Functional Programming

What is function?

A function is a structured code template used to perform some specific tasks when it gets called.
Function is also used to take input values at run time and process it with the function body by perform specific tasks and return the processed output values.

We can define a functions using **'def'** keyword along with meaningful function name.

Example of Function

```
# Functions
# how to definie functions
# using 'def' keyword we can define functions with proper name

# Examples of functions

def addition(a, b):
    return a + b

# now the above additions is a function which getting two arguments and returning the result

# how to call a function

result = addition(2, 6)
print(result)
8
# 2 and 6 will be added inside the addtion func and returned to result variable
```

Use of Functions

- Code reusability
- Reduce complexity
- Optimize memory

Types of functions

- User defined functions
- Built-in functions
- Anonymous function(Lambda)
- Recursive functions

User Defined Functions

These are functions created by users to perform specific tasks. They are defined using the def keyword and can be reused throughout the code.

example:

```
>> # User Defined Functions
>>
>> def sampleFunc():
..     return "this is the sample function return a string"
..
>>
>> # calling the function
>>
>> print(sampleFunc())
this is the sample function return a string
>>
>> # or we can save the return data from that function into a variable
>>
>> funcData = sampleFunc()
>> print(funcData)
this is the sample function return a string
>>
```

Built-in Functions

Built-in functions in Python are pre-defined functions that are provided as part of the Python programming language.

These functions are readily available for use without the need for importing any external modules.

They cover a wide range of functionalities and are designed to perform common tasks efficiently.

Some of the Built-in functions

```
# some built-in functions examples

abs(-13)
13
abs(-53.21)
53.21
bin(1)
'0b1'
bin(20)
'0b10100'
bool(0)
False
bool(1)
True
eval("23+25")
48
```

```
eval("10<5")
False
hex(12)
'0xc'
id(50)
140721650329928
input("Enter something : ")
Enter something : 56
'56'
len("softlogic")
9
max(152, 689, 12)
689
min(152, 689, 12)
12
```

```
>> import builtins
>> dir(builtins)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BaseExceptionGroup'
, 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'ChildProcessError', '
ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetError',
'DeprecationWarning', 'EOFError', 'Ellipsis', 'EncodingWarning', 'EnvironmentError', 'Excepti
on', 'ExceptionGroup', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError',
'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationErro
r', 'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', '
LookupError', 'MemoryError', 'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError'
, 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWar
ning', 'PermissionError', 'ProcessLookupError', 'RecursionError', 'ReferenceError', 'Resource
Warning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'SyntaxErr
or', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeE
rror', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'Unic
odeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError',
'ZeroDivisionError', '__build_class__', '__debug__', '__doc__', '__import__', '__loader__', '
__name__', '__package__', '__spec__', 'abs', 'aiter', 'all', 'anext', 'any', 'ascii', 'bin',
'bool', 'breakpoint', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'com
plex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec
', 'exit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr', 'hash',
'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'l
ist', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'p
ow', 'print', 'property', 'quit', 'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'sl
ice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
>>
```

Math functions

Math functions in Python are a collection of pre-defined functions in the math module that enable complex mathematical calculations and transformations on numeric data. They cover various areas of mathematics, including trigonometry, logarithms, exponentiation, rounding, and factorials.

```
# math functions

import math
dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh
', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'cbrt', 'ceil', 'comb', 'copysign
', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'exp2', 'expm1'
, 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot
', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt', 'lcm', 'ldexp', 'lg
amma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'nextafter', 'perm', 'pi
', 'pow', 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh',
'tau', 'trunc', 'ulp']
```

```
import math

math.ceil(45.2)
46
math.floor(45.2)
45
math.exp(4)
54.598150033144236
math.factorial(8)
40320
math.pi
3.141592653589793
math.pow(2,2)
4.0
math.pow(2,6)
64.0
math.degrees(120)
6875.493541569878
math.radians(6875.5)
120.00011272087013
math.sin(120)
0.5806111842123143
math.cos(120)
0.8141809705265618
```

User defined functions

A function created by the user to perform a specific task, allowing code reusability and modularity.

- you can define your own functions using the **def** keyword, followed by a function name, parentheses for optional parameters, and a colon.

