Double-click (or enter) to edit

# FUNCTIONS IN PYTHON

## BUILT-IN FUNCTIONS

```
print(abs(-67))
print(sorted([2,4,1,2]))
```

```
67
[1, 2, 2, 4]
```

## map() function

```
num = [1,2,3,4,5]
str_num = list(map(str, num))
print(str_num)
```

```
['1', '2', '3', '4', '5']
```

```
strings = ["apple", "mango", "banana"]
length = list(map(len, strings))
print(length)
```

```
[5, 5, 6]
```

```
words = ["hello", "world", "again" ]
capital = list(map(str.capitalize, words))
print(capital)
```

```
['Hello', 'World', 'Again']
```

## LAMBDA FUNCTIONS

```
multiply = lambda x, y : x*y
print(multiply(7,8))
```

```
56
```

```
numbers = [1,2,3,4]
squared_num = list(map(lambda x : x**2, numbers))
print(squared_num)
```

```
[1, 4, 9, 16]
```

```
numbers = [1,2,3,4,5,6,7,8,9,10]
even_num = list(filter(lambda x : x % 2 == 0, numbers))
print(even_num)
```

```
[2, 4, 6, 8, 10]
```

```
tuples = [(1, "one"), (2, "two"), (3, "three"), (4, "four")]
sorted_tuple = sorted(tuples, key=lambda x : x[1])
print(sorted_tuple)
```

```
[(4, 'four'), (1, 'one'), (3, 'three'), (2, 'two')]
```

## USER-DEFINES FUNCTIONS

```
#syntax of function
def function_name(parameters) :
  return value
```

```
def is_even(number) :
  if(number % 2 == 0) :
    return True
  else :
    return False
print(is_even(5))
```

```
False
```

```
def addTwonum(a, b) :
  return a+b
print(addTwonum(1,2))
```

```
3
```

```
def isPrime(num) :
  if(num == 1) :
    return False
  for i in range(2, num) :
    if(num % i == 0) :
      return False
  return True
print(isPrime(35))
```

```
False
```

```
def greet() :
  print("hello")
greet()
```

```
hello
```

```
def power(num, exponent=2) : #by default exponent is 2
  return num ** exponent
power(2, 3)
power(3)
```

```
9
```

```
def calculate_average(*numbers) : #function with variable arguments , can have multiple arguments with one parame
  return sum(numbers) / len(numbers) if numbers else 0
calculate_average(2,3,4,5,6)
```

```
4.0
```

```
def printthis(**details) : # function with key, value pairs by use of **
  for key,value in details.items() :
    print(f"{key} : {value}")
printthis(name = "tanishak", city = "delhi", age = 20)
```

```
name : tanishak
city : delhi
age : 20
```

## ERRORS

1. Syntax error
2. indentation error
3. Name error
4. Type error
5. Index error
6. key error
7. Attribute error
8. Zero division error

syntax error are handles by our own

indentation errors are handles by our own

type error

```
try :
  result = 5 + "2"
except:
  print("there should be a type error")
```

```
there should be a type error
```

```
try:
  list1 = [1,2,3]
  print(list1[3])
except:
  print("there must be a index error")
```

```
there must be a index error
```

```
a = 10
b = 0
try:
  a / b
except:
  print("you cannot divide by zero")
```

```
you cannot divide by zero
```

## NUMPY

```
pip install numpy
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.0.2)
```

```
import numpy as np
import timeit
#Test function for list
def test_list() :
  large_list = list(range(1000000))
  sum(large_list)
#Test function for numpy array
def test_array() :
  large_array = np.arange(1000000)
  np.sum(large_array)
#measure  time for list
list_time = timeit.timeit(test_list, number=100)
print(f"Average time taken by python list: {list_time/100} seconds")

array_time = timeit.timeit(test_array, number=100)
print(f"Average time taken by numpy array: {array_time/100} seconds")
```

```
Average time taken by python list: 0.051259403259999774 seconds
Average time taken by numpy array: 0.0008840752799994789 seconds
```

```
import numpy as np
a = np.array([1,2,3,4,5])
type(a)
```

```
numpy.ndarray
```

```
import numpy as np
a = [1,2,3,4,5]
np.array(a)
```

```
array([1, 2, 3, 4, 5])
```

## Create a 2D array

```
arr = np.array([[1,2,3,4,5],[2,3,4,5,6]])
print(arr)
arr.ndim
#slicing
arr[0:2,1:3]
```

```
[[1 2 3 4 5]
 [2 3 4 5 6]]
array([[2, 3],
       [3, 4]])
```

```
arr = np.array([[[1,2,3,4,5],[2,3,4,5,6]],[[1,2,3,4,5],[2,3,4,5,6]],[[1,2,3,4,5],[2,3,4,5,6]]])
print(arr) #print the array
print(arr.shape) #print the shape of the array like (2,3,3) => (no. of 2d array, rows, cols)
arr.ndim # print the dimension of the array like 1d 2d 3d
#slicing
arr[0:,0:1,1:3]
#conditional slicing
arr[(arr == 2) | (arr < 5)]
```

```
arr[2:,0:,2]
arr.dtype
```

```
[[[1 2 3 4 5]
  [2 3 4 5 6]]

 [[1 2 3 4 5]
  [2 3 4 5 6]]

 [[1 2 3 4 5]
  [2 3 4 5 6]]]
(3, 2, 5)
dtype('int64')
```

## Functions of NP array

```
##arange
arr = np.arange(1,11,2)
arr
#linspace
arr_1 = np.linspace(10,100,num=18).astype("int")
# arr_1
#reshape : reshape the array
# print(arr_1.reshape(2,3,3))
arr_1 = np.linspace(10,100,num=27).astype("int")
print(arr_1.reshape(3,3,3))
```

```
[[[ 10  13  16]
  [ 20  23  27]
  [ 30  34  37]]

 [[ 41  44  48]
  [ 51  55  58]
  [ 61  65  68]]

 [[ 72  75  79]
  [ 82  86  89]
  [ 93  96 100]]]
```