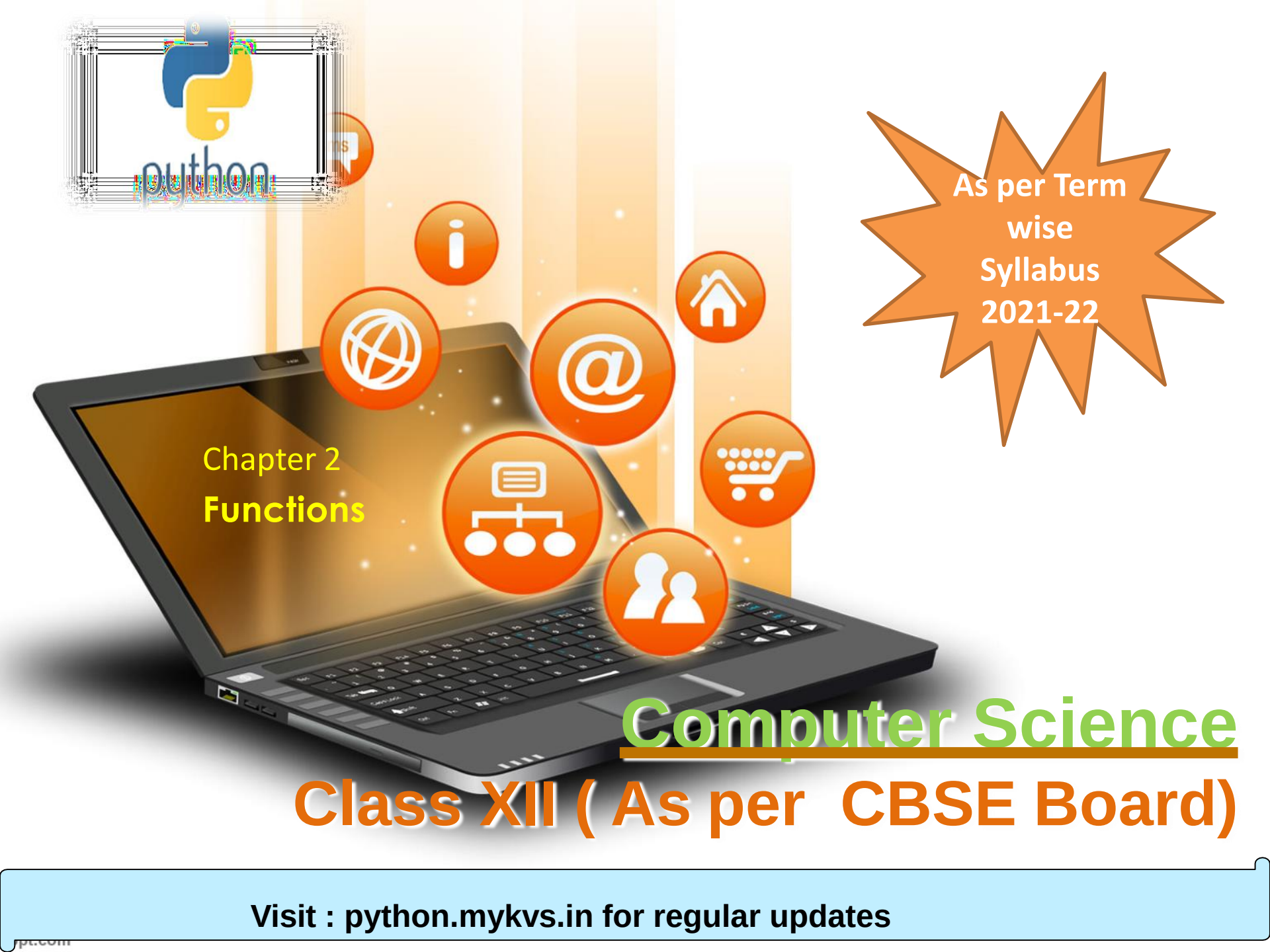
python
As per Term wise Syllabus 2021-22
Chapter 2
Functions
Computer Science
Class XII (As per CBSE Board)
Visit : python.mykvs.in for regular updates

Function Introduction

A function is a programming block of codes which is used to perform a single, related task. It only runs when it is called. We can pass data, known as parameters, into a function. A function can return data as a result.

We have already used some python **built in functions** like print(),etc.But we can also create our own functions. These functions are called ***user-defined functions.***

Advantages of Using functions:

1.Program development made easy and fast : Work can be divided among project members thus implementation can be completed fast.

2.Program testing becomes easy : Easy to locate and isolate a faulty function for further investigation

3.Code sharing becomes possible : A function may be used later by many other programs this means that a python programmer can use function written by others, instead of starting over from scratch.

4.Code re-usability increases : A function can be used to keep away from rewriting the same block of codes which we are going use two or more locations in a program. This is especially useful if the code involved is long or complicated.

5.Increases program readability : The length of the source program can be reduced by using/calling functions at appropriate places so program become more readable.

6.Function facilitates procedural abstraction : Once a function is written, programmer would have to know to invoke a function only ,not its coding.

