Python Programming

High performance. Delivered.

Module 6 - Numeric & Strings Functions

accenture

Strategy | Digital | Technology | Operations

Agenda

- Functions
- Built-in Functions
- Numeric Functions
- String Functions

Module Objectives

At the end of this module, you will be able to understand

- Python Functions, Syntax of Function , Function Call , Docstring , The return statement
- Scope and Lifetime of variables , Types of Functions, Python Function Arguments
- Variable Function Arguments , Default Arguments to functions, Arbitrary Arguments
- Python Recursion, Advantages and Disadvantages of recursion
- Python Anonymous/Lambda Function
- Numeric Functions
- String Functions

Python Functions

In Python,

- ➢Function is a group of related statements that perform a specific task.
- ➢Functions help break our program into smaller and modular units.
- ➢Functions avoids repetition and makes code reusable.

Types of Functions

Functions can be of two types:

- Built-in functions - Functions that are built into Python.
- User-defined functions - Functions defined by the users themselves.

Defining a function

- Syntax

```
def function_name(parameters):
    """docstring"""
    statement(s)
```

A function definition which consists of following components.

1. Keyword `def` marks the start of function header.
2. A function name to uniquely identify it. Function naming follows the same rules of writing identifiers in Python.
3. Parameters (arguments) through which we pass values to a function. They are optional.
4. A colon (:) to mark the end of function header.
5. Optional documentation string (docstring) to describe what the function does.
6. One or more valid python statements that make up the function body. Statements must have same indentation level (usually 4 spaces).
7. An optional `return` statement to return a value from the function.

Function Definition - Example

```
import os
os.system('cls')
def Welcome(name):
   """This function welcomes the person(name) passed in as Parameter """
   print("Hello, " + name + ". Welcome to Python functions!")
   print("Inside Function.....");
   print("End of Function.....");

print("Outside Function........");
```

OUTPUT

```
Outside Function........
```

Note: Function is not called

Calling a function

- After defining a function, we can call it from another function, program or even the Python prompt.
- To call a function we simply type the function name with appropriate parameters.

```
import os
os.system('cls')
def Welcome(name):
   """This function welcomes the person(name) passed in as Parameter """
   print("Hello, " + name + ". Welcome to Python functions!")
   print("Inside Function.....");
   print("End of Function.....");

print("Outside Function........");
Welcome("ACCENTURE");                    # Function Call
```

```
OUTPUT
Outside Function........
Hello, ACCENTURE. Welcome to Python functions!
Inside Function.....
End of Function.....!
```

Docstring

- The first string after the function header is called the docstring and is short for documentation string.
- Used to explain in brief, what a function does.
- Docstring is optional, but, documentation is a good programming practice.
- Use triple quotes so that docstring can extend up to multiple lines.
- This string is available to us as __doc__ attribute of the function.

```
import os
os.system('cls')
def Welcome(name):
   """This function welcomes the person(name) passed in as Parameter """
   print("Hello, " + name + ". Welcome to Python functions!")
   print("Inside Function.....");
   print("End of Function.....");
print("Outside Function........");
Welcome("ACCENTURE");
print("DOCSTRING = "+Welcome.__doc__);
```

OUTPUT

```
Outside Function........
Hello, ACCENTURE. Welcome to Python functions!
Inside Function.....
End of Function.....
DOCSTRING = This function welcomes the person(name) passed in as Parameter
```

The return statement

The return statement is used to exit a function and go back to the place from where it was called.

Syntax:

```
return [expression_list]
```

- ✓ This statement can contain expression which gets evaluated and the value is returned.
- ✓ If there is no expression in the statement or the `return` statement itself is not present inside a function, then the function will return the `None` object.