- The function body is indented and contains the code to be executed when the function is called.
- User-defined functions can be called and reused throughout a program to improve code organization and maintainability.

```
# sample basic functions

def greet():
    print("Hello Softlogic!")


greet()
Hello Softlogic!

# addition function with return keyword

def addition():
    return 2 + 3

print(addition())
5
```

Types of arguments used in user defined functions

- Default arguments
- Keyword arguments
- Positional arguments
- Arbitrary arguments

Default arguments

```
# default argument example in function

def profile(name="suresh", age=54, city="chennai"):
    print("my name is ", name)
    print("age is ", age)
    print("my native is ", city)


profile()
my name is  suresh
age is  54
```

```
my native is  chennai
```

Keyword arguments

```
# keyword argument example

def profile(name="suresh", age=26, city="chennai"):
    print("my name is ", name)
    print("age is ", age)
    print("native is ", city)


profile(age=54, city="chengalpattu", name="jagadish")
my name is  jagadish
age is  54
native is  chengalpattu
```

Positional arguments

```
# example of positional argument

profile("sundar", 35, "nagapattinam")
my name is  sundar
age is  35
native is  nagapattinam

profile("trichy", "ramesh", 30)
my name is  trichy
age is  ramesh
native is  30
```

Arbitrary arguments

```
# example of arbitrary arguments

def profile(*names):
    print(names)


profile("suresh", "ramesh", "michel", "senthil")
('suresh', 'ramesh', 'michel', 'senthil')

# if you want to print one by one, you could use looping

def profile(*names):
    for i in names:
        print("hello Mr.", i "!, Good morning")

SyntaxError: invalid syntax
def profile(*names):
    for i in names:
        print("hello Mr.", i, "!, Good morning")


profile("suresh", "ramesh", "michel", "senthil")
hello Mr. suresh !, Good morning
hello Mr. ramesh !, Good morning
hello Mr. michel !, Good morning
hello Mr. senthil !, Good morning
```

Recursive functions

A recursive function in Python is a function that calls itself during its execution.

It solves complex problems by breaking them down into smaller, more manageable subproblems.

```
# example of recursive functions
# find factorial of given number

def factorial(number):
    if number==0:
        return 1
    else:
        return number * factorial(number - 1)


factorial(3)
6
factorial(4)
24
```

```
# find fibonacci

def fibonacci(num):
    if num <= 1 :
        return num
    else:
        return fibonacci(num-1) + fibonacci(num-2)


fibonacci(5)
5
fibonacci(6)
8
```

Lambda Functions

- Lambda function is a nameless function which can be used for a short period of time.
- Lambda function uses lambda keyword instead of def keyword.
- Lambda function is also known as anonymous function.
- Lambda functions are used along with higher order functions.
- Lambda function never uses return keyword.
- Lambda functions frequently used with built in functions like map (), filter () and reduce().

```
# normal function

def add(a, b):
    return a+b

add(4, 6)
10

# lambda function

add = lambda a, b: a+b
add(4, 6)
10

# example 2

def findGreater(a, b, c):
    if a>b and a>c:
        return "a is greater than b and c"
    elif b>c and b > a:
        return "b is greater than a and c"
    else:
        return "c is greater than a and b"
```

```
findGreater(15, 10, 25)
'c is greater than a and b'
 # now lambda function


findGreaterNum = lambda a,b,c: "a is greater than b and c" if a>b and a>c else "
b is greater than a and c" if b>a and b>c else "c is greater than a and b"
findGreaterNum(15, 10, 25)
'c is greater than a and b'
findGreaterNum(25, 15, 10)
'a is greater than b and c'
findGreaterNum(25, 35, 10)
'b is greater than a and c'
```

```
# lambda function with map, filter, reduce

lis = [2,3,4,5,6,7,8,9]

res = list(map(lambda a: a * 2, lis))
print(res)
[4, 6, 8, 10, 12, 14, 16, 18]

sett = {2,6,8,12,10,4,3}

result = set(filter(lambda a: a % 2 == 0, sett))
print(result)
{2, 4, 6, 8, 10, 12}

# reduce function
# for reduce func you need to import reduce function from funcTools

from functools import reduce
lis = [2,3,4,5,6,7,8,9]
result = reduce(lambda a,b: a+b, lis)
print(result)
44
```