What is a function in Python?

In Python, a function is a group of related statements that performs a specific task.

Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

Furthermore, it avoids repetition and makes the code reusable.

Syntax of Function:

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

Above shown is a function definition that consists of the following components.

Keyword def that marks the start of the function header.

A function name to uniquely identify the function. Function naming follows the same rules of writing identifiers in Python.

Parameters (arguments) through which we pass values to a function. They are optional.

A colon (:) to mark the end of the function header.

Optional documentation string (docstring) to describe what the function does. One or more valid python statements that make up the function body. Statements must have the same indentation level (usually 4 spaces).

An optional return statement to return a value from the function.

```
def greet(name):
    """
    This function greets to
    the person passed in as
    a parameter
    """
    print("Hello, " + name + ". Good morning!")
greet('Rahul')
#print(greet.__doc__)
```

```
Hello, Rahul. Good morning!
```

```
def is_even(number):
    """
    This fuction tells given number is even or odd
    Input-any valid intiger
    Output- odd/even
    Created by- Dr Safdar sardar Khan
    Last edited -17/9/2024
    """
```

```
    if number % 2 == 0:
        return "Even"
    else:
        return "odd"

for i in range(1,11):
    print(is_even(i))
print(is_even.__doc__)
```

```
odd
Even
odd
Even
odd
Even
odd
Even
odd
Even
```

```
    This fuction tells given number is even or odd
    Input-any valid intiger
    Output- odd/even
    Created by- Dr Safdar sardar Khan
    Last edited -17/9/2024
```

```
print.__doc__
```

```
"print(value, ..., sep=' ', end='\\n', file=sys.stdout, flush=False)\
n\nPrints the values to a stream, or to sys.stdout by default.\
nOptional keyword arguments:\nfile:  a file-like object (stream);
defaults to the current sys.stdout.\nsep:   string inserted between
values, default a space.\nend:   string appended after the last value,
default a newline.\nflush: whether to forcibly flush the stream."
```

```
type.__doc__
```

```
"type(object_or_name, bases, dict)\ntype(object) -> the object's type\
ntype(name, bases, dict) -> a new type"
```

```
print(is_even.__doc__)
```

```
    This fuction tells given number is even or odd
    Input-any valid intiger
    Output- odd/even
    Created by- Dr Safdar sardar Khan
    Last edited -17/9/2024
```

```
def absolute_value(num):
    """This function returns the absolute
    value of the entered number"""

    if num >= 0:
        return num
    else:
        return -num


print(absolute_value(2))

print(absolute_value(-4))
```

```
2
4
```

```
def my_func():
     x = 10
     print("Value inside function:",x)

x = 20
my_func()
print("Value outside function:",x)
```

```
Value inside function: 10
Value outside function: 20
```

```
def vote(num):
    if num >= 18:
        print("Eligible for vote")
    else:
        print("Not eligible for vote")

print(vote(17))
```

```
Not eligible for vote
None
```

What is recursion?

Recursion is the process of defining something in terms of itself.

A physical world example would be to place two parallel mirrors facing each other. Any object in between them would be reflected recursively.

```
def factorial(x):
    """This is a recursive function
    to find the factorial of an integer"""

    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))


num = 3
print("The factorial of", num, "is", factorial(num))
```

```
The factorial of 3 is 6
```

factorial(3) # 1st call with 3 3 * factorial(2) # 2nd call with 2 3 * 2 * factorial(1) # 3rd call with 1 3 * 2 * 1 # return from 3rd call as number=1 3 * 2 # return from 2nd call 6 # return from 1st call

Advantages of Recursion

Recursive functions make the code look clean and elegant. A complex task can be broken down into simpler sub-problems using recursion. Sequence generation is easier with recursion than using some nested iteration.

Disadvantages of Recursion

Sometimes the logic behind recursion is hard to follow through. Recursive calls are expensive (inefficient) as they take up a lot of memory and time. Recursive functions are hard to debug.

What are lambda functions in Python?

In Python, an anonymous function is a function that is defined without a name.

While normal functions are defined using the def keyword in Python, anonymous functions are defined using the lambda keyword.

Hence, anonymous functions are also called lambda functions.

How to use lambda Functions in Python?

A lambda function in python has the following syntax.

