while loop

Works till a certain condition is true

Used for ***indefinite iterations***

```
# DRY - do not repeat your self
# call - block of code ,we can name this block of code.
# pass - passing the vallue of an list or etc to block of code.
# return - block of code will give me an value in place where we call
it for.

x = 9
while x<15:
    print(x)
    x = x+1



9
10
11
12
13
14

y = 2
while y<15:
    print(y)
    y = y+15



2

# nested while
i=1
while i<4:
    lst=[1,2,3,4]
    x=1
    print(lst)
    while lst!=[]:
        lst.remove(x)
        print(lst)
        x = x+1
    i = i+1
```

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

```
# take a positive integer (3-digit)as input from the user.
# check whether the given number is an armstrong  number.
# eg.153 = 1**3 + 5**3 + 3**3
#    407 = 4**3 + 0**3 + 7**3

x = int(input("ENTER NUMBER :- "))
while x>1 :
    y = str(x)
    q=int(y[0])
    w=int(y[1])
    e=int(y[2])
    i = q**3+w**3+e**3
    a = i
    if x==a:
        print(a,"It is armstrong num")
        break
    else:
        print(a,"It is no armstrong num")
        break
```

```
ENTER NUMBER :- 407
407 It is armstrong num
```

```
x = int(input("ENTER NUMBER :- "))

while x :

    y=str(x)
    q=int(y[0])
    w=int(y[1])
    e=int(y[2])
    i = q**3+w**3+e**3
    a = i
```

```
    while a==i :
        if a==x :
            print(a,"It is armstrong num")
        else:
            print(a,"It is no armstrong num")

        a = a!=i
    x = x<1



x = int(input("ENTER NUMBER :- "))


while x > 0 :
    w = x % 10
    y += x ** w
    x //= 10

    if z == x :
        print(x,"It is armstrong num")
        print(y)
    else:
        print(x,"It is no armstrong num")




x = int(input())
r = x

while r :
    h = str(x)
    w = int(h[0: :1 ])
    c = int(h)
    if w == x :
        for w in range(h):
            h += h**1
            print(w ,"-: It is an armstrong number ")
    else:
        print(w ,"-: It is an not armstrong number ")
r = r < 1
```

```
407
```

```
---------------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
Cell In[3], line 9
```

```
      7 c = int(h)
      8 if w == x :
----> 9     for w in range(h):
     10         h += h**1
     11         print(w ,"-: It is an armstrong number ")

TypeError: 'str' object cannot be interpreted as an integer
```

```
407%10,407//7
```

```
x = "407"
q=int(x[0])
w=int(x[1])
e=int(x[2])
q**3+w**3+e**3
```

```
x = int(input())
h = str(x)
w = (h[0: 10:-1])
d = int(w)
for a in h :
    d = d**3
print(d)
```

```
123
```

```
---------------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
Cell In[1], line 4
      2 h = str(x)
      3 w = (h[0: 10:-1])
----> 4 d = int(w)
      5 for a in h :
      6     d = d**3

ValueError: invalid literal for int() with base 10: ''
```

```
H = input()
m = H[0]
print(m)
```

```
# take a positive integer as input from the user. Find the factorial
of that number.
#eg. factorial of 4 =1*2*3*4


x = int(input("enter num :- "))
```

```
while x>0:
    y = 1
    z = 1
    a = x
    while y==1:
        if x < 0 :
            print(" no ")
        elif a==0:
            print("the factorial num of ",a," is :- ", z)
        else :
             for a in range(1,x+1):
                    y = y * a
        print("the factorial num of ",x,"is", y)

        y!=x
    x = x<0
```

```
enter num :- 4
the factorial num of  4 is 24
```

```
x = int(input("Enter num:-  "))
y = 1
z = 1
if x<0:
    print( "no")
elif x==0:
    print("factorial  of 0 is ",z)

else:
    for a in range(1,x+1):
        y = y * a
    print("the factorial num of ",x ,"is :- ", y)
```

```
Enter num:-  4
the factorial num of  4 is :-  24
```

Function are a block of code defined with a name

DRY--DO not repeat your self

Function provides reseability of code.

Type of functions

- Built-in Function or predefined Funvction.

