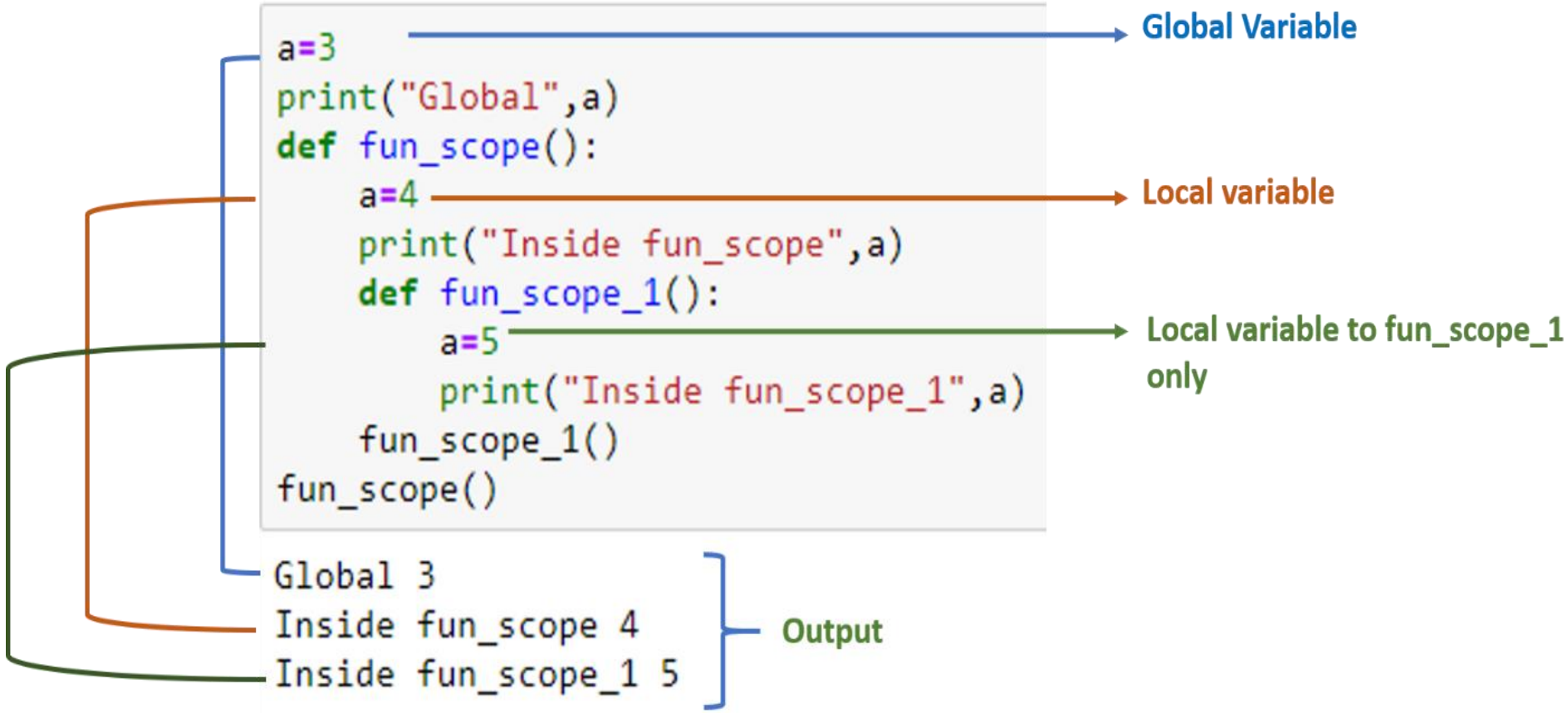
Types of variables

Three kinds of variables in Python:

- Global Variable
- Local Variables
- Non local variables



Demonstration of Local and Global



Global Variables

Variables that are declared outside of a function are known as global variables.

Example: Write a program to create global variable in python.

```
global_varible = "ABES Engineering College"

def welcome():
    print("Welcome to :",global_varible)

welcome()
```

Global variable declaration

Function Calling

```
Welcome to : ABES Engineering College
```

Output

Local Variables

- These are variables which are defined inside a function.
- Their scope is only to that function.
- They cannot be accessed from the outside of the function.

Example: Write a program to create local variable in python.

```
def welcome():
    local_varible = "ABES Engineering College"
    print("Welcome to :",local_varible)

welcome()
```

Local variable declaration

Function Calling

```
Welcome to : ABES Engineering College
```

Output

Example: Local and global variable in one program.

```
college_name = "ABES Engineering College"
def welcome():
    college_name = "ABES Engineering College,Ghaziabad"
    print("Welcome to :",college_name)

welcome()
print(college_name)
```

Global variable declaration

local variable declaration

Function will print local variable

Function will print global variable

```
Welcome to : ABES Engineering College,Ghaziabad
ABES Engineering College
```

Output

Example: To modify the variable outside of the current scope.

```
college_name = "ABES Engineering College"
def welcome():
    college_name = "ABES Engineering College,Ghaziabad"
    print("Welcome to :",college_name)

welcome()
print(college_name)
```

Global variable declaration

local variable declaration

Function will print local variable

Function will print global variable

```
Welcome to : ABES Engineering College,Ghaziabad
ABES Engineering College
```

Output

Non - Local Variables

- Nonlocal variables are used in nested functions.
- When a variable is either in local or global scope, it is called a nonlocal variable.

Note: Nonlocal keyword will prevent the variable from trying to bind locally first, and force it to go a level 'higher up'.

Non - Local Variables

- Nonlocal variables are used in nested functions.
- When a variable is either in local or global scope, it is called a nonlocal variable.

Note: Nonlocal keyword will prevent the variable from trying to bind locally first, and force it to go a level 'higher up'.

Example 1

A program to create local variables for nested functions.

```
def welcome():
    college_name = "ABES Engineering College"

    def welcome_again():
        college_name = "ABES Engineering College,Ghaziabad"
        university_name="AKTU"
        print("Welcome to :",university_name)

    welcome_again()
    print("Welcome to :",college_name)

welcome()
```

Local variable for welcome()

local variable for welcome_again()

Function will print local variable college_name from welcome()

```
Welcome to : AKTU
Welcome to : ABES Engineering College
```

Output

Example 2

A program to use nonlocal keyword for nested functions.

```
def welcome():
    college_name = "ABES Engineering College"

    def welcome_again():

        nonlocal college_name
        college_name = "ABES Engineering College,Ghaziabad"
        university_name="AKTU"
        print("Welcome to :",university_name)

    welcome_again()
    print("Welcome to :",college_name)

welcome()
```

Local variable for welcome()

{Keyword}

local variable for welcome_again()

Function will print local variable college_name from welcome_again()

```
Welcome to : AKTU
Welcome to : ABES Engineering College,Ghaziabad
```

Output

Review Questions

What will be the output of the following Python code?

```
def cube(x):
    return x * x * x
x = cube(3)
print(x)
```

a) 9
b) 3
c) 27
d) 30

What will be the output of the following Python code?

```
def power(x, y=2):
    r = 1
    for i in range(y):
        r = r * x
    return r
print(power(3))
print(power(3, 3))
```

a) 9
   27

b) 27
   9

c) None of the above

d) Both of the above

Session Plan - Day 3

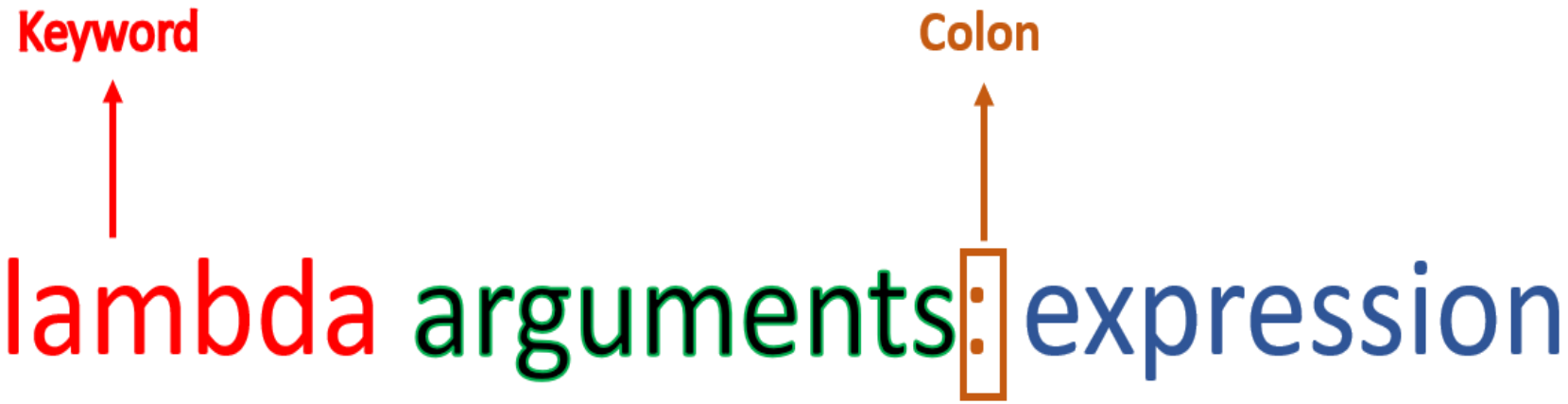
4.2 Anonymous and Higher Order Function

4.2.1 Anonymous Function
4.2.1 Lambda Function

Anonymous Function

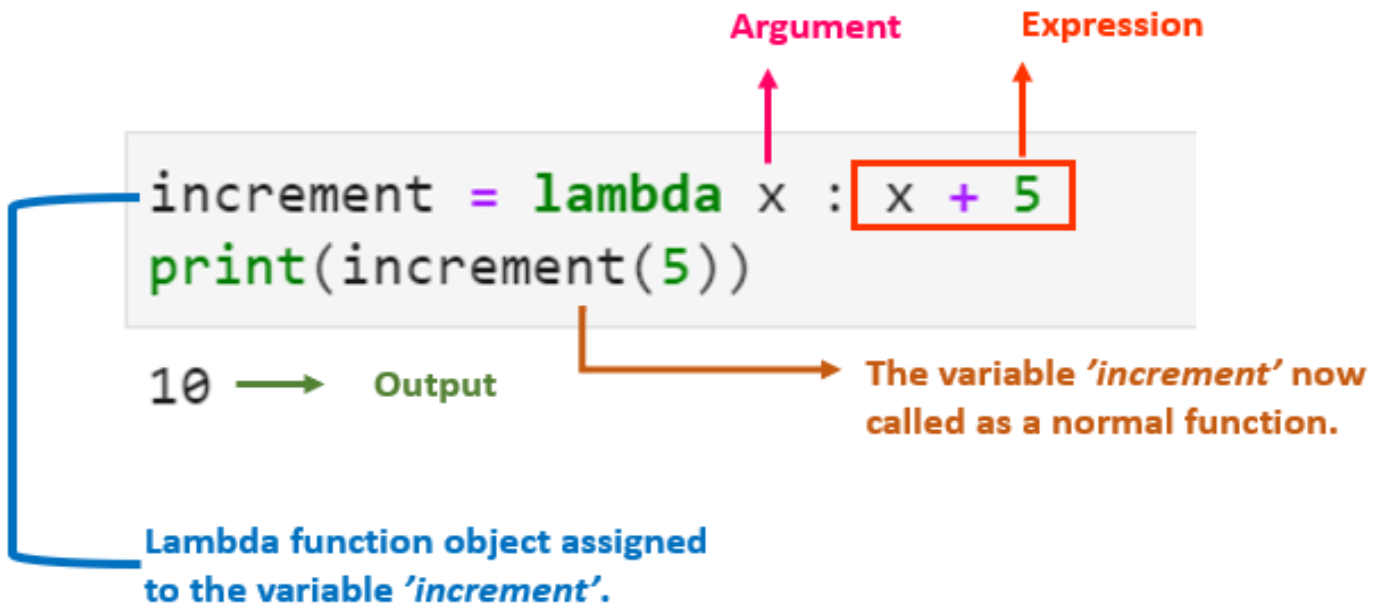
- Sometimes we only have single expression to evaluate, so rather than creating regular function we create anonymous Function.
- These functions do not have return statement, because by default evaluated single expression is returned.
- This is one liner code and have following properties –
  - Function without name
  - One expression statement but any number of arguments.