Syntax of Lambda Function in python

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.

```
# Program to show the use of lambda functions
double = lambda x: x * 3

print(double(5))

15
```

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

double = lambda x: x * 2 is nearly the same as:

def double(x): return x * 2

Use of Lambda Function in python

We use lambda functions when we require a nameless function for a short period of time.

In Python, we generally use it 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.

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

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

print(new_list)

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

Example use 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()

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

new_list = list(map(lambda x: x + 3 , my_list))

print(new_list)
```

```
[4, 8, 7, 9, 11, 14, 6, 15]
```

Global Variables

In Python, a variable declared outside of the function or in global scope is known as a global variable. This means that a global variable can be accessed inside or outside of the function.

Let's see an example of how a global variable is created in Python.

```
x = "global"

def foo():
    print("x inside:", x)


foo()
print("x outside:", x)
```

```
x inside: global
x outside: global
```

Local Variables

A variable declared inside the function's body or in the local scope is known as a local variable.

```
def foo():
    y = "local"
```

```
foo()
print(y)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
C:\PROGRA~1\KMSpico\temp/ipykernel_9904/1777326041.py in <module>
      4
      5 foo()
----> 6 print(y)

NameError: name 'y' is not defined
```

```
def foo():
    y = "local"
    print(y)

foo()
```

```
local
```

```
#Example 5: Global variable and Local variable with same name

x = 5

def foo():
    x = 10
    print("local x:", x)


foo()
print("global x:", x)
```

```
local x: 10
global x: 5
```

What is the global keyword

In Python, global keyword allows you to modify the variable outside of the current scope. It is used to create a global variable and make changes to the variable in a local context.

Rules of global Keyword:

The basic rules for global keyword in Python are:

When we create a variable inside a function, it is local by default. When we define a variable outside of a function, it is global by default. You don't have to use global keyword. We use global keyword to read and write a global variable inside a function. Use of global keyword outside a function has no effect.

```
c = 1 # global variable

def add():
    c = c + 2 # increment c by 2
    print(c)

add()

#This is because we can only access the global variable but cannot
modify it from inside the function.

#The solution for this is to use the global keyword.
```

```
---------------------------------------------------------------------
-----
UnboundLocalError                         Traceback (most recent call
last)
C:\PROGRA~1\KMSpico\temp/ipykernel_9904/1708956339.py in <module>
      5     print(c)
      6
----> 7 add()
      8
      9 #This is because we can only access the global variable but
cannot modify it from inside the function.

C:\PROGRA~1\KMSpico\temp/ipykernel_9904/1708956339.py in add()
      2
      3 def add():
----> 4     c = c + 2 # increment c by 2
      5     print(c)
      6

UnboundLocalError: local variable 'c' referenced before assignment
```

```
c = 4 # global variable

def add():
    global c
    c = c + 2 # increment by 2
    print("Inside add():", c)

add()
print("In main:", c)
```

```
Inside add(): 6
In main: 6
```

```
#Example 5: Using a Global Variable in Nested Function
def foo():
    x = 20
```

```
    def bar():
        global x
        x = 25
        print("After calling bar: ", x)

    print("Before calling bar: ", x)
    print("Calling bar now")
    bar()
    print("After calling bar: ", x)

foo()
print("x in main: ", x)
```

```
Before calling bar:  20
Calling bar now
After calling bar:  25
After calling bar:  20
x in main:  25
```

```
#Python import statement
#We can import a module using the import statement and access the
definitions inside it using the dot operator as described above. Here
is an example.

# import statement example
# to import standard module math

import math
print("The value of pi is", math.pi)
```

```
The value of pi is 3.141592653589793
```

```
# import only pi from math module

from math import pi
print("The value of pi is", pi)
```

```
The value of pi is 3.141592653589793
```

Python Built-in Functions

Python has several functions that are readily available for use. These functions are called built-in functions. On this reference page, you will find all the built-in functions in Python.

abs()

The abs() function returns the absolute value of the given number. If the number is a complex number, abs() returns its magnitude.

Example

```
number = -20

absolute_number = abs(number)
print(absolute_number)

# Output: 20
```

```
20
```

```
# random integer
integer = -20
print('Absolute value of -20 is:', abs(integer))

#random floating number
floating = -30.333333333
print('Absolute value of -30.33 is:', abs(floating))
```

