

Functions as First-Class Data Objects

- ✓ In python functions can be treated as **first-class data objects**.
- ✓ It means that they can be assigned to a **variables**, **passed arguments to other functions**, **returned as the values of other functions** and **stored in data structures such as lists and dictionaries**.

```
# https://docs.python.org/3.9/library/functions.html#abs
```

Ex1

```
# a is an alias for abs
```

```
a = abs
```

```
print(a) # <built-in function abs>
```

```
print(type(a)) # <class 'builtin_function_or_method'>
```

Ex2

```
import math
```

```
m = math.sqrt
```

```
print(m) # <built-in function sqrt>
```

```
print(type(m)) # <class 'builtin_function_or_method'>
```

abs() function

- ✓ Passing **int/float/complex** value to **abs() function**
- ✓ In Python **abs() function** will take only one argument and return the value of a number, i.e, it will remove the negative sign of a number

Syntax: **abs(number)**

number: Can be an integer, a floating-point

Note: The **abs() function** make any negative number to positive, while positive numbers are unaffected.

```
Ex3
# Applying a to an argument
a = abs
b = a(-4)
print('Integer: ', b)    # 4

c = abs
d = c(-4.0)
print('Float: ', d)     # 4.0

# Applying the functions in a list
lst = [a, m]
print(lst) # [<built-in function abs>, <built-in function sqrt>]
print(lst[0](2))    # 2
print(lst[1](2))    # 1.4142135623730951
# print(lst[2](2)) # IndexError: list index out of range
```

```
Ex4
# Return func as argument
import math

def d1(argOne, argTwo):
    return argOne(argTwo)

e = d1(abs, -4)
print(e)    # 4

e = d1(math.sqrt, 2)
print(e)    # 1.4142135623730951
```

Map

Map function is used when you need to alter all items within an iterable data collection.

It takes two arguments `map(func, *iterables)`

```
Ex5
# Map without lambda
def d1(a, b):
    return a + b

# map(func, *iterables)
x = map(d1, (1,2,3,4), (1,2,3,4))
print(x)    # <map object at 0x000000FDC50E8DC0>
print(list(x))  # [2, 4, 6, 8]
```

```
Ex6
# Map Using Lambda
lst = [1,2,3,4]

# lambda arguments : expression
m = map(lambda a: a+a, lst)
print(m)    # <map object at 0x00000006D9143B400>
print(list(m))  # [2, 4, 6, 8]
```

```
Ex7
# Find the length of the String
def d1(n):
    return len(n)

m = map(d1, ("HeMan", "XMan", "SuperMan", "BatMan"))
print('Sequence Allowed: ', list(m))  # [5, 4, 8, 6]
# print('Sequence Allowed: ', tuple(m))
# print('No Sequence: ', set(m))
```

Ex8

```
# Find the range of 10
for i in range(0, 10):
    print(i, end=' ') # 0 1 2 3 4 5 6 7 8 9
```

Ex9

```
# Find the range of 10 in descending
for i in range(10, 0, -1):
    print(i, end=' ') # 10 9 8 7 6 5 4 3 2 1
```

With map (Output based on Ex8 and Ex9)

```
def d1(x, y):
    return x*y











def d2():
    a = range(0, 10) # Asending Order
    b = range(10, 0, -1) # Desending Order

    multiples = map(d1, a, b)

    for i in multiples:
        print(i, end = " ")
```

d2()

0 9 16 21 24 25 24 21 16 9

Output	Output Example 08	Output Example 09
0 ←	0 ← 	10
9 ←	1 ← 	9
16 ←	2 ← 	8
21 ←	3 ← 	7
24 ←	4 ← 	6
25 ←	5 ← 	5
24 ←	6 ← 	4
21 ←	7 ← 	3
16 ←	8 ← 	2
9 ←	9 ← 	1