Syntax



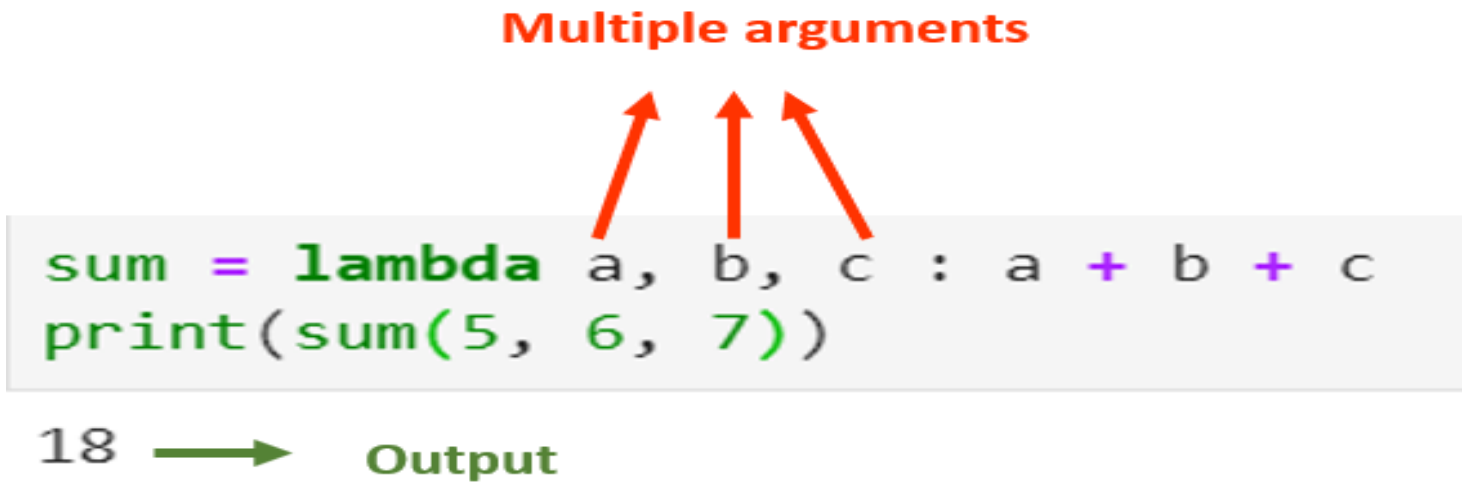
Scenario 1

Suppose we want to increment the value of variable by five, without writing lengthy code we can use lambda function.



Example 1

A program to add three values, using multiple arguments in lambda function.



Example 2

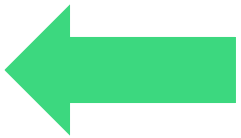
A program using lambda function to calculate the simple interest.

```
SimpleInterest = lambda P, R, T: P * R * T
print(SimpleInterest(400, 6, 2 ))
```

4800 → Output

Review Questions

1. Python supports the creation of anonymous functions at runtime, using a construct called _____

a) lambda 
b) pi
c) anonymous
d) none of the mentioned

2. What will be the output of the following Python code?

```
y = 6
z = lambda x: x * y
print z(8)
```

a) 48

b) 14

c) 64

d) None of the mentioned

Session Plan - Day 4

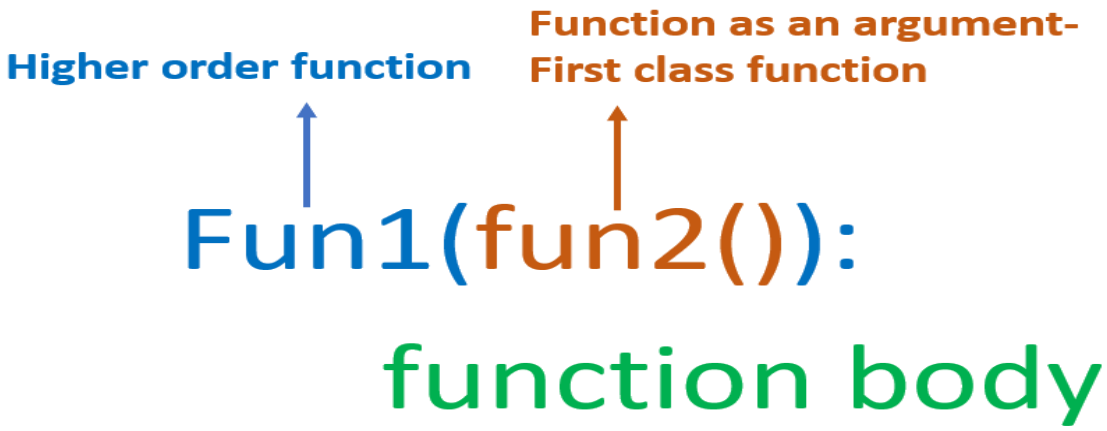
4.2 Anonymous and Higher Order Function

- 4.2.2 First Class Function and
- 4.2.2 Higher Order Function

First Class Function and Higher Order Function

- In first class function, function which can be passed as an argument.
- Functions which take first class function as an argument are called Higher order function.

Syntax:



Properties of First-Class Function:

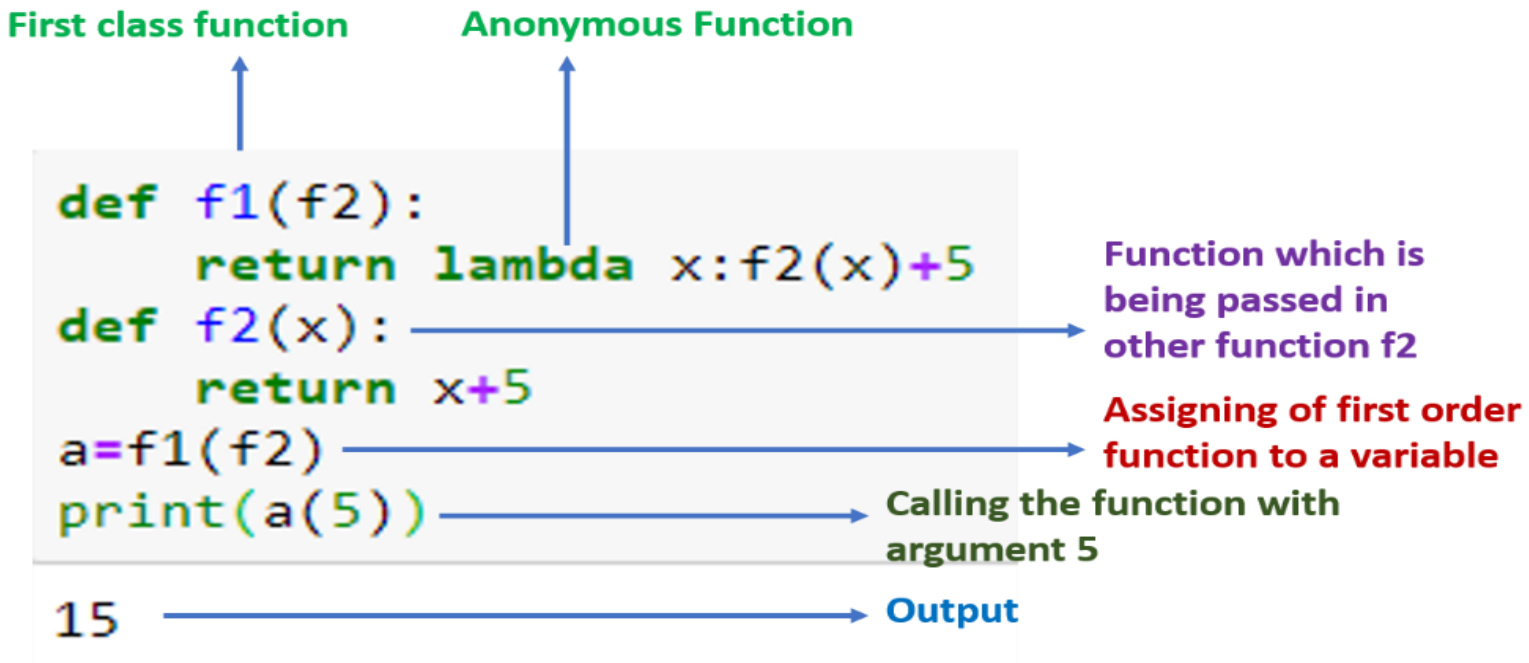
- Function can be passed as an argument into another function.
- Function can be assigned to a variable.
- Function can be returned from other function.

Properties of High Order Function:

- Function can take other function as an argument.
- Function can return other function.

Example 1

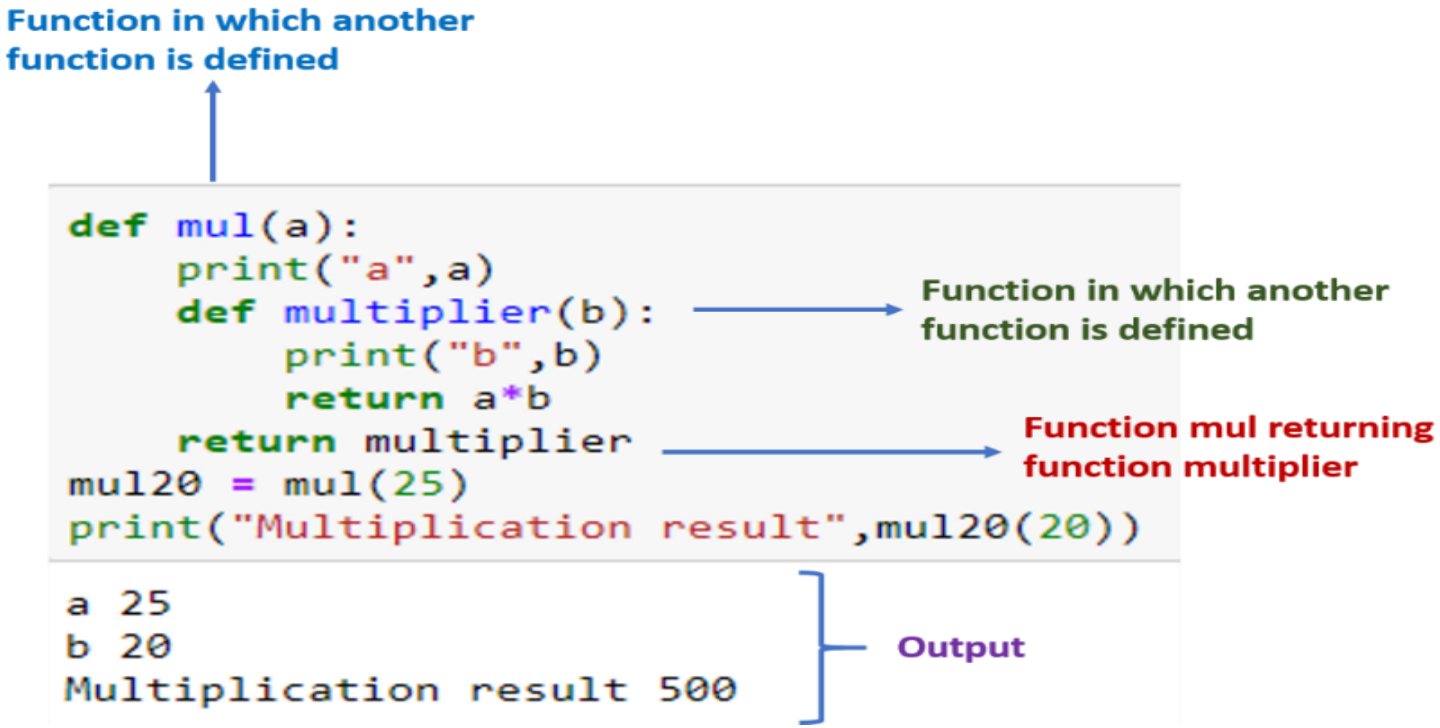
In this example function f2 is being implemented and passed as an argument in function f1. Function f1 evaluates f2 and return the final value as output.



Example 2

A program to demonstrate to return function from a function.

In this example a function **multiplier** has been defined inside function **mul** and **mul** is returning.



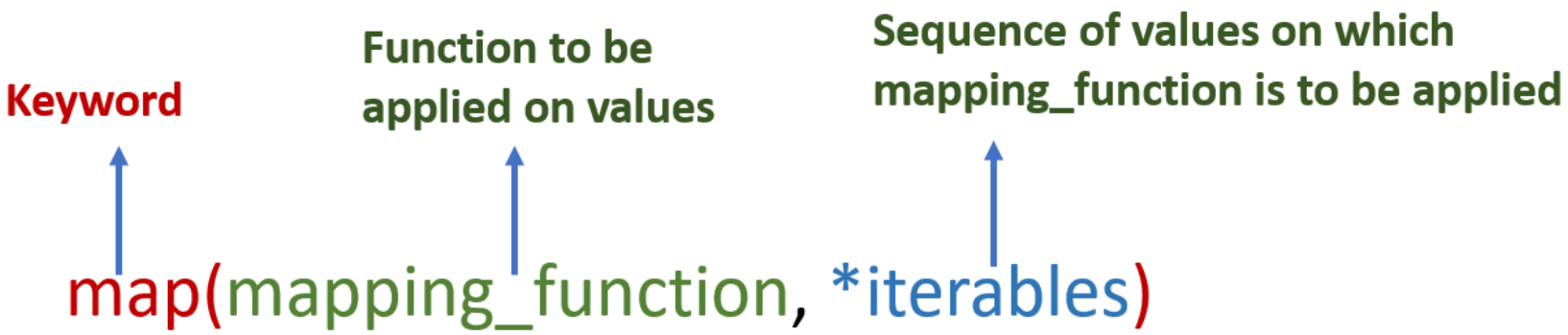
Built-in Higher order function in Python

- Built-in higher order functions are capable of taking other function as argument.
- These higher order functions are applied over a sequence of data.
- Here one thing to note that we can apply same functionality without using these higher order function by doing type casting and loop. But by using higher order function we avoid using of loop.

Map ()

First higher order function we would discuss is map function.

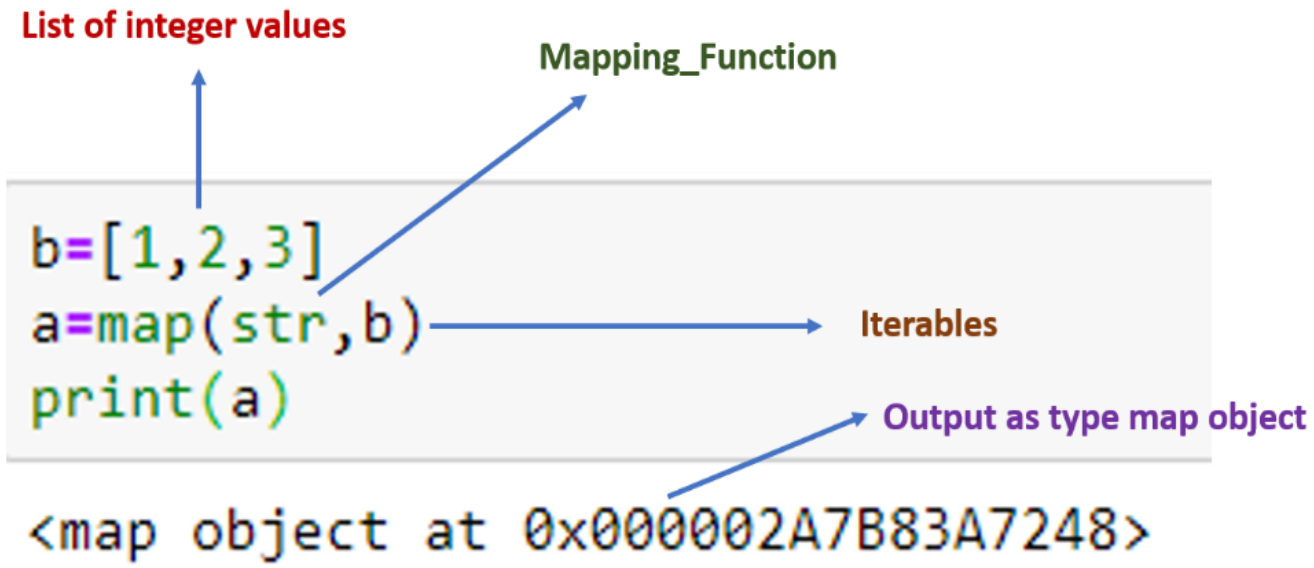
Syntax:



Contd..

One important thing to note that map function always returns map object.

Syntax:



Contd..

To access values, we need to **type cast map** object as some sequence datatype like **lists, tuple, etc.**

```
b=[1,2,3]
a=map(str,b)
list(a)
```

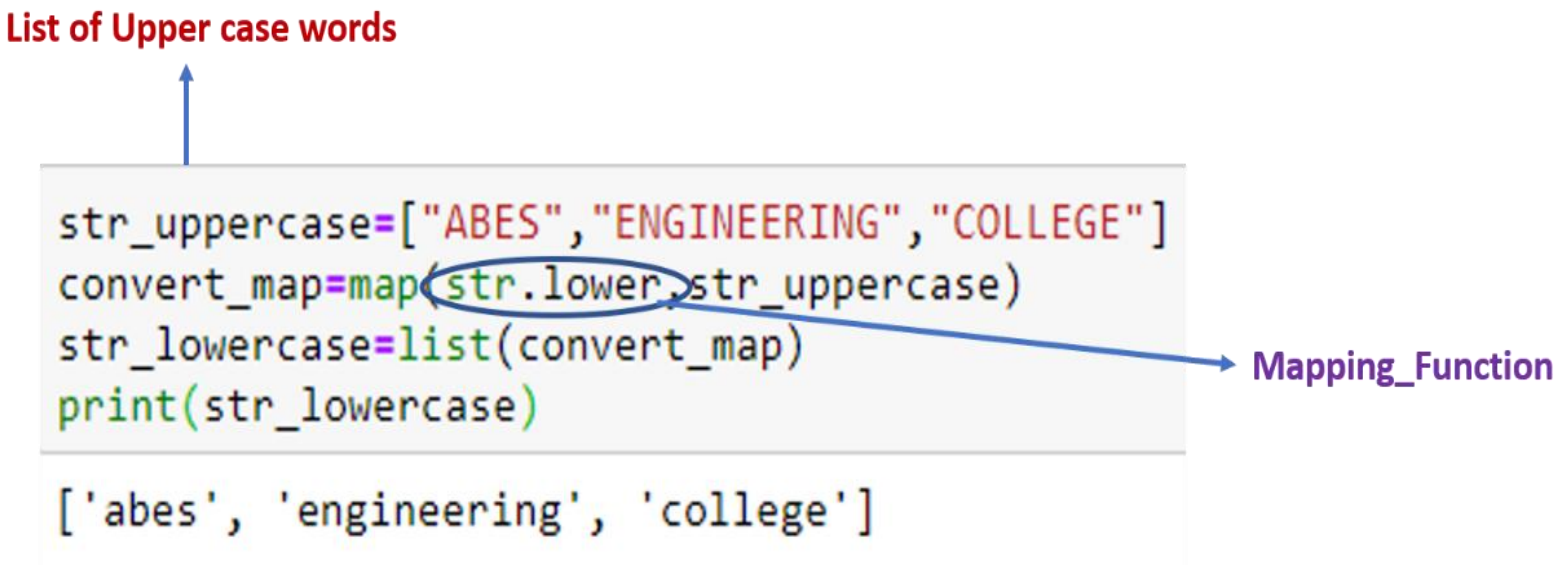
Type casting of map object into list

```
['1', '2', '3']
```

Output List of string values

Contd..

A program to convert a string of upper-case words into lower case.

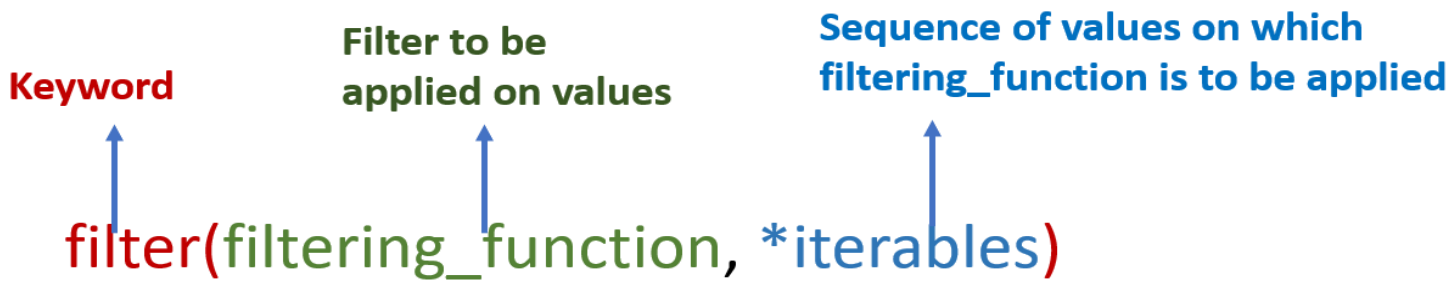


Filter()

Filter function is the high order function which takes a filtering criterion, need to apply on any sequence.

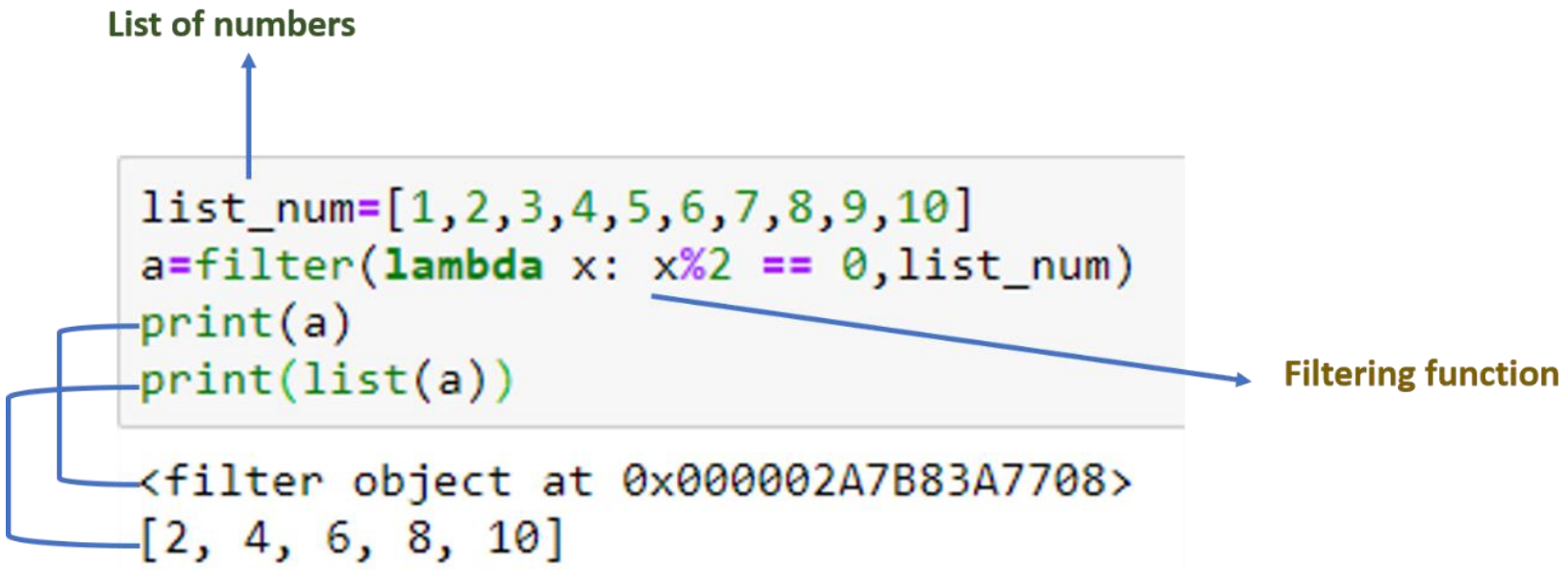
For example, in a bucket of balls of different colors, we have to find only black color balls, then filter function would work.