The return statement – Example 1

```
import os
os.system('cls')
def Welcome(name):
   """This function welcomes the person(name) passed in as Parameter """
   print("Hello, " + name + ". Welcome to Python functions!")
   print("Inside Function.....");
   print("End of Function.....");
   return "Great";
print("Outside Function........");
Message = Welcome("ACCENTURE");                # Function Call
print (Message);
```

OUTPUT

```
Outside Function........
Hello, ACCENTURE. Welcome to Python functions!
Inside Function.....
End of Function.....
Great
```

The return statement – Example 2

```
import os
os.system('cls')
def Welcome(name):
   """This function welcomes the person(name) passed in as Parameter """
   print("Hello, " + name + ". Welcome to Python functions!")
   print("Inside Function.....");
   print("End of Function.....");
   return ;
print("Outside Function........");
Message = Welcome("ACCENTURE");                    # Function Call
print (Message);
```

<u>OUTPUT</u>

```
Outside Function........
Hello, ACCENTURE. Welcome to Python functions!
Inside Function.....
End of Function.....
None
```

Scope and Lifetime of variables

Scope of a variable is the portion of a program where the variable is recognized.

There are two basic scopes of variables in Python −

- Global variables
- Local variables

Parameters and variables defined inside a function have local scope and they are not visible from outside.

Lifetime of a variable is the period throughout which the variable exists in the memory.

The lifetime of variables inside a function is as long as the function executes.

They are automatically destroyed once the function exits.

Scope and Lifetime of variables – Example 1

```
def function1():
   x = 25
   print("Value of X inside function1:",x)

x = 35
my_function1()
print("Value of X outside function1:",x)
```

```
Output
Value of X inside function: 25
Value of X outside function: 35
```

Here, we can see that the value of x is 35 initially. Even though the function function1() changed the value of x to 25, it did not affect the value outside the function. This is because the variable x inside the function is different (local to the function) from the one outside. Although they have same names, they are two different variables with different scope.

Scope and Lifetime of variables – Example 2

```
total = 0; # This is global variable.
# Function definition is here
def sum( arg1, arg2 ):
   # Add both the parameters and return them."
   total = arg1 + arg2; # Here total is local variable.
   print ("Inside the function local total : ", total);
   return total;
# Now you can call sum function
sum( 10, 20 );
print ("Outside the function global total : ", total)
```

OUTPUT

```
Inside the function local total :  30
Outside the function global total : 0
```

Global Variables

- Variables outside of the function are visible from inside.
- They have a global scope.
- Global variables can be accessed throughout the program body by all functions.
- We can read these values from inside the function but cannot change (write) them.
- In order to modify the value of global variables inside the function, They must be declared as global variables using the keyword global inside the function.

Global Variables – Example 1

```
Y=100;    # Y is a global variable
def function1():
   x = 25
   print("Value of X = ",x)
   print("Value of Y inside Function = ",Y)

function1()
print("Value of Y Outside Function = ",Y)
```

```
OUTPUT
Value of X =  25
Value of Y inside Function =  100
Value of Y Outside Function =  100
```

Global Variables – Example 2

```
Y=100;    # Y is a global variable
def function1():
   x = 25
   Y=200;
   print("Value of X = ",x)
   print("Value of Y inside Function = ",Y)

function1()
print("Value of Y Outside Function = ",Y)
```

```
OUTPUT
Value of X =  25
Value of Y inside Function =  200
Value of Y Outside Function =  100
```

Global Variables – Example 3

```
Y=100;    # Y is a global variable
def function1():
   global Y    # Y is declare as global inside the function
   x = 25
   Y=200;
   print("Value of X = ",x)
   print("Value of Y inside Function = ",Y)

function1()
print("Value of Y Outside Function = ",Y)
```

```
OUTPUT
Value of X =  25
Value of Y inside Function =  200
Value of Y Outside Function =  200
```

Function Arguments

- Functions can accept arguments

```
import os
os.system('cls')
def Welcome(name):
   """This function welcomes the person(name) passed in as Parameter """
   print("Hello, " + name + ". Welcome to Python functions!")
   print("Inside Function.....");
   print("End of Function.....");
   return ;
print("Outside Function........");

print(Welcome("ACCENTURE"));
```

Output

```
Outside Function........
Hello, ACCENTURE. Welcome to Python functions!
Inside Function.....
End of Function.....!
```

Variable Function Arguments

In the earlier slides, we saw that functions had fixed number of arguments.