- User-defined function.

```
# def is for defined
def first_function():
    print(" |data science is amazing|")

first_function()    # function is calling
```

```
 |data science is amazing|
```

Creating a function with parameters

```
def first_function(x):                    # x is a parameter
(variable)
    print("data science is amazing")
    print(x)
first_function(12)                        # 12 is an arguments
(value)
first_function(13)

def second_function (x,y,z,c):
    print(x)
    print(y)
    print(z)
    print(c)
print(type(second_function("str",12,True,{2+3})))
```

Returning value from a Function.

```
def calculator(x,y):
    a = x+y
    return a
b = calculator(23,45)
print(b)
```

```
68
```

```
def calculator(x,y):
    a = x+y
    return a
calculator(23,35) +2
```

```
60
```

```
def second_function (x,y,z,c):
    print(x)
    print(y)
    print(z)
    print(c)
    return x,y,z,c
print(type(second_function("str",12,True,{2+3})))


def my_func(x,y):
    print("we are adding numbers")
    a = x+y
    return a
my_func(12,45)

def func():
    print("xyz")

x = func()   # x have no value(argument) by defolt will printing None.

print(x)

xyz
None

def func(a):
    print("xyz")
    print(a)
    return a
x = func(12)
print("returning ",x)

xyz
12
returning  12

func(12 - 2)
print(x)

xyz
10
12

def func(a):
    print("xyz")
    print(a)
    return None,1,2,3,4  # what ever you right with return it will
return the same
x = func(12)
print("returning ",x)
type(x)
```

```
xyz
12
returning  (None, 1, 2, 3, 4)

tuple
```

```
def func(a):
    print("xyz")
    print(a)
    return a
x = func(12)
print("returning :- ",x)

print("hello")

None

print(None)

print(print(print("rishi")))

print("h")
type(print(print("h")))

def add(a,b):
    sum = a+b
    print(sum)


print(add(45,79)) # this is giving none because value has not pass to
your
                  #  call func And print fun give allwyas none
```

Returning mulitple value from a function

```
a,b = ("hello", 67)
y = "true"
x = 12
print(a,b)
print(y)
print(x)

def calculator(a,b):
    sum = a+b
    diff = a-b
    product = a*b
    div = a/b

    return sum,diff,product,div       # return the value in the form
of a tuple
```

```
x = calculator(10,2)
print(x)

def calculator(a,b):
    sum = a+b
    diff = a-b


    return sum,diff
x = calculator(10,2)
print(x)

def calculator(a,b):
    sum = a+b
    diff = a-b
    product = a*b
    div = a/b

    return sum,diff,product,div
a,b,c,d = calculator(10,2)
print(a,b,c,d)
print(a)
print(b)
print(c)
print(d)
print(calculator(a,b))
```

```
12 8 20 5.0
12
8
20
5.0
(20, 4, 96, 1.5)
```

```
def calculator(a,b):
    sum = a+b
    diff = a-b
    product = a*b
    div = a/b

    return sum,diff,product,div    # return the value in the form of a
tuple
p,q,r,s = calculator(10,2)
print(p,q,r,s)

y = calculator(a,b)        # indexing need 96 only in the output
print(calculator(a,b))
print(y)
```

Using Docstring in Functions

Single-line docstring

```
def show_result():
    """let us understand docstring"""
    print("this is a test program")
show_result()
```

```
this is a test program
```

```
show_result.__doc__
```

```
'let us understand docstring'
```

```
help(show_result)
```

```
Help on function show_result in module __main__:

show_result()
    let us understand docstring
```

Multi- line docstring

```
def show_result():
    """let us understand docstring

    i am using this function to understand about docstrings
    Docstring are used to document functions"""
    print("this is a test program")

show_result.__doc__

help(show_result)
```

Scop of Variables

- ***global variable*** - are created outside the function. we can accessed global variable everywere program, inside the function and outside the function.
- ***local variable*** - are created only in inside the function, it is only accessible inside the function.

```
t = " python"       # global variable.

def scope_var():
```

```
    u = "analytics"     # local variable.
    print('T is :- ',t)
scope_var()
print("the value of t is",t)
print("the value of u is",u)

t = " python"       # global variable t
u = "analytics"      # global variable t


def scope_var():
    u = "analytics"    #local variable u
    print(u)
    print('T is ',t)
scope_var()

print("the value of t is ",t)

print("the value of u is ",u)     #global variable u

t = " python"       # global variable t
u = "analytics"      # global variable t

def scope_var():
    print("value of u inside function before declaring it as local
",u)
    u = "data science"    #local variable u
    print(u)
    print('T is ',t)

scope_var()

print("the value of t is ",t)

print("the value of u is ",u)     #global variable u
```

Make globle variable

```
# Creating a global variable inside a function

def test_global():
    global x
    x = 78
    print("x inside function is ", x)

test_global()

print("x outside function is",x)
```

```
def test_global():
    global x = 78
    print("x inside function is ", x)

test_global()

print("x outside function is",x)

# Make a function inside a function and call both.

def first_function(x,y):

    """ I this program to make function inside function

    first function i will do sum of two num
    in second we did the division"""
    w = x+y


    def second_function(x,y):
        w = x*y
        print(w)
    second_function(5
                    ,2)
    print(w)
first_function(1,2)




print(first_function.__doc__)

print(second_function.__doc__)
```

non local variable

```
def outer():
    w = 89
    def inner():
        w = 67
        print(w)
    inner()
    print(w)
outer()
```

```
def outer():
    w = 89
    def inner():
        nonlocal w      # non-local will change the value of outer
same variable
        w = 67
        print(w)
    inner()
    print(w)
outer()
```

```
def outer():
    r = 89
    def inner():
        nonlocal w
        w = 67
        print(w)
    inner()
    print(w)
outer()
```

```
def enter_name():
    x = "rohit"
    def add_age():
        nonlocal x
        x = "22"
        print(x)   # add_name fun x
    add_age()
    print(x)   #enter_name fun x
    print(x)
enter_name()
```

```
def first_fun():
    x = 2
    y = 4
    if x > y:
        print("It is Grater then  y :-" ,x)
    else:
        print("It is not Grater then x :-", y)
        def second_fun():
            for x in range(1,10+1):
                print(2 ,"X", x, "=" ,x*2)
    second_fun()

first_fun()
```

Pyhthon Function Arguments

- a. Positinonal Arguements
- a. keyword Arguements
- a. Default Arguements
- a. Variable-Length Arguements

Positinonal Arguements

are arguements that are passed to a function in proper positional function

```
def calc(p,q):
    diff = p-q
    print("p arguement is :-", p)
    print("q arguement is :-",q)
    return diff
calc(89,45) #p 89 ,q 45

# Positinonal Arguements
def my_func():
    pass

my_func(2)
```

Keyword arguement-

- Values get assingned to the parameter by their name(keyword)
- Here the order of the arguemnents does not matter.

```
def calc(p,q):
    diff = p-q
    return diff
calc(p=10,q=2) #p 89 ,q 45

calc(q=2,p=10)

calc(a=89,q=45)  # the parameter should be the same vo put in the fun
```

Default Arguements-

Take the default value during the function call if we do not pass them

```
def get_message(msg):
    print(msg)

get_message("hi")
```

```
def get_message(msg = "hello world"):
    print(msg)
    return msg
get_message()
```

```
def get_message(msg = "hello world"):
    print(msg)

get_message("hi","hello")
```

```
def get_message(msg = "hello world",q = 56):
    print(q)
    return msg
get_message()
```

```
def get_message(msg = "hello world",q = 56):
    return msg,q
get_message()
```

```
def get_message(msg):
    print(msg)

get_message()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[6], line 4
      1 def get_message(msg):
      2     print(msg)
----> 4 get_message()

TypeError: get_message() missing 1 required positional argument: 'msg'
```

```
def get_message(msg,q=56):
    print(msg,q)
    return q

get_message("hi",q = 45)
```

```
hi 45
```

```
45
```

Variable- length Arguments or Arbitary Arguments-

When we need to pass mulitple arguments to the function,we can use variable- length arguments

```
# Arbitary postional arguments
def test(*numbers):  # numnbers will come in tuple
    print(numbers)
    diff = 20
    for x in numbers:
        diff = diff - x
    print(diff)
test(9,4,12)
```

```
(9, 4, 12)
-5
```

```
def test(*numbers):  # numnbers will come in tuple
    print(numbers)
    diff = 5
    for x in numbers:
        diff = diff - x
    print(diff)
test(9,4,12)
```

```
(9, 4, 12)
-20
```

```
# Arbitary keywords arguments

def my_func(**key_word):  # keywords arguments gives output in a
dictionary.
    print(key_word)
    for sub in key_word:
        print("key of dic   :-",sub)   # get arguments key
        sub_marks = key_word[sub]
        print("vale of a key   :-",sub_marks)     # get arguments
value
        print(sub, "=" ,sub_marks)

# pass multiple values
my_func( maths=56 , english=61 , scince=73 )
```

```
{'maths': 56, 'english': 61, 'scince': 73}
key of dic   :- maths
vale of a key   :- 56
maths = 56
key of dic   :- english
vale of a key   :- 61
english = 61
key of dic   :- scince
vale of a key   :- 73
scince = 73
```

```
def first_func(x):
    if x>10:

        return "hello world"
    else:
        return first_func(x+1)

first_func(15)
```

```
'hello world'
```

```
 3571220714
```