Let's create a function(with docstring)

```
def is_even(num):
  """
  This function returns if a given number is odd or even
  input - any valid integer
  output - odd/even
  created on - 16th Nov 2022
  """
  if type(num) == int:
    if num % 2 == 0:
      return 'even'
    else:
      return 'odd'
  else:
    return 'pagal hai kya?'
```

```
# function
# function_name(input)
for i in range(1,11):
  x = is_even(i)
  print(x)
```

```
odd
even
odd
even
odd
even
odd
even
odd
even
```

```
print(type.__doc__)
```

```
type(object_or_name, bases, dict)
type(object) -> the object's type
type(name, bases, dict) -> a new type
```

2 Point of views

```
is_even('hello')
```

```
'pagal hai kya?'
```

Parameters Vs Arguments

Types of Arguments

- Default Argument
- Positional Argument
- Keyword Argument

```
def power(a=1,b=1):
  return a**b
```

```
power()
```

```
1
```

```
# positional argument
power(2,3)
```

```
8
```

```
# keyword argument
power(b=3,a=2)
```

```
8
```

`*args and **kwargs`

`*args and **kwargs` are special Python keywords that are used to pass the variable length of arguments to a function

```
# *args
# allows us to pass a variable number of non-keyword arguments to a function.

def multiply(*kwargs):
  product = 1

  for i in kwargs:
    product = product * i

  print(kwargs)
  return product
```

```
multiply(1,2,3,4,5,6,7,8,9,10,12)
```

```
    (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12)
    43545600
```

```
# **kwargs
# **kwargs allows us to pass any number of keyword arguments.
# Keyword arguments mean that they contain a key-value pair, like a Python dictionary.

def display(**salman):

  for (key,value) in salman.items():
    print(key,'->',value)
```

```
display(india='delhi',srilanka='colombo',nepal='kathmandu',pakistan='islamabad')
```

```
    india -> delhi
    srilanka -> colombo
    nepal -> kathmandu
    pakistan -> islamabad
```

Points to remember while using `*args and **kwargs`

- order of the arguments matter(normal -> `*args` -> `**kwargs`)
- The words “args” and “kwargs” are only a convention, you can use any name of your choice

How Functions are executed in memory?

Without return statement

```
L = [1,2,3]
print(L.append(4))
print(L)
```

```
    None
    [1, 2, 3, 4]
```

Variable Scope

```
def g(y):
    print(x)
    print(x+1)
x = 5
g(x)
print(x)
```

```
def f(y):
    x = 1
    x += 1
    print(x)
x = 5
f(x)
print(x)
```

```
def h(y):
    x += 1
x = 5
h(x)
print(x)
```

```
def f(x):
   x = x + 1
   print('in f(x): x =', x)
   return x

x = 3
z = f(x)
print('in main program scope: z =', z)
print('in main program scope: x =', x)
```

Nested Functions

```
def f():
  def g():
    print('inside function g')
    f()
  g()
  print('inside function f')
```

```
f()
```

```
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
 inside function g
```

```python
def g(x):
    def h():
        x = 'abc'
    x = x + 1
    print('in g(x): x =', x)
    h()
    return x

x = 3
z = g(x)
```

```
 inside function g
```

```python
def g(x):
    def h(x):
        x = x+1
        print("in h(x): x = ", x)
    x = x + 1
    print('in g(x): x = ', x)
    h(x)
    return x

x = 3
z = g(x)
print('in main program scope: x = ', x)
print('in main program scope: z = ', z)
```

```
 inside function g


 inside function g
```

Functions are 1st class citizens

```
    inside function g
```

```
# type and id
def square(num):
  return num**2

type(square)

id(square)
```

```
    140471717004784
```

```
# reassign
x = square
id(x)
x(3)
```

```
    9

    inside function g
```

```
a = 2
b = a
b
```

```
    2
    inside function g
```

```
# deleting a function
del square
```

```
    inside function g
```

```
square(3)
```

```
    ---------------------------------------------------------------------------
    NameError                                 Traceback (most recent call last)
    <ipython-input-104-2cfd8bba3a88> in <module>
    ----> 1 square(3)

    NameError: name 'square' is not defined
```

SEARCH STACK OVERFLOW

```
    inside function g
```

```
# storing
L = [1,2,3,4,square]
L[-1](3)
```

```
    9

    inside function g
```

```
s = {square}
s
```

```
    {<function __main__.square(num)>}

    inside function g
```

```
# returning a function
```

```
    inside function g


    inside function g
```

```
def f():
    def x(a, b):
        return a+b
    return x

val = f()(3,4)
print(val)
```

```
# function as argument
```

```
def func_a():
    print('inside func_a')

def func_b(z):
    print('inside func_c')
    return z()

print(func_b(func_a))
```

```
    inside function g
```

```
inside function g
```

Benefits of using a Function

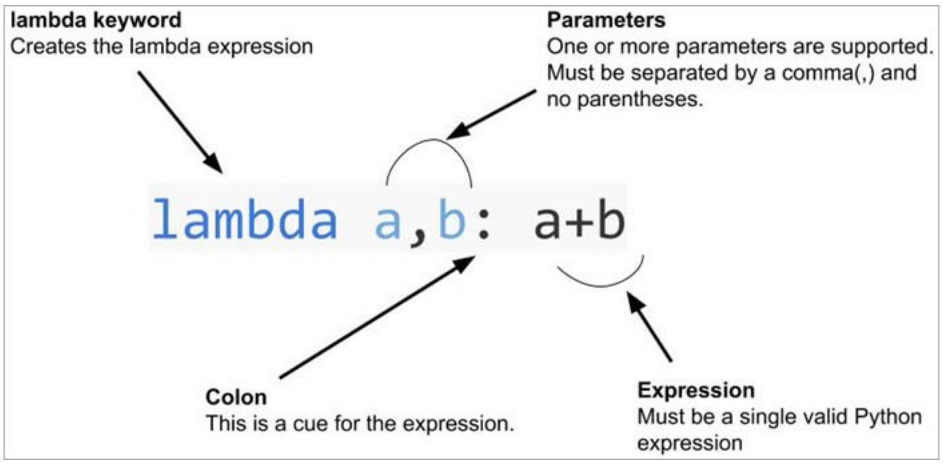
- Code Modularity
- Code Readibility
- Code Reusability

```
inside function g
```

Lambda Function

A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.


lambda keyword
Creates the lambda expression
Parameters
One or more parameters are supported.
Must be separated by a comma(,) and no parentheses.
lambda a,b: a+b
Colon
This is a cue for the expression.
Expression
Must be a single valid Python expression

```
inside function g
```

```
# x -> x^2
lambda x:x**2
```

```
<function __main__.<lambda>(x)>
```

```
inside function g
```

```
# x,y -> x+y
a = lambda x,y:x+y
a(5,2)
```

```
7
inside function g
```

Diff between lambda vs Normal Function

- No name
- lambda has no return value(infact,returns a function)
- lambda is written in 1 line
- not reusable

Then why use lambda functions?

They are used with HOF

```
inside function g
```

```
# check if a string has 'a'
a = lambda s:'a' in s
a('hello')
```

```
False
inside function g
```

```
# odd or even
a = lambda x:'even' if x%2 == 0 else 'odd'
a(6)
```

```
'even'
inside function g
```

Higher Order Functions

```
inside function g
```

```
# Example

def square(x):
  return x**2

def cube(x):
  return x**3

# HOF
def transform(f,L):
  output = []
  for i in L:
    output.append(f(i))

  print(output)
```

Map

```
# square the items of a list
list(map(lambda x:x**2,[1,2,3,4,5]))
```

```
[1, 4, 9, 16, 25]

inside function g
```

```
# odd/even labelling of list items
L = [1,2,3,4,5]
list(map(lambda x:'even' if x%2 == 0 else 'odd',L))
```

```
['odd', 'even', 'odd', 'even', 'odd']

inside function g
```

```
# fetch names from a list of dict

users = [
    {
        'name':'Rahul',
        'age':45,
        'gender':'male'
    },
    {
        'name':'Nitish',
        'age':33,
        'gender':'male'
    },
    {
        'name':'Ankita',
        'age':50,
        'gender':'female'
    }
]

list(map(lambda users:users['gender'],users))
```

```
['male', 'male', 'female']

inside function g
```

Filter

```
inside function g
```

```
# numbers greater than 5
L = [3,4,5,6,7]

list(filter(lambda x:x>5,L))
```

```
[6, 7]

inside function g
```

```
# fetch fruits starting with 'a'
fruits = ['apple','guava','cherry']
```