Python Functions

- Define functions
- Passing arguments to Function
- Return a value from function
- Scope of Objects
- Default arguments
- Positional and keyword arguments
- Variable length arguments

Functions

- Piece of reusable code
- Solves particular task
- Call function instead of writing code yourself

Built-in Functions

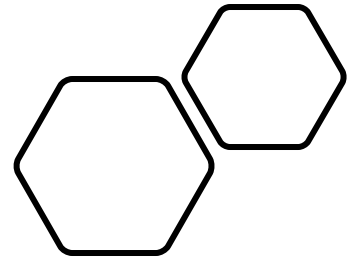
```
round(1.68, 1)
```

Syntax of Function

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

Defining a function

```
def my_function():
    print("Hello From My Function!")
```



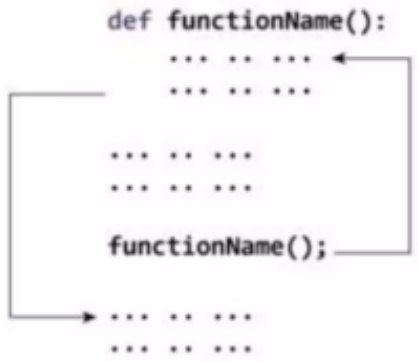
Function Call

- Once we have defined 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 parameter

Calling the Function

my_function()

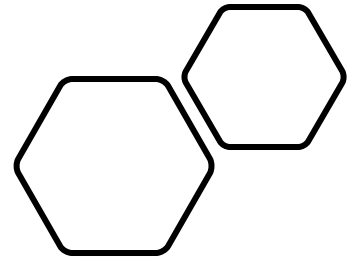
How Function works in Python?



Square function:Take one arguments and prints its square

```
def square(n):
    '''This function calculates the square of a given number'''
    res = n * n
    print("square(",n,") = ",res)


num = int(input("Enter a number " ))
square(num)
```



Square function:Take one arguments and returns its square

```
def square(n):
    res = n * n
    return res

num = int(input("Enter a number"))
res = square(num)
print("Square of",num,"=",res)
```

Function returning multiple value

```
import math

def quadEquation(a,b,c):
    x1 = (-b + math.sqrt( b**2 - 4 * a * c)) / (2 * a)
    x2 = (-b - math.sqrt( b**2 - 4 * a * c)) / (2 * a)

    return x1,x2

if __name__ == "__main__":
    print("Program to calculate the quadratic equation ")
    a = int(input("Enter the value of a : "))
    b = int(input("Enter the value of b : "))
    c = int(input("Enter the value of c : "))

    x1,x2 = quadEquation(a,b,c)
    print(x1, x2)
```

Scope and Lifetime of variables

- Scope of a variable is the portion of a program where the variable is recognized.
- Parameters and variables defined inside a function is not visible from outside. Hence, they have a local scope.
- Lifetime of a variable is the period throughout which the variable exits in the memory. The lifetime of variables inside a function is as long as the function executes.
- They are destroyed once we return from the function. Hence, a function does not remember the value of a variable from its previous calls.

```
def add(a , b):
    total = a + b
    print('Total = ', total)

add(4, 9)
```

```
def add(a , b):
    total = a + b
    print('Inside add Total = ', total)

add(4, 9)
print('Main Block Total = ', total)
```

```
total = 0
def add(a , b):
    total = a + b
    print('Inside add Total = ', total)

add(4, 9)
print('Main Block Total = ', total)
```

Default Arguments

- Function arguments can have default values in Python.
- We can provide a default value to an argument by using the assignment operator (=).

```
def calculateTotal(amount , discountPercentage = 0):
    discountAmount = discountPercentage / 100 * amount
    return amount - discountAmount

amount = 500
totalBillAmount = calculateTotal(amount , 10)
print(totalBillAmount)

amount = 500
totalBillAmount = calculateTotal(amount)
print(totalBillAmount)
```

Default Arguments

- In this function, the parameter **amount** does not have a default value and is required (mandatory) during a call.
- On the other hand, the parameter discountPercentage has a default value of 0. So, it is optional during a call.
- If a value is provided, it will overwrite the default value.
- Any number of arguments in a function can have a default value.

Default Arguments

- Once we have a default argument, all the arguments to its right must also have default values.

```
def calculateTotal(amount = 100, discountPercentage):
```

- SyntaxError: non-default argument follows default argument

Keyword Arguments

```
def fun(a,b,c):
    print("a = " , a)
    print("b = " , b)
    print("c = " , c)


fun(c = 5,b = 10, a = 15)
```

- Positional argument cannot follow keyword argument

```
def fun(a = 0 ,b = 0,c = 0):
    print("a = " , a)
    print("b = " , b)
    print("c = " , c)

fun(c = 5, 2 , 3)
```

Variable number of arguments

```
def mysum(*a):
    total = 0
    for ele in a:
        print(total,"+",ele,"=", end='')
        total += ele
        print(total)
    return total

print(mysum(5, 2, 9, 4))
```

```
def mymax(a,*b):
    large = a
    for ele in b:
        if ele > large:
            large = ele
    return large

print(mymax(4,5,7,3,1,8,2))
```

Functions as Objects

- Although functions are created differently from normal variables, **functions** are just like any other kind of value.
- They can be assigned and reassigned to variables, and later referenced by those names.

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

a = 4
b = 7
operation = multiply
print(operation(a, b))
```

```python
def add(a,b):
    print("Add")
    return a + b

def sub(a,b):
    print("Sub")
    return a - b

def mul(a,b):
    print("Multiply")
    return a * b

def div(a,b):
    print("Divide")
    return a / b
```

```python
print("Arithmetic Operations")
print("Enter two numbers ")
a = int(input())
b = int(input())
print("1.Add 2.Sub 3.Multiply 4.Divide
ch = int(input())
if ch == 1:
    calculate = add
elif ch == 2:
    calculate = sub
elif ch == 3:
    calculate = mul
else:
    calculate = div
res = calculate(a, b)
print(res)
```

Built-in Functions

A
abs()
aiter()
all()
anext()
any()
ascii()

B
bin()
bool()
breakpoint()
bytearray()
bytes()

C
callable()
chr()
classmethod()
compile()
complex()

D
delattr()
dict()
dir()
divmod()

E
enumerate()
eval()
exec()

F
filter()
float()
format()
frozenset()

G
getattr()
globals()

H
hasattr()
hash()
help()
hex()

I
id()
input()
int()
isinstance()
issubclass()
iter()

L
len()
list()
locals()

M
map()
max()
memoryview()
min()

N
next()

O
object()
oct()
open()
ord()

P
pow()
print()
property()

R
range()
repr()
reversed()
round()

S
set()
setattr()
slice()
sorted()
staticmethod()
str()
sum()
super()

T
tuple()
type()

V
vars()

Z
zip()

_
__import__()