7.Functions facilitate the factoring of code : A function can be called in other function and so on...

Creating & calling a Function (user defined)/Flow of execution

A function is defined using the def keyword in python.E.g. program is given below.

```
def my_own_function():
  print("Hello from a function")


#program start here.program code
print("hello before calling a function")
my_own_function() #function calling.now function codes will be executed
print("hello after calling a function")
```

#Function block/ definition/creation

Save the above source code in python file and execute it

Variable's Scope in function

There are three types of variables with the view of scope.

1. Local variable – accessible only inside the functional block where it is declared.
2. Global variable – variable which is accessible among whole program using global keyword.
3. Non local variable – accessible in nesting of functions,using nonlocal keyword.

Local variable program:

```
def fun():
    s = "I love India!" #local variable
    print(s)

s = "I love World!"
fun()
print(s)
```

Output:

I love India!
I love World!

Global variable program:

```
def fun():
    global s #accessing/making global variable for fun()
    print(s)
    s = "I love India!“ #changing global variable's value
    print(s)
s = "I love world!"
fun()
print(s)
```

Output:

I love world!
I love India!
I love India!

Variable's Scope in function

#Find the output of below program

```
def fun(x, y): # argument /parameter x and y
    global a
    a = 10
    x,y = y,x
    b = 20
    b = 30
    c = 30
    print(a,b,x,y)

a, b, x, y = 1, 2, 3,4
fun(50, 100) #passing value 50 and 100 in parameter x and y of function fun()
```

Variable's Scope in function

```
#Find the output of below program
def fun(x, y): # argument /parameter x and y
    global a
    a = 10
    x,y = y,x
    b = 20
    b = 30
    c = 30
    print(a,b,x,y)

a, b, x, y = 1, 2, 3,4
fun(50, 100) #passing value 50 and 100 in parameter x and y of function fun()
print(a, b, x, y)
```

OUTPUT :-

```
10 30 100 50
10 2 3 4
```

Variable's Scope in function

Global variables in nested function

```
def fun1():
    x = 100
    def fun2():
        global x
        x = 200
    print("Before calling fun2: " + str(x))
    print("Calling fun2 now:")
    fun2()
    print("After calling fun2: " + str(x))

fun1()
print("x in main: " + str(x))
```

OUTPUT:

```
Before calling fun2: 100
Calling fun2 now:
After calling fun2: 100
x in main: 200
```

Variable's Scope in function

Non local variable

```
def fun1():
    x = 100
    def fun2():
        nonlocal x #change it to global or remove this declaration
        x = 200
    print("Before calling fun2: " + str(x))
    print("Calling fun2 now:")
    fun2()
    print("After calling fun2: " + str(x))

x=50
fun1()
print("x in main: " + str(x))
```

OUTPUT:

```
Before calling fun2: 100
Calling fun2 now:
After calling fun2: 200
x in main: 50
```

Function

Parameters / Arguments Passing and return value

These are specified after the function name, inside the parentheses. Multiple parameters are separated by comma.The following example has a function with two parameters x and y. When the function is called, we pass two values, which is used inside the function to sum up the values and store in z and then return the result(z):

```
def sum(x,y):  #x, y are formal arguments
    z=x+y
    return z  #return the value/result


x,y=4,5
r=sum(x,y) #x, y are actual arguments
print(r)
```

Note :- 1. Function Prototype is declaration of function with name ,argument and return type. 2. A formal parameter, i.e. a parameter, is in the function definition. An actual parameter, i.e. an argument, is in a function call.

Function

Function Arguments

Functions can be called using following types of formal arguments –

- Required arguments/**Positional parameter** - arguments passed in correct positional order
- Keyword arguments - the caller identifies the arguments by the parameter name
- Default arguments - that assumes a default value if a value is not provided to argu.
- Variable-length arguments – pass multiple values with single argument name.

```
#Required arguments

def square(x):
   z=x*x
   return z

r=square()
print(r)
#In above function square() we have to definitely need to pass some value to argument x.
```

```
#Keyword arguments
def fun( name, age ):
   "This prints a passed info into this function"
   print ("Name: ", name)
   print ("Age ", age)
   return;

# Now you can call printinfo function
fun( age=15, name="mohak" )
# value 15 and mohak is being passed to relevant argument based on keyword used for them.
```

Function

#Default arguments / #Default Parameter

```
def sum(x=3,y=4):
    z=x+y
    return z

r=sum()
print(r)
r=sum(x=4)
print(r)
r=sum(y=45)
print(r)
```

#default value of x and y is being used when it is not passed

```
#Variable length arguments
def sum( *vartuple ):
   s=0
   for var in vartuple:
      s=s+int(var)
   return s;

r=sum( 70, 60, 50 )
print(r)
r=sum(4,5)
print(r)
```

#now the above function sum() can sum n number of values

Lamda

Python Lambda

A lambda function is a small anonymous function which can take any number of arguments, but can only have one expression.

E.g.

```
x = lambda a, b : a * b
print(x(5, 6))
```

OUTPUT:

30

- **NEXT SLIDES FROM SLIDE NO 15 TO 26 ARE DEPRECATED AS PER CBSE SYLLABUS 2021-22**

Mutable/immutable properties of data objects w/r function

Everything in Python is an object,and every objects in Python can be either mutable or immutable.