Syntax:



Contd..

A list of integers find out even numbers.

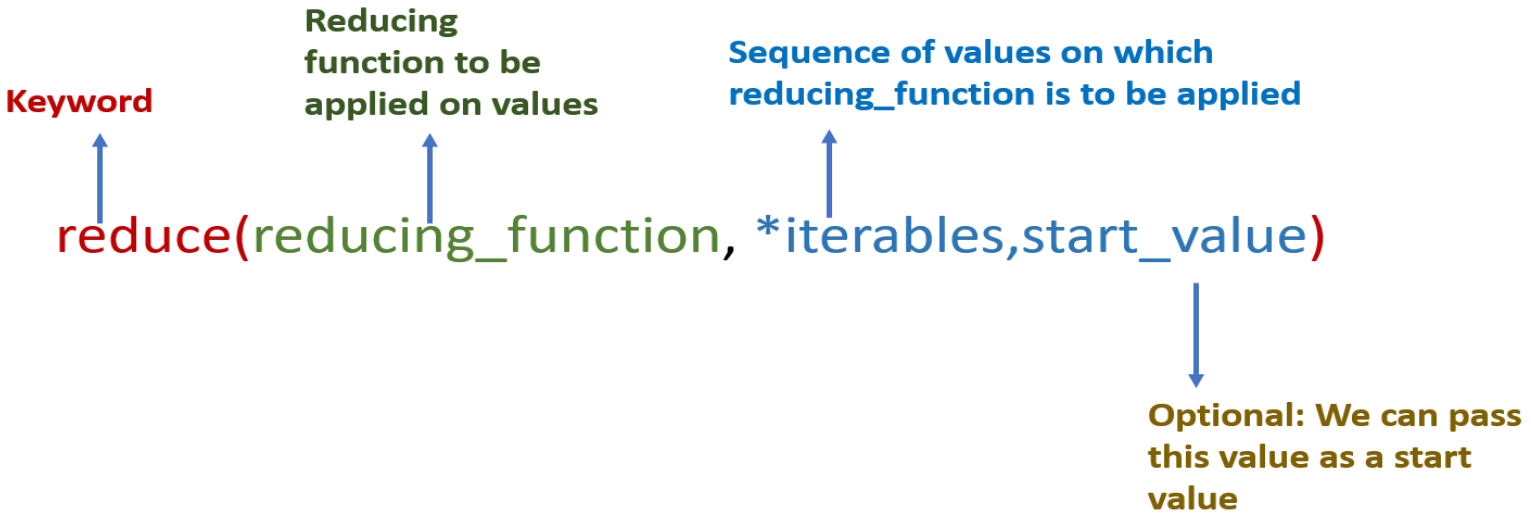


Reduce()

Reduce is used in such scenarios where the values in the iterable would be calculated to a final value.

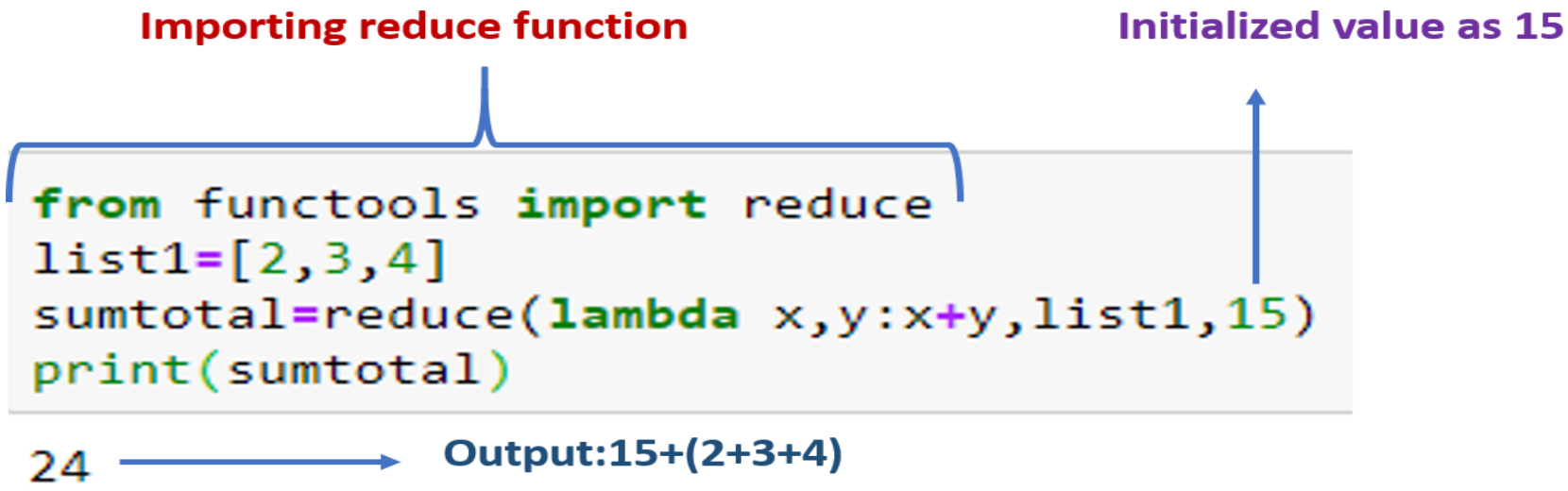
For example, take a collection of values (3,4,5) and we need to apply sum to it. Then we would get 12.

Syntax:



Example 1

A program to sum numbers of a list using reduce function.



Review Questions

What will be the output of the following Python code?

```
x = ['ab', 'cd']
print(len(list(map(list, x))))
```

a) 2
b) 4
c) error
d) none of the mentioned

What will be the output of the following Python code?

```
x = [12, 34]
print(len(list(map(len, x))))
```

a) 2
b) 1
c) error
d) none of the mentioned

Session Plan - Day 5

4.3 Modules

- **4.3.1 Creation of module**
- **4.3.2 Importing module**
- **4.3.3 Standard Built-In Module**

Introduction

As length of program grows in size and to maintain the reusability of some code we start writing function. Now if you want to use same function/functions in another program instead of writing their definition then ***module comes into picture.***

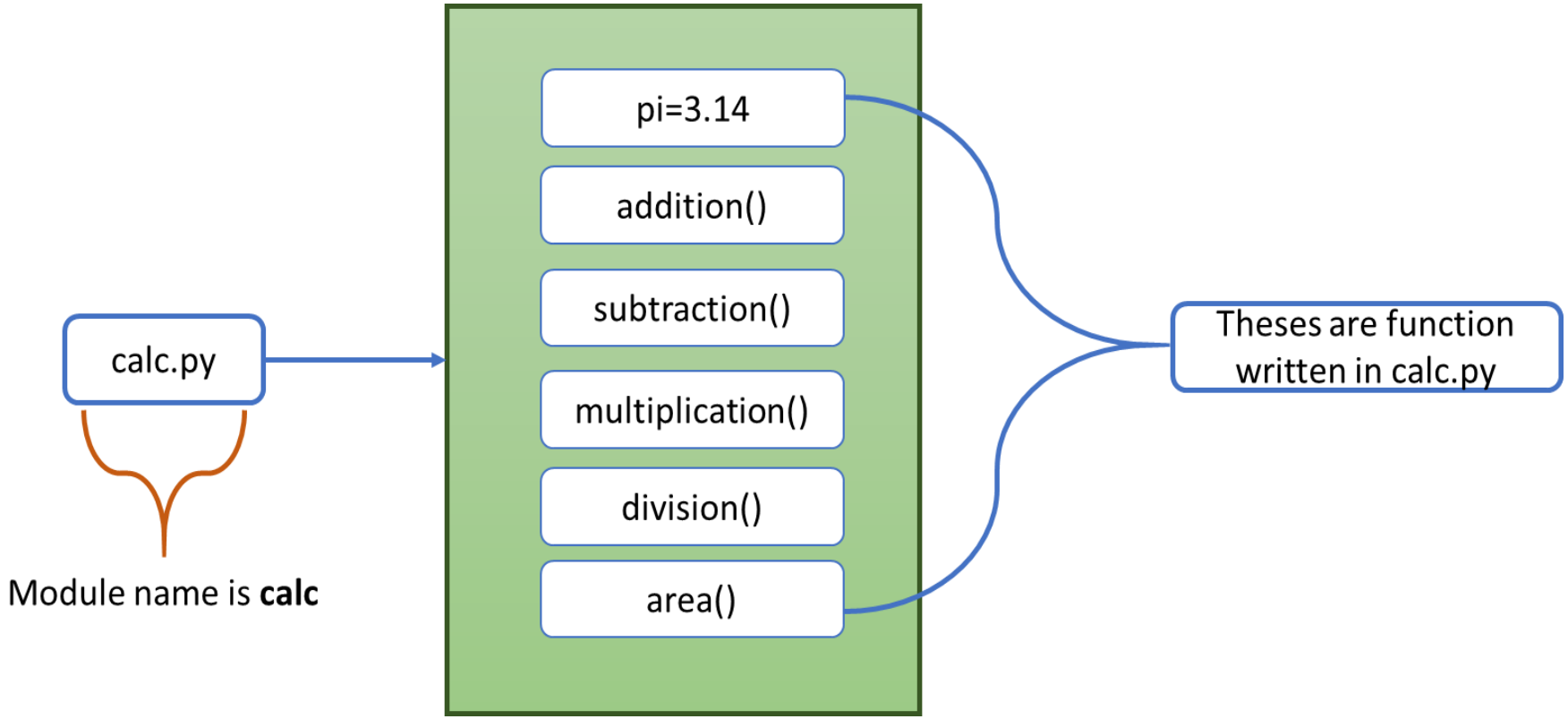
Python provides us to create our **own module** or provides use to use **rich source of given module** to increase the reusability.

Python module can be imported into any program

In python module is simply **a python file containing python code**, it may contain statements, functions, methods and classes.

Creation of module

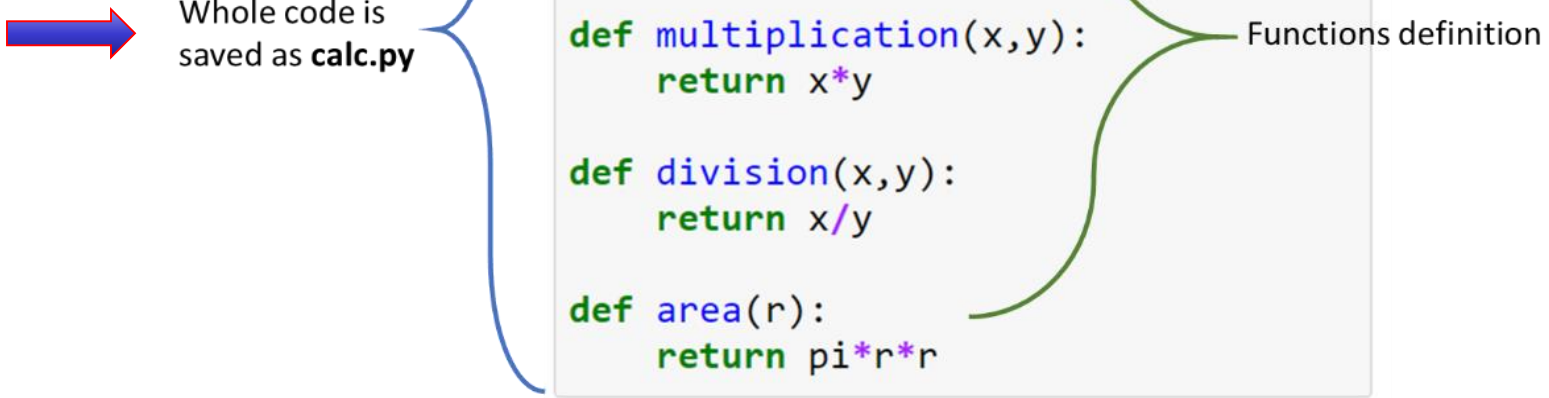
A file containing Python code, for example: calc.py, is called a module, and its name would be calc. A module is created simply putting all functions/statements in one python file



Contd..

A file containing Python code, for example: calc.py, is called a module, and its name would be calc. A module is created simply putting all functions/statements in one python file

Code illustrates that how to create a module name 'calc', which contains the user defined functions of addition, subtraction, multiplication and division and area.

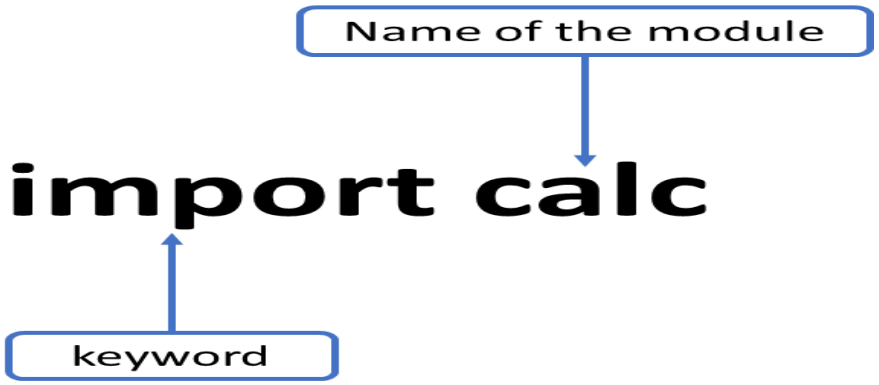


Importing module

To use the module in any application or program we need to import that module using import keyword.

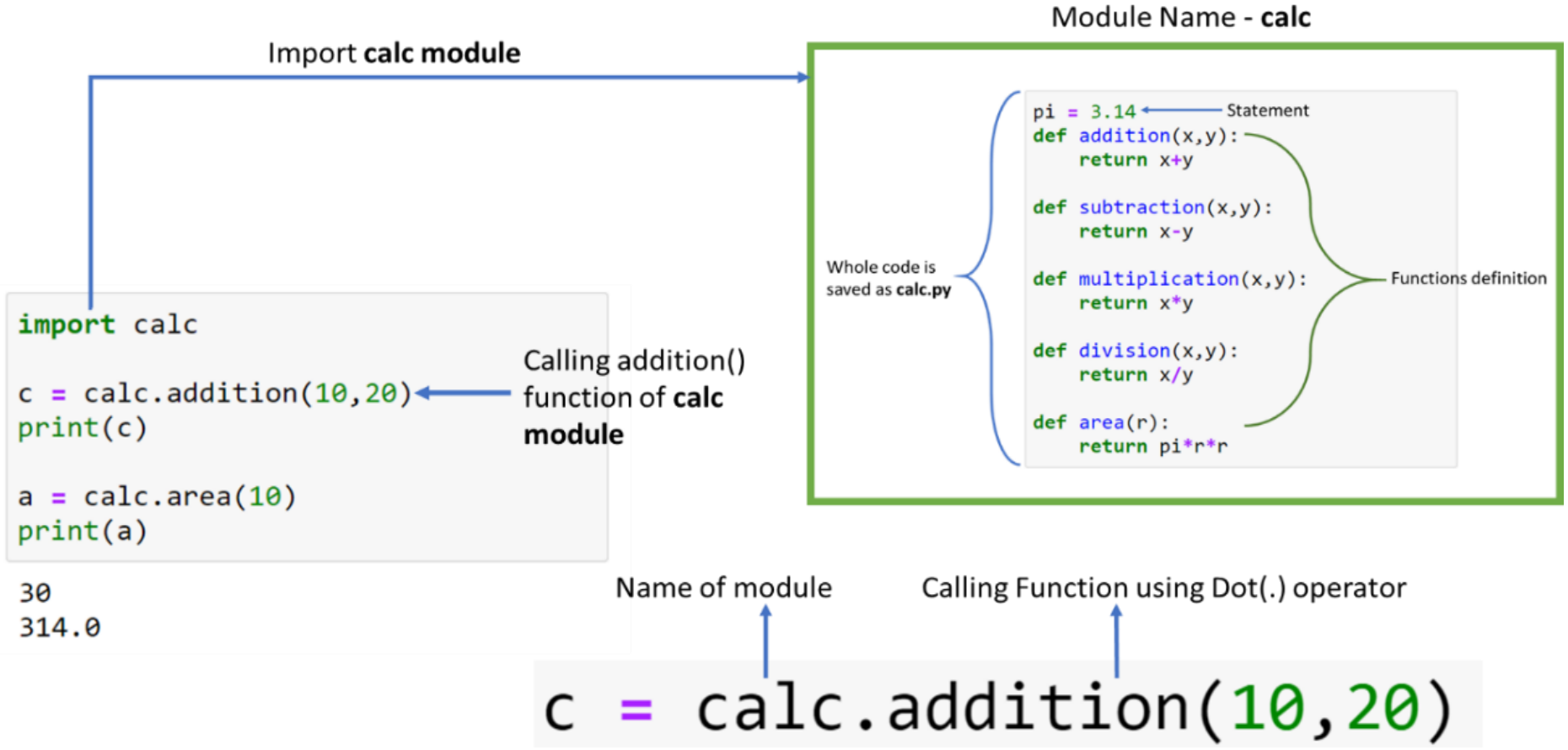
We can import module in multiple ways –

- **import** module-name
- **import ... as ...**
- **from ... import ...**



Contd..

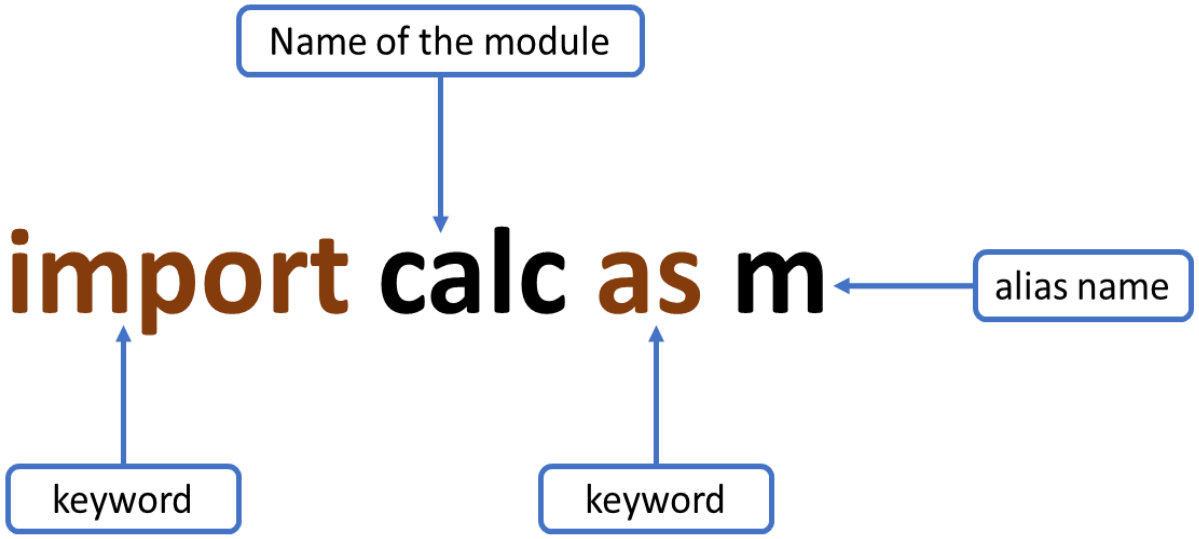
When we use “import module-name” syntax, it will import all functions/statements from that module and function will call using module-name and dot operator.



Contd..

import ... as ...

We use **import ... as ...** to give a module name a **short alias** name while importing it.



Contd..

import ... as ...

Example: We import module calc using alias name ‘m’. It will import all functions / statements from that module and function will call using alias name and dot operator.

alias name

```
import calc as m

c = m.addition(10,20)
print(c)
```

```
30
```

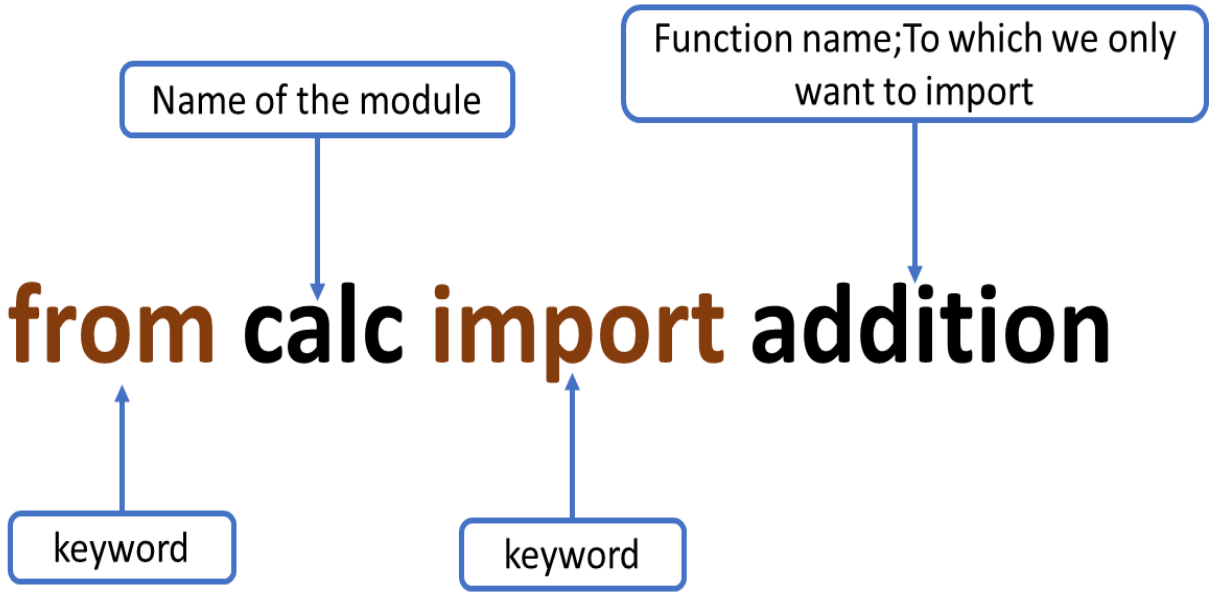
Module Name - **calc**

```
pi = 3.14
def addition(x,y):
    return x+y

def subtraction(x,y):
    return x-y

def multiplication(x,y):
    return x*y

def division(x,y):
    return x/y

def area(r):
    return pi*r*r
```

Contd..

from ... import ...

The **from…. Import …** statement allows us to import specific functions/variables from a module instead of importing everything.



Contd..

from ... import ...

Multiple function can also be imported using comma (,) operator.

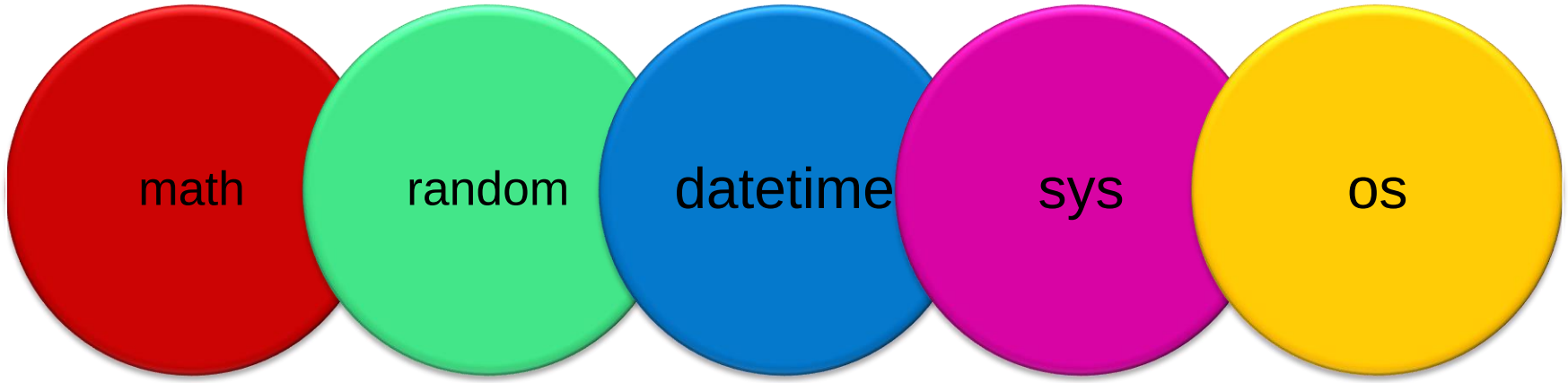
```
from calc import addition, subtraction
```

We can also use '*' , for importing all function/statements.

```
from calc import *
```

Standard Built-In Module

Python has large number of built in functions similarly python contains some of the pre-defined modules such as:



Contd..

The math Module

Python has set of built-in math functions and also has an extensive math module that allows us to perform mathematical tasks on data. These include trigonometric functions, representation functions, logarithmic functions, sine functions, cosine functions, exponential, pi etc.

math.sqrt()

This function will calculate the square root of number.

```
import math

result = math.sqrt(144)
print(result)
```

```
12.0
```

Contd..

math.ceil()

Rounds a number up to the nearest integer.

```
import math

result = math.ceil(14.5)
print(result)
```

```
15
```

```
import math

result = math.ceil(14.1)
print(result)
```

```
15
```

math.exp()

Return 'E' raised to the power of different numbers

```
import math

result = math.exp(65)
print(result)
```

```
1.6948892444103338e+28
```

Contd..

math. factorial()

Return the factorial of a number

```
import math

result = math.factorial(6)
print(result)
```

```
720
```

math.gcd()

Return the greatest common divisor of two integers.

```
import math

result = math.gcd(16,48)
print(result)
```

```
16
```

math.pow()

Returns the value of x to the power of y.

```
import math

result = math.pow(2,3)
print(result)
```

```
8.0
```

Contd..

math.fsum()

Returns the sum of all items in an iterable.

```
import math

l=[12,2,3,4,5]
result = math.fsum(l)
print(result)
```

```
26.0
```

math.sin()

Return the cosine of x (measured in radians).

```
import math

result = math.cos(0)
print(result)
```

```
1.0
```

Refrence : https://docs.python.org/3/library/math.html

Contd..

The random Module

This module implements pseudo-random number generators for various distributions. We can generate random numbers in Python by using random module.

random.seed()

The random number generator needs a number to start with (a seed value).
Use the seed() method to customize the start number of the random number generator.

```
import random

random.seed(10)
print(random.random())
```

```
0.5714025946899135
```

Contd..

random.random()

Return random number between 0.0 and 1.0

```
import random

result = random.random()
print(result)
```

```
0.5780913011344704
```

random.choice()

Choose a random element from a non- empty sequence like string, list, tuple etc.

```
import math

result = math.pow(2,3)
print(result)
```

```
8.0
```

random.randint(a,b)

Return random integer between a and b, including both end points.

```
import random

result = random.randint(11,20)
print(result)
```

```
14
```

random.shuffle()

The shuffle() method takes a sequence and reorganize the order of the items.

```
import random

color = ['Red','Black','Pink','Blue']
random.shuffle(color)
print(color)
```

```
['Black', 'Red', 'Blue', 'Pink']
```

Contd..

The datetime Module

In python there is no date and time data type so python date time module is used for displaying dates and times, modifying dates and time in different format.

This module supplies classes to work with date and time. These classes provide a number of functions and built in methods to deal with dates, times and time intervals.

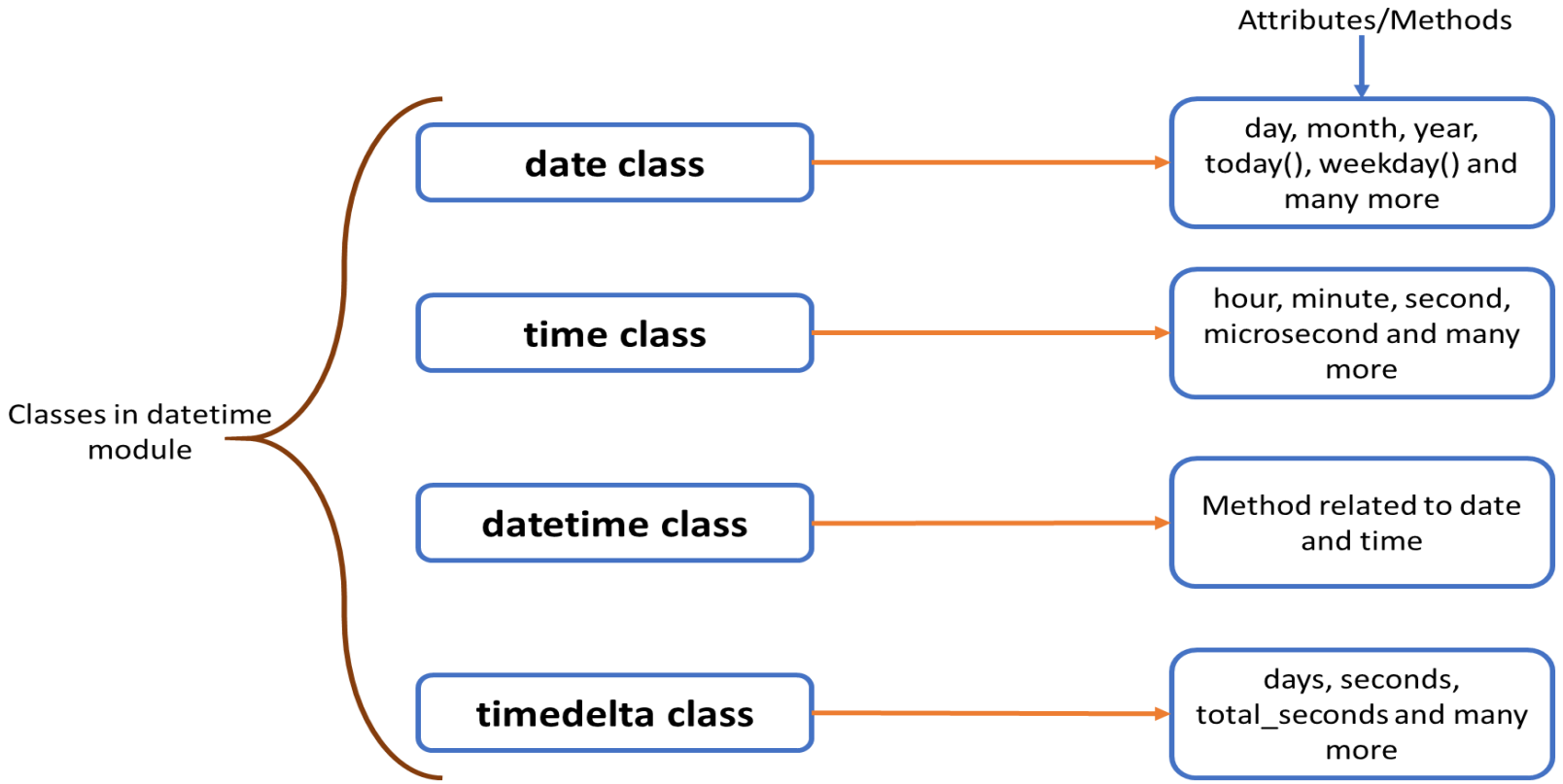
Note:

Classes and Objects we are going to discuss in Object Oriented Programming in Python.

Contd..

The datetime Module

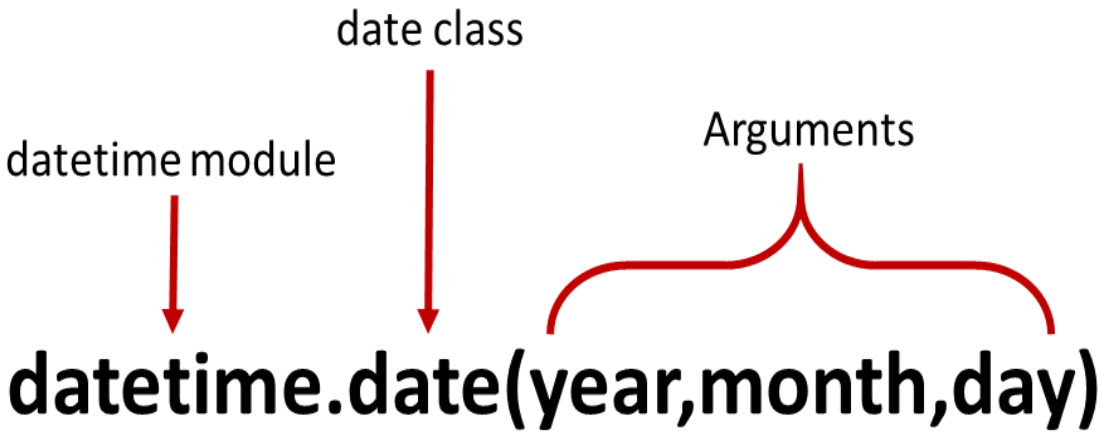
Commonly used classes and associated methods in the datetime m



Contd..

The date class

A date object of date class represents a date in year, month and day.



Contd..

All method and attributes of date object will be called using ‘d’ object and dot operator.

```
import datetime

d = datetime.date(2021,8,8)
print(d)

print(f"Day = {d.day}")

print(f"Month = {d.month}")

print(f"Weekday = {d.weekday()}")

print(f"Today's date{d.today()}")
```

Create a date object, with year=2021, month=8, day=8

d.day give day number in date

d.month give month number in date

d.weekday() return weekday. First day of week start with 0, second 1 and so on

d.today() return current date of system

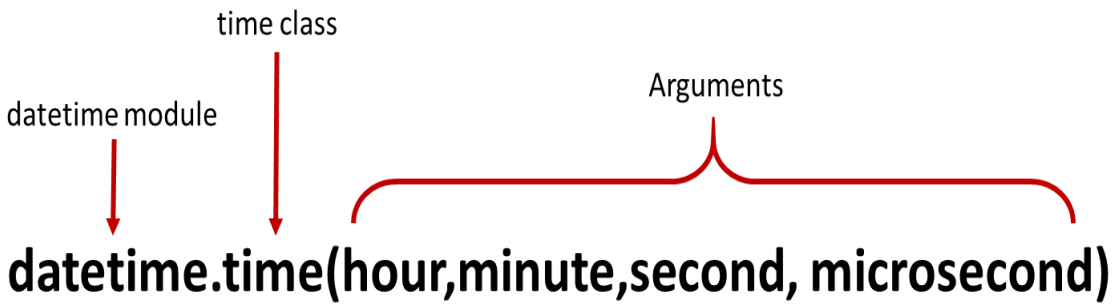
```
2021-08-08
Day = 8
Month = 8
Weekday = 6
Today's date2021-08-10
```

Output

Contd..

The time class

Time object represents local time, independent of any day.



Contd..

Create a time object and display hour, minute, second and microsecond in time object.

```
import datetime

t = datetime.time(9,35,12,123)
print(t)

print(f"Hour= {t.hour}")

print(f"Minutes= {t.minute}")

print(f"Microsecond= {t.microsecond}")
```

Create a time object, with hour=9, minute=35, second=12

t.hour give value of hour in time

t.minute give value of minute in time

t.microsecond give value of microsecond in time

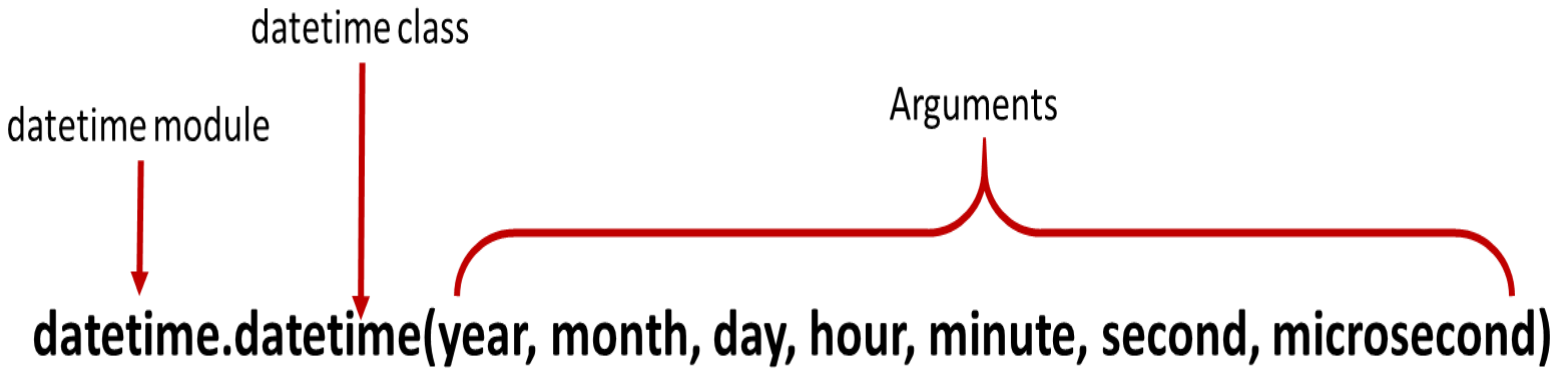
```
09:35:12.000123
Hour= 9
Minutes= 35
Microsecond= 123
```

Output

Contd..

The datetime class

Information on both date and time is contained in this class.



Contd..

Create a time object and display hour, minute, second and microsecond in time object.

```
import datetime

dt1 = datetime.datetime(2021, 8, 11, 10, 43, 18, 462154)
print(f"year = {dt.year}, month = {dt.month} ")
print(f"hour = {dt.hour}, minute = {dt.minute} ")
```

```
year = 2021, month = 8
hour = 10, minute = 43
```

Create a datetime object and display year, month, hour, minute in time object.

Write a program to fetch the current datetime of system and print it

```
import datetime

dt = datetime.datetime.now()
print(dt)
```

```
2021-08-11 12:27:37.279229
```

Contd..

The sys Module

The sys module provides system specific **parameters and functions.** It gives us information about constants, functions, and methods of the interpreter.

Following are some important use of sys module:

- Exiting current flow of execution in Python
- Command-line Arguments
- Getting version information of Interpreter
- Set or get recursion depth

For more function visit **- https://docs.python.org/3/library/sys.html)**

Contd..

sys.version

Gives version information of Interpreter.

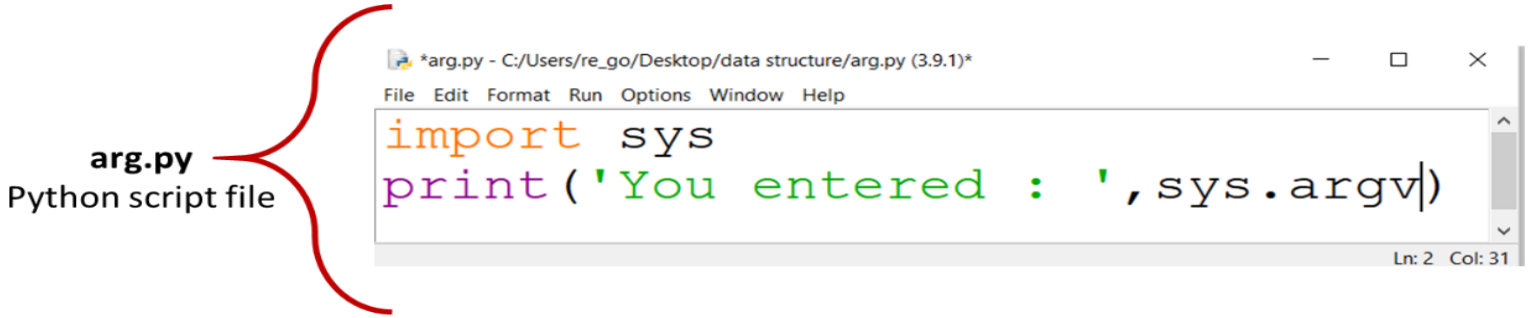
```
import sys
print(sys.version)
```

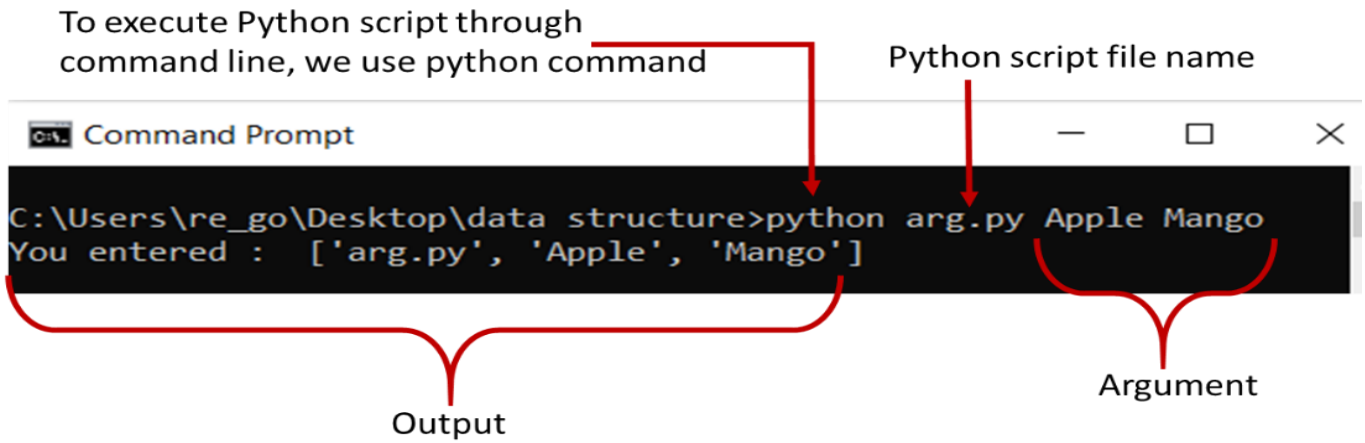
```
3.8.5 (default, Sep  3 2020, 21:29:08)
[MSC v.1916 64 bit (AMD64)]
```

Contd..

sys.argv

It returns a list of command line arguments passed to a Python script.

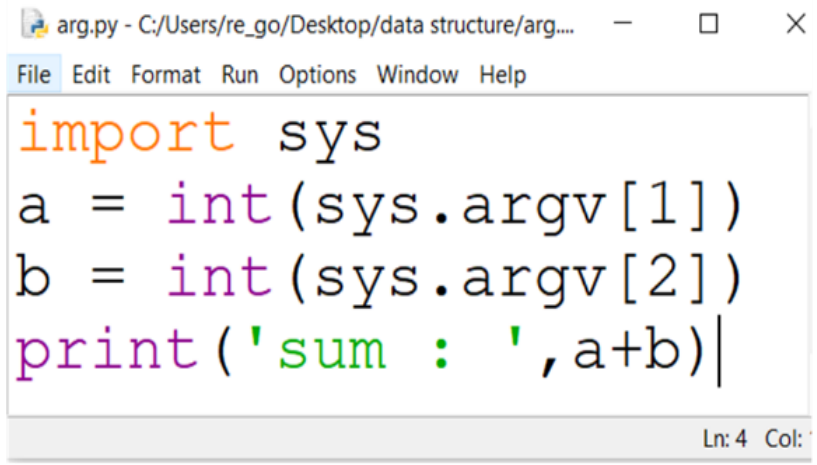




Contd..

sys.argv

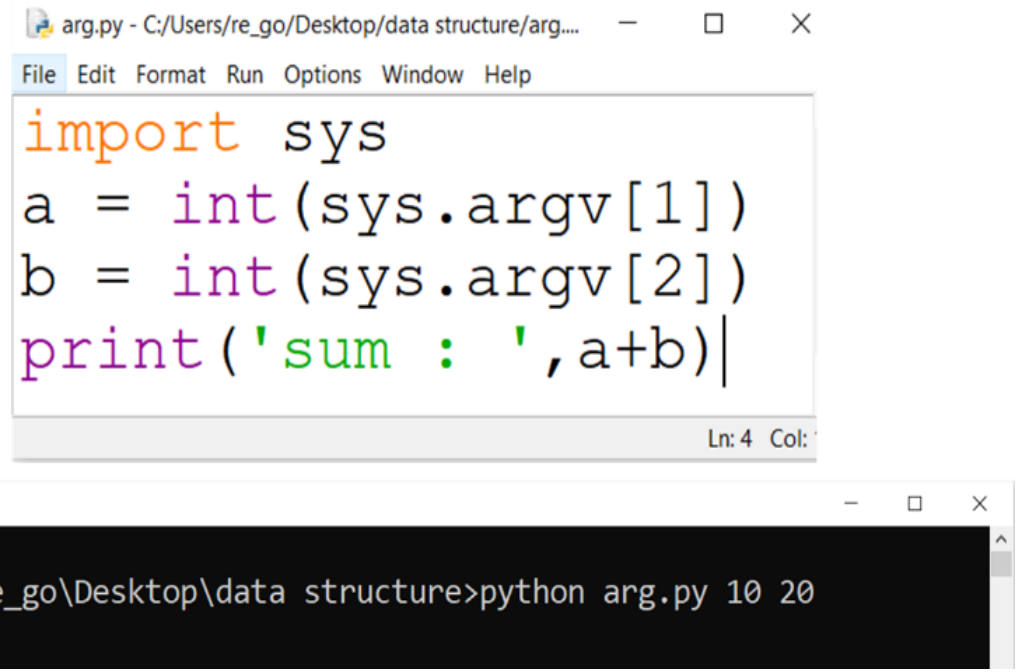
Write a python script which print sum of given two value through command line



Contd..

sys.argv

Write a python script which print sum of given two value through command line



Contd..

sys.getrecursionlimit() and sys.setrecursionlimit()

```
import sys
sys.getrecursionlimit()
```

```
Out[85]:

3000
```

sys.getrecursionlimit()
method is used to find the current recursion limit of the interpreter.

sys.setrecursionlimit()
method is used to set the recursion limit of the interpreter

```
import sys
sys.setrecursionlimit(100)
```

Review Questions

- What is the output of the code shown below if the system date is 18th August, 2016?

```
tday=datetime.date.today()

print(tday.month())
```

A. August

B. Aug

C. 08

D. 8 

Review Questions

What is returned by math.ceil(3.4)?

a) 3

b) 4 

c) 4.0

d) 3.0

What is returned by math.ceil(3.4)?

a) 3 

b) 4

c) 4.0

d) 3.0

Review Questions

- What will be the output of the following Python code?

```
import time
print(time.strftime("%d-%m-%Y"))
```

a) Print today's date in string format dd-mm-yy
b) Print today's date in string format dd-mm-yyyy
c) Error
d) None

- State whether true or false.

```
t1 = time.time()
t2 = time.time()
t1 == t2
```

a)True
b)False

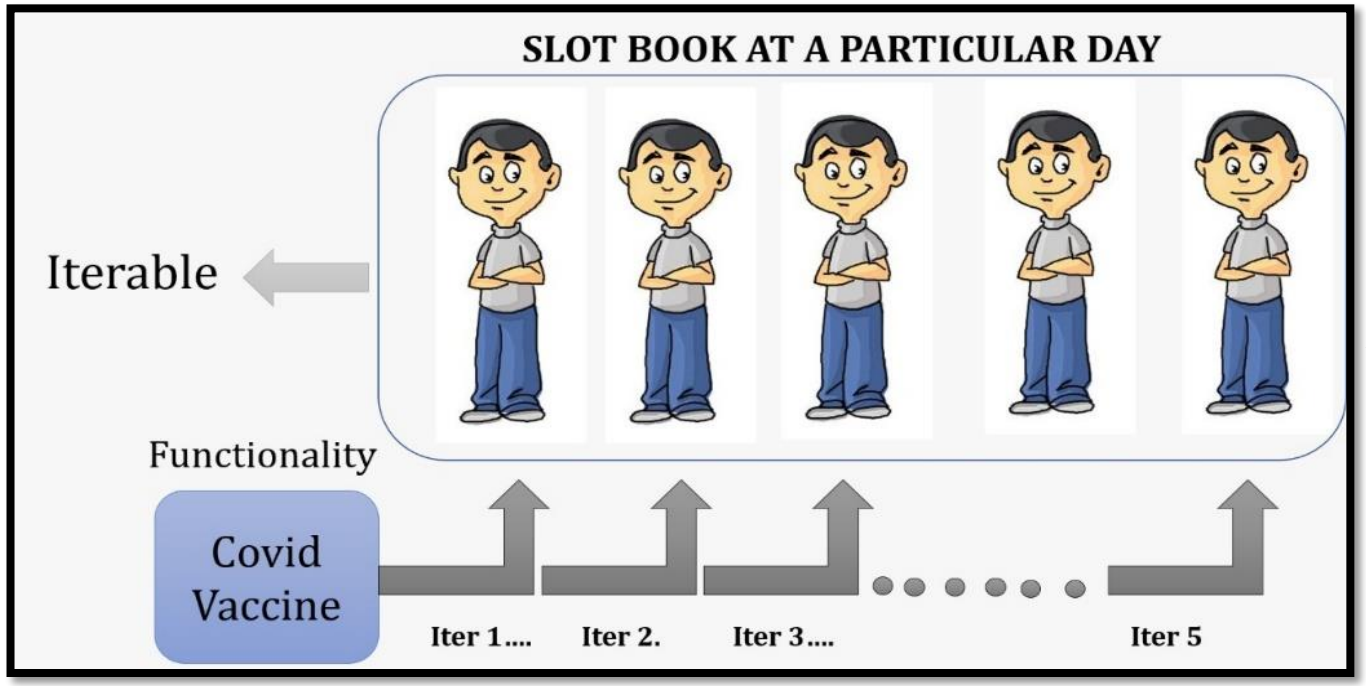
Session Plan - Day 6

4.3 Modules

- 4.4 Iterative Built-in Functions
  - Enumerate
  - Zip
  - Sorted
  - Zip

Iterative Built-in Functions

Enumerate: This is built-in function provided by Python. This function assigns count values to the iterable



What is iterable?

Iterable in python is a collection of items, sequences like lists, tuple or any object which have multiple values on which a repeat operation can be performed, like

name = [‘amit’, ‘ankur’, ‘aditya’, ‘anshul’, ‘anmol’],

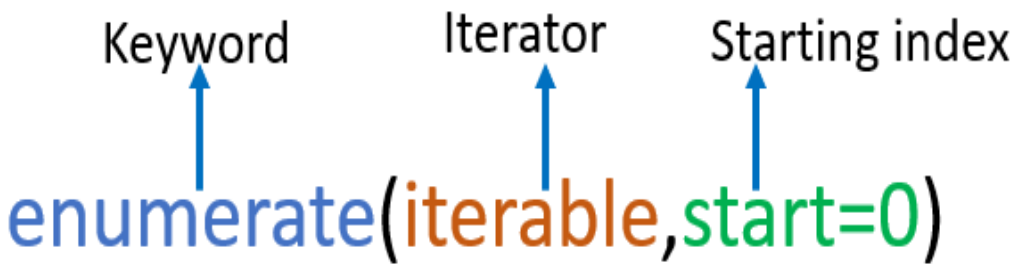
here the ***name*** has five items in it.

Contd..

When using the iterators, we need to keep track of the number of items in the iterator. This is achieved by built in method called enumerate.

This enumerated object can directly be used for loops or converted into a list of tuples using list () method.

Note: by default enumerate starts from 0 but we can change this start value as per our convenience.



Contd..

Example : Demonstrating the enumerate usage in printing days of week.

```
days= { 'Mon', 'Tue', 'Wed','Thu'}
enum_days = enumerate(days)
print(type(enum_days))

# converting it to alist
print(list(enum_days))

# changing the default counter to 5
enum_days = enumerate(days, 5)
print(list(enum_days))
```

Set

Print the days of the week from index 0

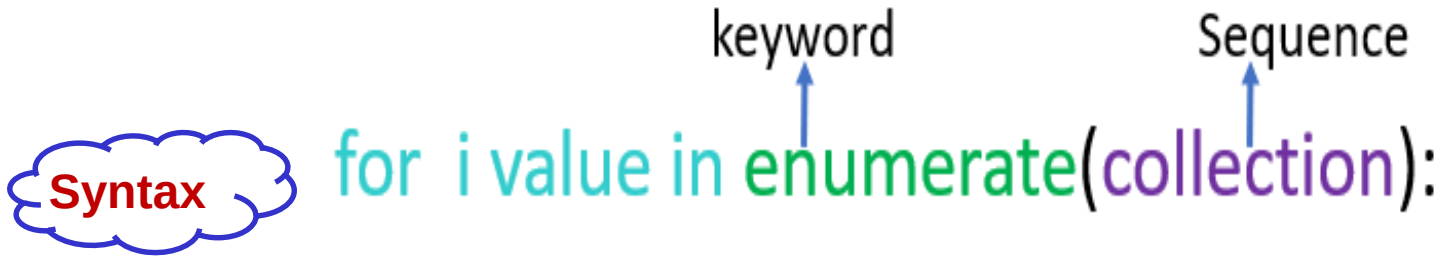
Print the days of the week from index 5

```
<class 'enumerate'>
[(0, 'Mon'), (1, 'Wed'), (2, 'Thu'), (3, 'Tue')]
[(5, 'Mon'), (6, 'Wed'), (7, 'Thu'), (8, 'Tue')]
```

Output

Contd..

Enumerate with for loop



Example

```
k = ['ABES', 'Engineering' ,'College', 'Ghaziabad']
for i in enumerate(k):
    print(i)
```

list

For loop

```
(0, 'ABES')
(1, 'Engineering')
(2, 'College')
(3, 'Ghaziabad')
```

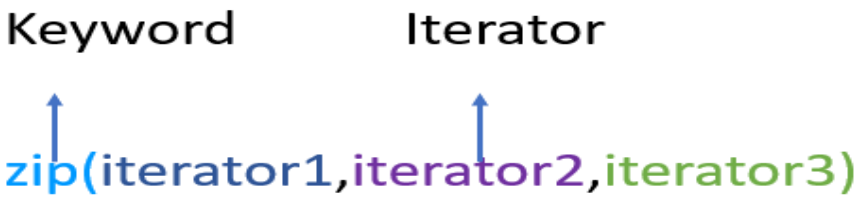
Output

Contd..

Zip

The function returns a zip object, which is an iterator of tuples where the **first item** in each passed iterator is paired together, and then the **second item** in each passed iterator are paired together etc.

Syntax



Example

```
a = ("John", "Charles", "Mike")
b = ("Jenny", "Christy", "Monica", "Vicky")
x = zip(a, b)
print(list(x))
```

tuple

tuple

```
[('John', 'Jenny'), ('Charles', 'Christy'), ('Mike', 'Monica')]
```

Output

Contd..

Sorted

Python contains a special built in method for **sorting** the data in a collection. It preserves the order of previous sequence or collection.

Syntax

Contd..

Sorted Example

```
x = [2, 8, 1, 4, 6, 3, 7]
# x is alist
print ("Sorted List returned :"),
print (sorted(x))
# this will print the list in ascending order
print ("Reverse sort :"),
print (sorted(x, reverse = True))
#this will print the list in descending order if reverse is set true
print ("Original list not modified :"),
print (x)
```

List

This will print in reverse order

```
Sorted List returned :
[1, 2, 3, 4, 6, 7, 8]
Reverse sort :
[8, 7, 6, 4, 3, 2, 1]
Original list not modified :
[2, 8, 1, 4, 6, 3, 7]
```

Output

Contd..

Reversed

Reversed () function returns the iterator in a reverse order. Iterator can be any sequence such as list, tuple etc.

Example

```
# for string
s = 'YELLOW'
print(list(reversed(s)))
# for tuple
t= ('P', 'y', 't', 'h', 'o', 'n')
print(list(reversed(t)))
# for range
r = range(5, 9)
print(list(reversed(r)))
# for list
l = [1, 2, 4, 3, 5]
print(list(reversed(l)))
```

String

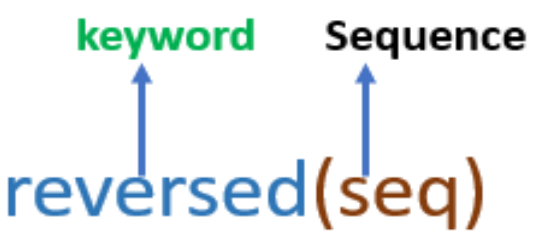
Tuple

range

```
['W', 'O', 'L', 'L', 'E', 'Y']
['n', 'o', 'h', 't', 'y', 'P']
[8, 7, 6, 5]
[5, 3, 4, 2, 1]
```

Output

Syntax



Review Questions

- Which of the following functions can help us to find the version of python that we are currently working on?
  a) sys.version
  b) sys.version()
  c) sys.version(0)
  d) sys.version(1)

- Suppose there is a list such that: l=[2,3,4]. If we want to print this list in reverse order, which of the following methods should be used?
  a) reverse(l)
  b) list(reverse[(l)])
  c) reversed(l)
  d) list(reversed(l))

Review Questions

- Which of the following functions is not defined under the sys module?
  a) sys.platform
  b) sys.path
  c) sys.readline
  d) sys.argv
- To obtain a list of all the functions defined under sys module, which of the following functions can be used?
  a) print(sys)
  b) print(dir.sys)
  c) print(dir[sys])
  d) print(dir(sys))

Summary

- In this unit you learnt the importance of using functions and how to use functions.
- Further you learnt how to define functions, to write functions with different argument types.
- Further you learnt describe the difference between the in- built functions and user defined functions,
- Further you to write user defined functions, to use Functions in- built functions, import math module in to a Python program and to use lambda functions.

References

1. https://docs.python.org/3/tutorial/controlflow.html
2. Think Python: An Introduction to Software Design, Book by Allen B. Downey
3. Head First Python, 2nd Edition, by Paul Barry
4. Python Basics: A Practical Introduction to Python, by David Amos, Dan Bader, Joanna Jablonski, Fletcher Heisler
5. https://www.fullstackpython.com/turbogears.html
6. https://www.cubicweb.org
7. https://pypi.org/project/Pylons/
8. https://www.upgrad.com/blog/python-applications-in-real-world/
9. https://www.codementor.io/@edwardbailey/coding-vs-programming-what-s-the-difference-yr0aeug9o

Thank You