```
Absolute value of -20 is: 20
Absolute value of -30.33 is: 30.333333333
```

all()

The all() function returns True if all elements in the given iterable are true. If not, it returns False.

Example boolean_list = ['True', 'True', 'True']

check if all elements are true

result = all(boolean_list) print(result)

Output: True

```
boolean_list = ['True', 'false', 'True']

# check if all elements are true
result = all(boolean_list)
print(result)
```

```
# Output: True
```

```
True
```

```
# all values true
l = [1, 3, 4, 5]
print(all(l))

# all values false
l = [0, False]
print(all(l))

# one false value
l = [1, 3, 4, 0]
print(all(l))

# one true value
l = [0, False, 5]
print(all(l))

# empty iterable
l = []
print(all(l))
```

```
True
False
False
False
True
```

```
s = "This is good"
print(all(s))

# 0 is False
# '0' is True
s = '000'
print(all(s))

s = ''
print(all(s))
```

```
True
True
True
```

ascii()

The ascii() method replaces a non-printable character with its corresponding ascii value and returns it.

Example text = 'Pythön is interesting'

replace ö with its ascii value

print(ascii(text))

Output: 'Pyth\xf6n is interesting'

```
text = 'Pythön is interesting'
print(ascii(text))
```

```
'Pyth\xf6n is interesting'
```

```
text1 = '√ represents square root'

# replace √ with ascii value
print(text1)
print(ascii(text1))

text2 = 'Thör is coming'

# replace ö with ascii value
print(ascii(text2))
```

```
√ represents square root
'\u221a represents square root'
'Th\xf6r is coming'
```

```
set = {'Π', 'Φ', 'η'}

# ascii() with a set
print(ascii(set))
```

```
{'\u03a6', '\u03b7', '\u03a0'}
```

str()

The str() method returns the string representation of a given object.

Example

string representation of Adam

print(str('Adam'))

Output: Adam

```
# string representation of Luke
name = str('Luke')
print(name)

# string representation of an integer 40
age = str(40)
print(age)
type(age)

# string representation of a numeric string 7ft
height = str('7ft')
print(height)
```

```
Luke
40
7ft
```

```
ag = str(10)
print(ag)
type(ag)
```

```
10
```

```
str
```

Python vars()

In this tutorial, you will learn about Python vars() with the help of examples.

The vars() method returns the **dict** (dictionary mapping) attribute of the given object.

```
# returns __dict__ of a list
print(vars(list))
```

```
{'__repr__': <slot wrapper '__repr__' of 'list' objects>, '__hash__':
None, '__getattribute__': <slot wrapper '__getattribute__' of 'list'
objects>, '__lt__': <slot wrapper '__lt__' of 'list' objects>,
'__le__': <slot wrapper '__le__' of 'list' objects>, '__eq__': <slot
wrapper '__eq__' of 'list' objects>, '__ne__': <slot wrapper '__ne__'
of 'list' objects>, '__gt__': <slot wrapper '__gt__' of 'list'
objects>, '__ge__': <slot wrapper '__ge__' of 'list' objects>,
```

```
'__iter__': <slot wrapper '__iter__' of 'list' objects>, '__init__':
<slot wrapper '__init__' of 'list' objects>, '__len__': <slot wrapper
'__len__' of 'list' objects>, '__getitem__': <method '__getitem__' of
'list' objects>, '__setitem__': <slot wrapper '__setitem__' of 'list'
objects>, '__delitem__': <slot wrapper '__delitem__' of 'list'
objects>, '__add__': <slot wrapper '__add__' of 'list' objects>,
'__mul__': <slot wrapper '__mul__' of 'list' objects>, '__rmul__':
<slot wrapper '__rmul__' of 'list' objects>, '__contains__': <slot
wrapper '__contains__' of 'list' objects>, '__iadd__': <slot wrapper
'__iadd__' of 'list' objects>, '__imul__': <slot wrapper '__imul__' of
'list' objects>, '__new__': <built-in method __new__ of type object at
0x00007FFA16A20AF0>, '__reversed__': <method '__reversed__' of 'list'
objects>, '__sizeof__': <method '__sizeof__' of 'list' objects>,
'clear': <method 'clear' of 'list' objects>, 'copy': <method 'copy' of
'list' objects>, 'append': <method 'append' of 'list' objects>,
'insert': <method 'insert' of 'list' objects>, 'extend': <method
'extend' of 'list' objects>, 'pop': <method 'pop' of 'list' objects>,
'remove': <method 'remove' of 'list' objects>, 'index': <method
'index' of 'list' objects>, 'count': <method 'count' of 'list'
objects>, 'reverse': <method 'reverse' of 'list' objects>, 'sort':
<method 'sort' of 'list' objects>, '__class_getitem__': <method
'__class_getitem__' of 'list' objects>, '__doc__': 'Built-in mutable
sequence.\n\nIf no argument is given, the constructor creates a new
empty list.\nThe argument must be an iterable if specified.'}
```

```
string = "Jones"

# vars() with a string
print(vars(string))
```

```
---------------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
C:\PROGRA~1\KMSpico\temp/ipykernel_9904/4213680531.py in <module>
      2
      3 # vars() with a string
----> 4 print(vars(string))

TypeError: vars() argument must have __dict__ attribute
```

```
print(vars(str))
```

```
{'__repr__': <slot wrapper '__repr__' of 'str' objects>, '__hash__':
<slot wrapper '__hash__' of 'str' objects>, '__str__': <slot wrapper
'__str__' of 'str' objects>, '__getattribute__': <slot wrapper
'__getattribute__' of 'str' objects>, '__lt__': <slot wrapper '__lt__'
of 'str' objects>, '__le__': <slot wrapper '__le__' of 'str' objects>,
'__eq__': <slot wrapper '__eq__' of 'str' objects>, '__ne__': <slot
wrapper '__ne__' of 'str' objects>, '__gt__': <slot wrapper '__gt__'
```

```
of 'str' objects>, '__ge__': <slot wrapper '__ge__' of 'str' objects>,
'__iter__': <slot wrapper '__iter__' of 'str' objects>, '__mod__':
<slot wrapper '__mod__' of 'str' objects>, '__rmod__': <slot wrapper
'__rmod__' of 'str' objects>, '__len__': <slot wrapper '__len__' of
'str' objects>, '__getitem__': <slot wrapper '__getitem__' of 'str'
objects>, '__add__': <slot wrapper '__add__' of 'str' objects>,
'__mul__': <slot wrapper '__mul__' of 'str' objects>, '__rmul__':
<slot wrapper '__rmul__' of 'str' objects>, '__contains__': <slot
wrapper '__contains__' of 'str' objects>, '__new__': <built-in method
__new__ of type object at 0x00007FFA16A24140>, 'encode': <method
'encode' of 'str' objects>, 'replace': <method 'replace' of 'str'
objects>, 'split': <method 'split' of 'str' objects>, 'rsplit':
<method 'rsplit' of 'str' objects>, 'join': <method 'join' of 'str'
objects>, 'capitalize': <method 'capitalize' of 'str' objects>,
'casefold': <method 'casefold' of 'str' objects>, 'title': <method
'title' of 'str' objects>, 'center': <method 'center' of 'str'
objects>, 'count': <method 'count' of 'str' objects>, 'expandtabs':
<method 'expandtabs' of 'str' objects>, 'find': <method 'find' of
'str' objects>, 'partition': <method 'partition' of 'str' objects>,
'index': <method 'index' of 'str' objects>, 'ljust': <method 'ljust'
of 'str' objects>, 'lower': <method 'lower' of 'str' objects>,
'lstrip': <method 'lstrip' of 'str' objects>, 'rfind': <method 'rfind'
of 'str' objects>, 'rindex': <method 'rindex' of 'str' objects>,
'rjust': <method 'rjust' of 'str' objects>, 'rstrip': <method 'rstrip'
of 'str' objects>, 'rpartition': <method 'rpartition' of 'str'
objects>, 'splitlines': <method 'splitlines' of 'str' objects>,
'strip': <method 'strip' of 'str' objects>, 'swapcase': <method
'swapcase' of 'str' objects>, 'translate': <method 'translate' of
'str' objects>, 'upper': <method 'upper' of 'str' objects>,
'startswith': <method 'startswith' of 'str' objects>, 'endswith':
<method 'endswith' of 'str' objects>, 'removeprefix': <method
'removeprefix' of 'str' objects>, 'removesuffix': <method
'removesuffix' of 'str' objects>, 'isascii': <method 'isascii' of
'str' objects>, 'islower': <method 'islower' of 'str' objects>,
'isupper': <method 'isupper' of 'str' objects>, 'istitle': <method
'istitle' of 'str' objects>, 'isspace': <method 'isspace' of 'str'
objects>, 'isdecimal': <method 'isdecimal' of 'str' objects>,
'isdigit': <method 'isdigit' of 'str' objects>, 'isnumeric': <method
'isnumeric' of 'str' objects>, 'isalpha': <method 'isalpha' of 'str'
objects>, 'isalnum': <method 'isalnum' of 'str' objects>,
'isidentifier': <method 'isidentifier' of 'str' objects>,
'isprintable': <method 'isprintable' of 'str' objects>, 'zfill':
<method 'zfill' of 'str' objects>, 'format': <method 'format' of 'str'
objects>, 'format_map': <method 'format_map' of 'str' objects>,
'__format__': <method '__format__' of 'str' objects>, 'maketrans':
<staticmethod object at 0x000001C1B51D4640>, '__sizeof__': <method
'__sizeof__' of 'str' objects>, '__getnewargs__': <method
'__getnewargs__' of 'str' objects>, '__doc__': "str(object='') -> str\
nstr(bytes_or_buffer[, encoding[, errors]]) -> str\n\nCreate a new
```