In Python there are other ways to define a function which can take variable number of arguments.

There are three different forms

- **Default Arguments**
- **Keyword Arguments**
- **Arbitrary Arguments**

Default Arguments

- Function arguments can have default values in Python.
- Default value can be assigned to an argument by using the assignment operator (=).

```
def greet(name, msg = "Good morning!"):
   """This function greets to the person with the provided message.
   If message is not provided, it defaults to "Good morning!" """
   print("Hello",name + ', ' + msg)
```

Note: *In the above function, the parameter name does not have a default value and is required (mandatory) during a call.*
*But, the parameter **msg** has a default value of "Good morning!". So, it is optional during a function call. If a value is provided, it will overwrite the default value*

Default Arguments - Example

```
def greet(name, msg = "Good morning!"):
   """This function greets to the person with the provided message.
   If message is not provided, it defaults to "Good morning!" """
   print("Hello",name + ', ' + msg)
```

Function Call	OutPut
greet("Kate")	Hello Kate, Good morning!
greet("Bruce","How do you do?")	Hello Bruce, How do you do?

Default Arguments – Contd..

Any number of arguments in a function can have a default value. But once we have a default argument, all the arguments to its right must also have default values.

non-default arguments cannot follow default arguments.
For example, if we had defined the function header above as:
def greet(msg = "Good morning!", name):

We would get an error as:
SyntaxError: non-default argument follows default argument

Keyword Arguments

When we call a function with some values, these values get assigned to the arguments according to their position.

Example:

greet("Bruce","How do you do?")

The value "Bruce" gets assigned to the argument *name* and similarly "How do you do?" to *msg*.

Python allows functions to be called using keyword arguments.

When we call functions in this way, the order (position) of the arguments can be changed.

Keyword Arguments (Contd..)

Following calls to the above function are all valid and produce the same result.

```
greet(name = "Bruce",msg = "How do you do?")    # 2 keyword arguments
greet(msg = "How do you do?",name="Bruce") # 2 keyword arguments(out of order)
greet("Bruce",msg = "How do you do?")       # 1 positional, 1 keyword argument
```

we can mix positional arguments with keyword arguments during a function call.

But ***keyword arguments must follow positional arguments***.

Having a ***positional argument after keyword arguments will result into errors***.

Keyword Arguments (Contd..)

Example

```
greet(name="Bruce","How do you do?")
```

Will result into error as:

```
SyntaxError: non-keyword arg after keyword arg
```

Arbitrary Arguments

Python allows to call a function with arbitrary number of arguments.

To achieve that, an asterisk (*) must be used before the parameter name to denote this kind of argument

Arbitrary Arguments - Example

```
def greet(*names):
   """This function greets all the person in the names tuple."""

   # names is a tuple with arguments
   for name in names:
       print("Hello",name)

greet("Monica","Luke","Steve","John")
```

Note:
Here , greet function is called with multiple arguments. These arguments get wrapped up into a tuple before being passed into the function.

<u>**Output**</u>

```
Hello Monica
Hello Luke
Hello Steve
Hello John
```

Python Recursive Function

A Function which calls itself is called as a Recursive Function

This phenomenon is called as “Recursion”

```
def recur_fact(x):
   """This is a recursive function to find the factorial of an integer"""
   if x == 1:
       return 1
   else:
       return (x * recur_fact(x-1))

num = int(input("Enter a number: "))
if num >= 1:
   print("The factorial of ", num, “ is ", recur_fact(num))
```

Output
```
Enter a number: 4
The factorial of 4 is 24
```

Python Recursive Function (Contd)

Advantages	Disadvantages
Recursive functions make the code look clean and elegant.	Sometimes the logic behind recursion is hard to follow through.
A complex task can be broken down into simpler sub-problems using recursion.	Recursive calls are expensive (inefficient) as they take up a lot of memory and time
Sequence generation is easier with recursion than using some nested iteration.	Recursive functions are hard to debug

Note: Avoid infinite recursion

Python Anonymous/Lambda Function

- ➢ Python allows to define an anonymous function.
- ➢ Anonymous function is a function that is defined without a name.
- ➢ Normal functions are defined using the def keyword, but in Python anonymous functions are defined using the lambda keyword.
- ➢ Hence, anonymous functions are also called lambda functions.

Syntax: **lambda arguments: expression**

- Lambda functions can have any number of arguments but only one expression.
- The expression is evaluated and returned.
- Lambda functions can be used wherever function objects are required.

Python Anonymous/Lambda Function - Example

Example:

```
# Program to show the
# use of lambda functions
MultiplyBy2 = lambda x: x * 2
print(MultiplyBy2 (15))
```

Output

```
30
```

Explanation

In the above program,
lambda x: x * 2 is the lambda function.
Here *x* is the argument and x * 2 is the expression that gets evaluated and returned.
This function has no name. It returns a function object which is assigned to the identifier double. We can now call it as a normal function.
The statement `MultiplyBy2 = lambda x: x * 2`
is same as

```
def MultiplyBy2 (x):
    return x * 2
```

Use of Lambda Function

Use Lambda functions when you require a nameless function for a short period of time.
In Python, generally Lambda function is as an argument to a higher-order function (a function that takes in other functions as arguments).
Lambda functions are used along with built-in functions like filter(), map() etc.

Using Lambda Function with filter()

The filter() function in Python takes in a function and a list as arguments. The function is called with all the items in the list and a new list is returned which contains items for which the function evaluates to True.
Here is an example use of filter() function to filter out only even numbers from a list.

```
# Program to filter out
# only the even items from
# a list using filter() and
# lambda functions

my_list = [1, 5, 4, 6, 8, 11, 3, 12]

new_list = list(filter(lambda x: (x%2 == 0) , my_list))
print(new_list)
```

Output
```
[4, 6, 8, 12]
```

Using Lambda Function with map()

The map() function in Python takes in a function and a list.
The function is called with all the items in the list and a new list is returned which contains items returned by that function for each item.
Here is an example use of map() function to double all the items in a list.

```
# Program to double each
# item in a list using map() and
# lambda functions

my_list = [1, 5, 4, 6, 8, 11, 3, 12]

new_list = list(map(lambda x: x * 2 , my_list))
print(new_list)
```

Output

```
[2, 10, 8, 12, 16, 22, 6, 24]
```

Built-in Functions in Python

Function	Description
abs()	Return the absolute value of a number.
all()	Return True if all elements of the iterable are true (or if the iterable is empty).
any()	Return True if any element of the iterable is true. If the iterable is empty, return False.
ascii()	Return a string containing a printable representation of an object, but escape the non-ASCII characters.
bin()	Convert an integer number to a binary string.
bool()	Convert a value to a Boolean.
bytearray()	Return a new array of bytes.
bytes()	Return a new "bytes" object.
callable()	Return True if the object argument appears callable, False if not.
chr()	Return the string representing a character.
classmethod()	Return a class method for the function.
compile()	Compile the source into a code or AST object.
complex()	Create a complex number or convert a string or number to a complex number.
delattr()	Deletes the named attribute of an object.
dict()	Create a new dictionary.

Built-in Functions in Python

Function	Description
divmod()	Return a pair of numbers consisting of quotient and remainder when using integer division.
enumerate()	Return an enumerate object.
eval()	The argument is parsed and evaluated as a Python expression.
exec()	Dynamic execution of Python code.
filter()	Construct an iterator from elements of iterable for which function returns true.
float()	Convert a string or a number to floating point.
format()	Convert a value to a "formatted" representation.
frozenset()	Return a new frozenset object.
getattr()	Return the value of the named attribute of an object.
globals()	Return a dictionary representing the current global symbol table.
hasattr()	Return True if the name is one of the object's attributes.
hash()	Return the hash value of the object.
help()	Invoke the built-in help system.
hex()	Convert an integer number to a hexadecimal string.
id()	Return the "identity" of an object.

