EXPERIMENT NO:4

AIM OF THE EXPERIMENT : PROGRAMME ON FUNCTIONS

Definition: function is a reusable block of code that performs a specific task, taking input (arguments), processing it, and optionally returning an output.

* Functions are designed to make code more organized, readable, and reusable by encapsulating a set of instructions that can be called and executed multiple times.

Declaration:

In Python, a function declaration, or definition, uses the def keyword followed by the function name, parentheses for optional parameters, a colon, and an indented code block containing the function's logic.

Utility:

It is a small, reusable piece of code designed to perform a specific, common task, improving code readability, reducing redundancy, and promoting modularity.

Utility functions are created to encapsulate reusable logic, making code cleaner and easier to maintain.

#PROGRAM-1

def check\_relation(a,b):

if(a==b):

return 0

if(a>b):

return 1

if(a<b):

return -1

a=int(input("enter 1st number:"))

b=int(input("enter 2nd number:"))

res= check\_relation(a,b)

if(res==0):

print("a==b")

if(res==-1):

print("a<b")

if(res==1):

print("a>b")

#OUTPUT

enter 1st number:65

enter 2nd number:56

a>b

#PROGRAM-2

\*\*Write a function named right\_justify that takes a string named s as a parameter and prints the string with enough leading spaces so that the last letter of the string is in column 70 of the display.

def right\_justify(s):

print(s.rjust(70))

right\_justify("hello")

#OUTPUT

hello

#PROGRAM-3

def do\_twice(func, arg):

func(arg)

func(arg)

def print\_twice(arg):

print(arg)

print(arg)

def do\_four(func, arg):

do\_twice(func, arg)

do\_twice(func, arg)

do\_twice(print, 'spam')

print('')

do\_four(print, 'spam')

#OUTPUT

spam

spam

spam

spam

spam

spam

#PROGRAM-4

def do\_twice(f):

f()

f()

def do\_four(f):

do\_twice(f)

do\_twice(f)

def print\_beam():

print('+ - - - -', end=' ')

def print\_post():

print('| ', end=' ')

def print\_beams():

do\_twice(print\_beam)

print('+')

def print\_posts():

do\_twice(print\_post)

print('|')

def print\_row():

print\_beams()

do\_four(print\_posts)

def print\_grid():

do\_twice(print\_row)

print\_beams()

print\_grid()

#four by four grid

def one\_four\_one(f, g, h):

f()

do\_four(g)

h()

def print\_plus():

print('+', end=' ')

def print\_dash():

print('-', end=' ')

def print\_bar():

print('|', end=' ')

def print\_space():

print(' ', end=' ')

def print\_end():

print()

def nothing():

"do nothing"

def print1beam():

one\_four\_one(nothing, print\_dash, print\_plus)

def print1post():

one\_four\_one(nothing, print\_space, print\_bar)

def print4beams():

one\_four\_one(print\_plus, print1beam, print\_end)

def print4posts():

one\_four\_one(print\_bar, print1post, print\_end)

def print\_row():

one\_four\_one(nothing, print4posts, print4beams)

def print\_grid():

one\_four\_one(print4beams, print\_row, nothing)

print\_grid()

#PROGRAM 5

def b(z):

prod = a(z, z)

print(z, prod)

return prod

def a(x, y):

x = x + 1

return x \* y

def c(x, y, z):

total = x + y + z

square = b(total)\*\*2

return square

x = 1

y = x + 1

print(c(x, y+3, x+y))

#OUTPUT

9 90

8100

#PROGRAM-6

\*\*Write a function named ack that evaluates the Ackermann function. Use your function to evaluate ack(3, 4), which should be 125. What happens for larger values of m and n?

def ackermann(m, n):

if m == 0:

return n+1

if n == 0:

return ackermann(m-1, 1)

return ackermann(m-1, ackermann(m, n-1))

print(ackermann(3, 4))

#OUTPUT

125#PROGRAM 7

def first(word):

return word[0]

def last(word):

return word[-1]

def middle(word):

return word[1:-1]

def is\_palindrome(word):

if len(word) <= 1:

return True

if first(word) != last(word):

return False

return is\_palindrome(middle(word))

print(is\_palindrome('alice'))

print(is\_palindrome('donald'))

print(is\_palindrome('anne'))

print(is\_palindrome('redivider'))

#OUTPUT

False

False

False

True

#PROGRAM 8

\*\*A number, a, is a power of b if it is divisible by b and a/b is a power of b. Write a function called is\_power that takes parameters a and b and returns True if a is a power of b.

def is\_power(a, b):

if a == 1:

return True

if a % b != 0 or b == 0:

return False

return is\_power(a // b, b)

print(is\_power(8, 2))

print(is\_power(27, 3))

print(is\_power(10, 2))

#OUTPUT

True

True

False

#PROGRAM 9

Write a function called gcd that takes parameters a and b and returns their greatest common divisor.

def gcd(a, b):

while b:

a, b = b, a % b

return abs(a)

print(gcd(48, 18))

#OUTPUT

6