```
string object from the given object. If encoding or\nerrors is
specified, then the object must expose a data buffer\nthat will be
decoded using the given encoding and error handler.\nOtherwise,
returns the result of object.__str__() (if defined)\nor repr(object).\
nencoding defaults to sys.getdefaultencoding().\nerrors defaults to
'strict'."}
```

exec()

The exec() method executes a dynamically created program, which is either a string or a code object.

```
program = 'a = 5\nb=10\nprint("Sum =", a+b)'
exec(program)

# Output: Sum = 15
```

```
Sum = 15
```

```
program = 'a = 5\nb=10\nprint("Sum =", a+b)'
print(program)

# Output: Sum = 15
```

```
a = 5
b=10
print("Sum =", a+b)

# get a multi-line program as input
program = input('Enter a program:')

# compile the program in execution mode
b = compile(program, 'something', 'exec')

# execute the program
exec(b)
```

```
---------------------------------------------------------------------
-----
KeyboardInterrupt                         Traceback (most recent call
last)
C:\PROGRA~1\KMSpico\temp/ipykernel_9904/2490291651.py in <module>
      1 # get a multi-line program as input
----> 2 program = input('Enter a program:')
      3
      4 # compile the program in execution mode
      5 b = compile(program, 'something', 'exec')

~\anaconda3\lib\site-packages\ipykernel\kernelbase.py in
```

```
raw_input(self, prompt)
   1004                 "raw_input was called, but this frontend does
not support input requests."
   1005             )
-> 1006         return self._input_request(
   1007             str(prompt),
   1008             self._parent_ident["shell"],

~\anaconda3\lib\site-packages\ipykernel\kernelbase.py in
_input_request(self, prompt, ident, parent, password)
   1049             except KeyboardInterrupt:
   1050                 # re-raise KeyboardInterrupt, to truncate
traceback
-> 1051                 raise KeyboardInterrupt("Interrupted by user")
from None
   1052             except Exception:
   1053                 self.log.warning("Invalid Message:",
exc_info=True)

KeyboardInterrupt: Interrupted by user
```

globals()

The globals() method returns a dictionary with all the global variables and symbols for the current program.

Example print(globals())

```
print(globals())

age = 333

globals()['age'] = 40
print('The age is:', age)

The age is: 40
```

eval()

The eval() method parses the expression passed to this method and runs python expression (code) within the program.

```
number = 9

# eval performs the multiplication passed as argument
square_number = eval('number * number')
```

```
print(square_number)

# Output: 81
```

```
81
```

```
from math import *
names = {'square_root': sqrt, 'power': pow}
print(eval('dir()', names))

# Using square_root in Expression
print(eval('square_root(9)', names))
print(eval('power(2,3)', names))
```

```
['__builtins__', 'power', 'square_root']
3.0
8.0
```