Built-in Functions in Python

Function	Description
input()	Reads a line from input, converts it to a string (stripping a trailing newline), and returns that.
int()	Convert a number or string to an integer.
isinstance()	Return True if the object argument is an instance.
issubclass()	Return True if class is a subclass.
iter()	Return an iterator object.
len()	Return the length (the number of items) of an object.
list()	Return a list.
locals()	Update and return a dictionary representing the current local symbol table.
map()	Return an iterator that applies function to every item of iterable, yielding the results.
max()	Return the largest item in an iterable.
memoryview()	Return a "memory view" object created from the given argument.
min()	Return the smallest item in an iterable.
next()	Retrieve the next item from the iterator.
object()	Return a new featureless object.
oct()	Convert an integer number to an octal string.

Built-in Functions in Python

Function	Description
open()	Open file and return a corresponding file object.
ord()	Return an integer representing the Unicode.
pow()	Return power raised to a number.
print()	Print objects to the stream.
property()	Return a property attribute.
range()	Return an iterable sequence.
repr()	Return a string containing a printable representation of an object.
reversed()	Return a reverse iterator.
round()	Return the rounded floating point value.
set()	Return a new set object.
setattr()	Assigns the value to the attribute.
slice()	Return a slice object.
sorted()	Return a new sorted list.
staticmethod()	Return a static method for function.
str()	Return a str version of object.

Built-in Functions in Python

Function	Description
sum()	Sums the items of an iterable from left to right and returns the total.
super()	Return a proxy object that delegates method calls to a parent or sibling class.
tuple()	Return a tuple
type()	Return the type of an object.
vars()	Return the __dict__ attribute for a module, class, instance, or any other object.
zip()	Make an iterator that aggregates elements from each of the iterables.
__import__()	This function is invoked by the import statement.

Numeric Functions in Python - Type Conversion

Function	Meaning
int(x)	convert x to a plain integer
long(x)	convert x to a long integer
float(x)	convert x to a floating-point number
complex(x)	convert x to a complex number with real part x and imaginary part zero.
complex(x, y)	convert x and y to a complex number with real part x and imaginary part y. x and y are numeric expressions

Numeric Functions in Python - Mathematical

Function	Description (Returns)
abs(x)	The absolute value of x: the (positive) distance between x and zero.
ceil(x)	The ceiling of x: the smallest integer not less than x
cmp(x, y)	-1 if x < y, 0 if x == y, or 1 if x > y
exp(x)	The exponential of x: e^x
fabs(x)	The absolute value of x.
floor(x)	The floor of x: the largest integer not greater than x
log(x)	The natural logarithm of x, for x> 0
log10(x)	The base-10 logarithm of x for x> 0 .
max(x1, x2,...)	The largest of its arguments: the value closest to positive infinity
min(x1, x2,...)	The smallest of its arguments: the value closest to negative infinity
modf(x)	The fractional and integer parts of x in a two-item tuple. Both parts have the same sign as x. The integer part is returned as a float.
pow(x, y)	The value of x**y.
round(x [,n])	x rounded to n digits from the decimal point. Python rounds away from zero as a tie-breaker: round(0.5) is 1.0 and round(-0.5) is -1.0.
sqrt(x)	The square root of x for x > 0

Numeric Functions in Python - Random Number Functions

Random numbers are used for games, simulations, testing, security, and privacy applications. Python includes following functions that are commonly used.

Function	Description
choice(seq)	A random item from a list, tuple, or string.
randrange ([start,] stop [,step])	A randomly selected element from range(start, stop, step)
random()	A random float r, such that 0 is less than or equal to r and r is less than 1
seed([x])	Sets the integer starting value used in generating random numbers. Call this function before calling any other random module function. Returns None.
shuffle(lst)	Randomizes the items of a list in place. Returns None.
uniform(x, y)	A random float r, such that x is less than or equal to r and r is less than y

Numeric Functions in Python - Trigonometric

Function	Description
acos(x)	Return the arc cosine of x, in radians.
asin(x)	Return the arc sine of x, in radians.
atan(x)	Return the arc tangent of x, in radians.
atan2(y, x)	Return atan(y / x), in radians.
cos(x)	Return the cosine of x radians.
hypot(x, y)	Return the Euclidean norm, sqrt(x*x + y*y).
sin(x)	Return the sine of x radians.
tan(x)	Return the tangent of x radians.
degrees(x)	Converts angle x from radians to degrees.
radians(x)	Converts angle x from degrees to radians.

Numeric Functions in Python - Mathematical Constants

Constants	Description
pi	The mathematical constant pi.
e	The mathematical constant e.

String functions in Python

Function	Description
capitalize()	Capitalizes first letter of string
center(width, fillchar)	Returns a space-padded string with the original string centered to a total of width columns.
count(str, beg= 0,end=len(string))	Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given.
decode(encoding='UTF-8',errors='strict')	Decodes the string using the codec registered for encoding. encoding defaults to the default string encoding.
encode(encoding='UTF-8',errors='strict')	Returns encoded string version of string; on error, default is to raise a ValueError unless errors is given with 'ignore' or 'replace'.
endswith(suffix, beg=0, end=len(string))	Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise.
expandtabs(tabsize=8)	Expands tabs in string to multiple spaces; defaults to 8 spaces per tab if tabsize not provided.
find(str, beg=0 end=len(string))	Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.
index(str, beg=0, end=len(string))	Same as find(), but raises an exception if str not found.
isalnum()	Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.
isalpha()	Returns true if string has at least 1 character and all characters are alphabetic and false otherwise.

String functions in Python

Function	Description
isdigit()	Returns true if string contains only digits and false otherwise.
islower()	Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.
isnumeric()	Returns true if a unicode string contains only numeric characters and false otherwise.
isspace()	Returns true if string contains only whitespace characters and false otherwise.
istitle()	Returns true if string is properly "titlecased" and false otherwise.
isupper()	Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.
join(seq)	Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string.
len(string)	Returns the length of the string
ljust(width[, fillchar])	Returns a space-padded string with the original string left-justified to a total of width columns.
lower()	Converts all uppercase letters in string to lowercase.
lstrip()	Removes all leading whitespace in string.
maketrans()	Returns a translation table to be used in translate function.

String functions in Python

Function	Description
max(str)	Returns the max alphabetical character from the string str.
min(str)	Returns the min alphabetical character from the string str.
replace(old, new [, max])	Replaces all occurrences of old in string with new or at most max occurrences if max given.
rfind(str, beg=0,end=len(string))	Same as find(), but search backwards in string
rindex(str, beg=0, end=len(string))	Same as index(), but search backwards in string.
rjust(width,[, fillchar])	Returns a space-padded string with the original string right-justified to a total of width columns.
rstrip()	Removes all trailing whitespace of string.
split(str="", num=string.count(str))	Splits string according to delimiter str (space if not provided) and returns list of substrings; split into at most num substrings if given.
splitlines(num=string.count('\n'))	Splits string at all (or num) NEWLINEs and returns a list of each line with NEWLINEs removed.
startswith(str, beg=0,end=len(string))	Determines if string or a substring of string (if starting index beg and ending index end are given) starts with substring str; returns true if so and false otherwise.
strip([chars])	Performs both lstrip() and rstrip() on string
swapcase()	Inverts case for all letters in string.
title()	Returns "titlecased" version of string, that is, all words begin with uppercase and the rest are lowercase.

String functions in Python

Function	Description
translate(table, deletechars="")	Translates string according to translation table str(256 chars), removing those in the del string
upper()	Converts lowercase letters in string to uppercase.
zfill (width)	Returns original string leftpadded with zeros to a total of width characters; intended for numbers, zfill() retains any sign given (less one zero).
isdecimal()	Returns true if a unicode string contains only decimal characters and false otherwise

Questions