Since everything in Python is an Object, every variable holds an object instance. When an object is initiated, it is assigned a unique object id. Its type is defined at runtime and once set can never change, however its state can be changed if it is mutable.

Means a mutable object can be changed after it is created, and an immutable object can't.

Mutable objects: list, dict, set, byte array

Immutable objects: int, float, complex, string, tuple, frozen set ,bytes

Mutable/immutable properties of data objects w/r function

How objects are passed to Functions

#Pass by reference

```
def updateList(list1):
    print(id(list1))
    list1 += [10]
    print(id(list1))
n = [50, 60]
print(id(n))
updateList(n)
print(n)
print(id(n))
```

OUTPUT

```
34122928
34122928
34122928
[50, 60, 10]
34122928
```

#In above function list1 an object is being passed and its contents are changing because it is mutable that's why it is behaving like pass by reference

#Pass by value

```
def updateNumber(n):
    print(id(n))
    n += 10
    print(id(n))
b = 5
print(id(b))
updateNumber(b)
print(b)
print(id(b))
```

OUTPUT

```
1691040064
1691040064
1691040224
5
1691040064
```

#In above function value of variable b is not being changed because it is immutable that's why it is behaving like pass by value

Pass arrays/list to function

Arrays are popular in most programming languages like: Java, C/C++, JavaScript and so on. However, in Python, they are not that common. When people talk about Python arrays, more often than not, they are talking about Python lists. Array of numeric values are supported in Python by the array module.

e.g.

```
def dosomething( thelist ):
    for element in thelist:
        print (element)

dosomething( ['1','2','3'] )
alist = ['red','green','blue']
dosomething( alist )
```

OUTPUT:

```
1
2
3
red
green
Blue
```

Note:- List is mutable datatype that's why it treat as pass by reference.It is already explained in topic Mutable/immutable properties of data objects w/r function

Pass String to a function

String can be passed in a function as argument but it is used as pass by value.It can be depicted from below program. As it will not change value of actual argument.

e.g.

```
def welcome(title):
    title="hello"+title


r="Mohan"
welcome(r)
print(r)
```

OUTPUT

Mohan

Pass tuple to a function

in function call, we have to explicitly define/pass the tuple.It is not required to specify the data type as tuple in formal argument.

E.g.

```
def Max(myTuple):
    first, second = myTuple
    if first>second:
        return first
    else:
        return second

r=(3, 1)
m=Max(r)
print(m)
```

OUTPUT

3

Pass dictionary to a function

In Python, everything is an object, so the dictionary can be passed as an argument to a function like other variables are passed.

```
def func(d):
    for key in d:
        print("key:", key, "Value:", d[key])

# Driver's code
Mydict = {'a':1, 'b':2, 'c':3}
func(Mydict)
```

OUTPUT

```
key: a Value: 1
key: b Value: 2
key: c Value: 3
```

Mathematical functions:

Mathematical functions are available under math module.To use mathematical functions under this module, we have to import the module using import math.

For e.g.

To use sqrt() function we have to write statements like given below.

```
import math
r=math.sqrt(4)
print(r)
```

OUTPUT :

2.0

Functions using libraries

Functions available in Python Math Module

Function	Description	Example
ceil(n)	It returns the smallest integer greater than or equal to n.	math.ceil(4.2) returns 5
factorial(n)	It returns the factorial of value n	math.factorial(4) returns 24
floor(n)	It returns the largest integer less than or equal to n	math.floor(4.2) returns 4
fmod(x, y)	It returns the remainder when n is divided by y	math.fmod(10.5,2) returns 0.5
exp(n)	It returns e**n	math.exp(1) return 2.718281828459045
log2(n)	It returns the base-2 logarithm of n	math.log2(4) return 2.0
log10(n)	It returns the base-10 logarithm of n	math.log10(4) returns 0.6020599913279624
pow(n, y)	It returns n raised to the power y	math.pow(2,3) returns 8.0
sqrt(n)	It returns the square root of n	math.sqrt(100) returns 10.0
cos(n)	It returns the cosine of n	math.cos(100) returns 0.8623188722876839
sin(n)	It returns the sine of n	math.sin(100) returns -0.5063656411097588
tan(n)	It returns the tangent of n	math.tan(100) returns -0.5872139151569291
pi	It is pi value (3.14159...)	It is (3.14159...)
e	It is mathematical constant e (2.71828...)	It is (2.71828...)

Functions using libraries (System defined function)

String functions:

String functions are available in python standard module.These are always availble to use.

For e.g. capitalize() function Converts the first character of string to upper case.

```
s="i love programming"
r=s.capitalize()
print(r)
```

OUTPUT:

I love programming

Functions using libraries

String functions:

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found

Functions using libraries

String functions:

Method	Description
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Joins the elements of an iterable to the end of the string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
partition()	Returns a tuple where the string is parted into three parts

Functions using libraries

String functions:

Method	Description
replace()	Returns a string where a specified value is replaced with a specified value
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning