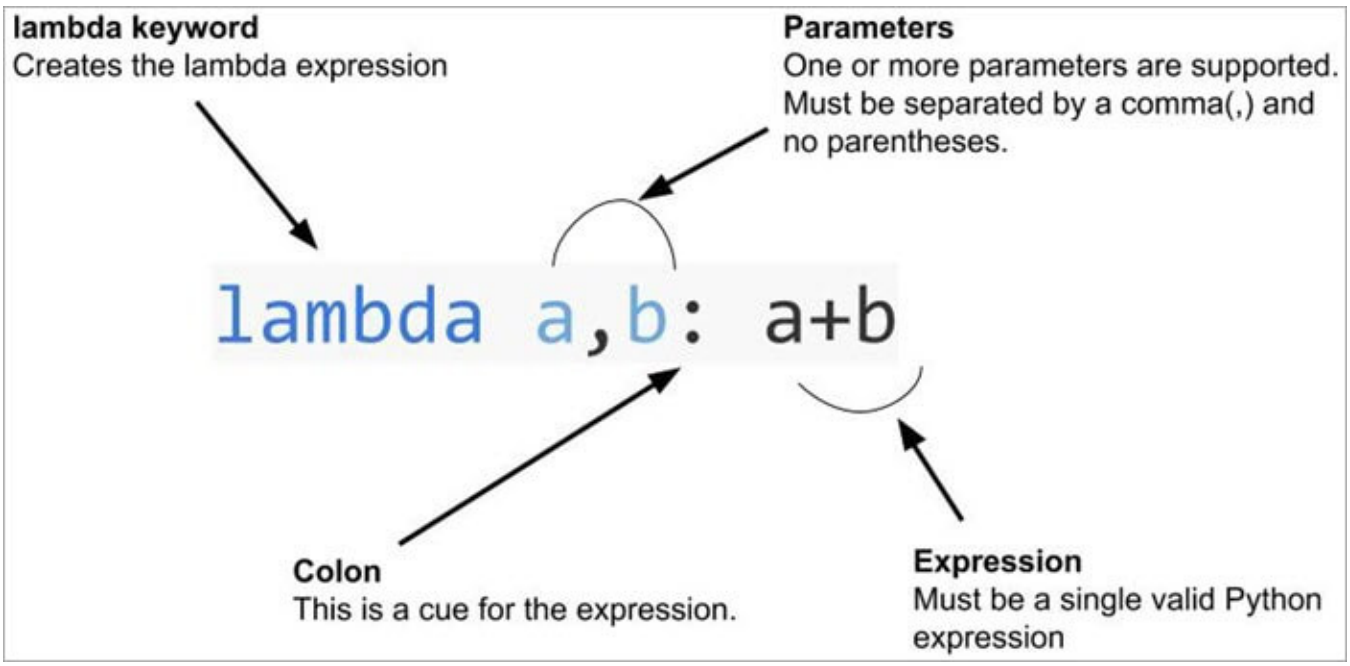
Lambda function

- A lambda function is a small/one line function.
- It is also called anonymous function
- use any where, you have a function definition that takes one argument,
- Use lambda functions when an anonymous function is required for a short period of time.
- A lambda function can take any number of arguments, but can only have one expression.



In [1]:

```
# lambda input: "expression"
# this is the form of writing the lambda function
```

In [2]:

```
x = lambda x : x**2
x(12)
```

Out[2]:

```
144
```

In [3]:

```
type(x)
```

Out[3]:

```
function
```

In [4]:

```
a=lambda x,y : x+y
a(5,9)
```

Out[4]:

```
14
```

In [5]:

```
type(a)
```

Out[5]:

```
function
```

Difference between the normal function and lambda function

1. lambda has no return values like in normal function
2. lambda will return the whole function instead of giving you a value
3. lambda will be written in only one line
4. normal functions are made for code reuseability where as lambda is nor for code reuseability
5. lambda function will not have any type of name

Why Use Lambda Functions?

- we use lambda function with higher order functions
- The power of lambda is better shown when you use them as an anonymous function inside another function.
- Use lambda functions when an anonymous function is required for a short period of time.

In [6]:

```
# creating a lambda function if the first alphabet of the word is a or not
b=lambda x: x[0]=='a'
# we will get the results in the boolean form
```

In [7]:

```
b('Apple')
```

Out[7]:

```
False
```

In [8]:

```
b('apple')
```

Out[8]:

```
True
```

In [9]:

```
b('banana')
```

Out[9]:

```
False
```

In [10]:

```
# we want to know if the given number is odd or even
a=lambda x: "Even" if x%2==0 else"Odd"
# we are expressing our whole logic in the single line
```

In [11]:

```
a(9)
```

Out[11]:

```
'Odd'
```

In [12]:

```
a(8)
```

Out[12]:

```
'Even'
```

In [13]:

```
n,p=2,4
def test():
    return n**p
test()
```

Out[13]:

```
16
```

In [14]:

```
# lambda function
lambda n,p :n**p
```

Out[14]:

```
<function __main__.<lambda>(n, p)>
```

In [15]:

```
a = lambda n,p :n**p
a(n,p)
```

Out[15]:

```
16
```

In [16]:

```
#lambda can take any number of argument
la = lambda a,b,c : a+b+c
la(2,3,8)
```

Out[16]:

```
13
```

In [17]:

```
# add tow numbers by using lambda function
add = lambda x,y : x+y
add(3,6)
```

Out[17]:

```
9
```

In [18]:

```
# find celsius from fahrenheit
c_to_f = lambda c : (9/5)*c+32
c_to_f(45)
```

Out[18]:

```
113.0
```

In [19]:

```
# find max form two numbers
find_max = lambda x,y : x if x>y else y
find_max(4,13)
```

Out[19]:

```
13
```

In [20]:

```
# find string length by using lambda function
Str="this is string"
str_len = lambda Str : len(Str)
str_len(Str)
```

Out[20]:

```
14
```

Higher order functions

- A function is called Higher Order Function if it contains other functions as a parameter or returns a function as an output i.e, the functions that operate with another function are known as Higher order Functions.

Properties of higher-order functions:

- A function is an instance of the Object type.
- You can store the function in a variable.
- You can pass the function as a parameter to another function.
- You can return the function from a function.

In [21]:

```
# creating a list
L=[11,14,21,23,56,78,45,29,28]
L
```

Out[21]:

```
[11, 14, 21, 23, 56, 78, 45, 29, 28]
```

In [22]:

```
'''we want to make one function such that when we pass
the above list it will return the three output'''

# first output = we want sum of all the even numbers inside the list L
# second output = we want output of all odd number
# third output = we want sum of numbers which are devisible by 3
```

Out[22]:

```
'we want to make one function such that when we pass \nthe above list it w
ill return the three output'
```

In [23]:

```
# our normal code would be like this
def return_sum(L):
    even_sum=0 # initial sum of even numbers
    odd_sum=0
    div3_sum=0

    # now we will run a loop on the list L
    for i in L:
        if i%2==0:
            even_sum=even_sum+i
        if i%2!=0:
            odd_sum=odd_sum+i
        if i%3==0:
            div3_sum=div3_sum+i
    return(even_sum,odd_sum,div3_sum)
```

In [24]:

```
return_sum(L)
# we are getting the answer but we can still improve the logic
# so we will improve this normal function to higher order function
```

Out[24]:

```
(176, 129, 144)
```

In [25]:

```
def return_sum(func,L):
    result=0

    for i in L:
        if func(i):
            result=result+i
    return result
```

In [26]:

```
# here x,y and z are values one by one inside the list L

x=lambda x:x%2==0
y=lambda y:y%2!=0
z=lambda z:z%3==0
print(return_sum(x,L))
print(return_sum(y,L))
print(return_sum(z,L))
```

```
176
129
144
```

In [27]:

```
# Python program to illustrate functions
# can be treated as objects
def shout(text):
    return text.lower()

print(shout('HELLO'))

# Assigning function to a variable
yell = shout

print(yell('HELLO'))
'''In the above example, a function object referenced
by shout and creates a second name pointing to it, yell.'''
```

```
hello
hello
```

Out[27]:

```
'In the above example, a function object referenced \nby shout and creates
a second name pointing to it, yell.'
```

In [28]:

```
'''Say you have a function definition that takes one argument,
and that argument will be multiplied with an unknown number:'''

def myfunc(n):
    return lambda a : a * n
```

In [29]:

```
'''Use that function definition to make a function
that always doubles the number you send in:'''

def myfunc(n):
    return lambda a : a * n

mydoubler = myfunc(2)

print(mydoubler(11))
```

```
22
```

In [30]:

```
'''Or, use the same function definition to make a
function that always triples the number you send in:'''

def myfunc(n):
    return lambda a : a * n

mytripler = myfunc(3)

print(mytripler(11))
```

```
33
```

we will look at some higher order functions of python

1. Map
2. Filter
3. Reduce

1. Map

- The map() function executes a specified function for each item in an iterable(i.e, List, Tuple, Set etc). The item is sent to the function as a parameter.
- map will work on every item of the given iterables

Syntax

- map(function, iterables)
  - Parameter - Description
  - function - Required. The function to execute for each item
  - iterable - Required. A sequence, collection or an iterator object. You can send as many iterables as you like, just make sure the function has one parameter for each iterable.

In [31]:

```
# Make new fruits by sending two iterable objects into the function:

def myfunc(a, b):
    return a + b
x = map(myfunc, ('apple', 'banana', 'cherry'),
        ('orange', 'lemon', 'pineapple'))

print(x)
# it will return the output as a map object which we can not see
# so in the below cell we will do the type conversion into the list
```

```
<map object at 0x000001C29342CEE0>
```

In [32]:

```
#convert the map into a list, for readability:
print(list(x))
```

```
['appleorange', 'bananalemon', 'cherrypineapple']
```

In [33]:

```
L=[1,2,3,4,5,6,7]
L
```

Out[33]:

```
[1, 2, 3, 4, 5, 6, 7]
```

In [34]:

```
map(lambda x:x*2,L)
```

Out[34]:

```
<map at 0x1c293423100>
```

In [35]:

```
list(map(lambda x:x*2,L))
```

Out[35]:

```
[2, 4, 6, 8, 10, 12, 14]
```

In [36]:

```
# we can store that map into a variable and then convert the variable
a=map(lambda x:x*2,L)
list(a)
```

Out[36]:

```
[2, 4, 6, 8, 10, 12, 14]
```

In [37]:

```
list(map(lambda x:x%2==0,L))
```

Out[37]:

```
[False, True, False, True, False, True, False]
```

In [38]:

```
list(map(lambda x: "even" if x%2==0 else "odd",L))
```

Out[38]:

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

In [39]:

```
# fetch names from a list of dict

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

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

Out[39]:

```
['Rahul', 'Nitish', 'Disha']
```

2. Filter

- The filter() function returns an iterator were the items are filtered through a function to test if the item is accepted or not.
- filter function will work only on filtrate conditions

Syntax

filter(function, iterable)

```
Parameter |    Description
function  -    A Function to be run for each item in the iterable
iterable  -    The iterable to be filtered
```

In [40]:

```
L = [1, 2, 3, 4, 5, 6, 7]
L
```

Out[40]:

```
[1, 2, 3, 4, 5, 6, 7]
```

In [41]:

```
a=filter(lambda x:x>4,L)
print(filter(a,L))

# this line will give us the filter object which we can not see
# so we will convert it in to list form
list(a)
```

```
<filter object at 0x000001C29341AE80>
```

Out[41]:

```
[5, 6, 7]
```

In [42]:

```
# we want fruit names which contains the charecter e
fruit=['apple','orange','mango','guava']
list(filter(lambda fruit: 'e' in fruit,fruit))
```

Out[42]:

```
['apple', 'orange']
```

In [43]:

```
# Filter the list, and return a new list
# with only the values equal to or above 18:
ages = [5, 12, 17, 18, 24, 32]

def myFunc(i):
    if i < 18:
        return False
    else:
        return True

adults = filter(myFunc, ages)

for i in adults:
    print(i)
```

```
18
24
32
```

3. Reduce

- The reduce(fun,seq) function is used to apply a particular function passed in its argument to all of the list elements mentioned in the sequence passed along.This function is defined in “functools” module.

Working :

- At first step, first two elements of sequence are picked and the result is obtained.
- Next step is to apply the same function to the previously attained result and the number just succeeding the second element and the result is again stored.
- This process continues till no more elements are left in the container.
- The final returned result is returned and printed on console.

In [44]:

```
# python code to demonstrate working of reduce()
# we have to import the functool module first

# importing functools for reduce()
import functools

# initializing list
list = [1, 3, 5, 6, 2]

# using reduce to compute sum of list
print("The sum of the list elements is : ", end="")
print(functools.reduce(lambda a,b : a+b, list))
```

```
The sum of the list elements is : 17
```

In [45]:

```
# using reduce to compute maximum element from list
print("The maximum element of the list is : ", end="")
print(functools.reduce(lambda a, b: a if a > b else b, list))
```

```
The maximum element of the list is : 6
```

In [46]:

```
L1=[12,34,56,11,21,58]
L1
```

Out[46]:

```
[12, 34, 56, 11, 21, 58]
```

In [47]:

```
functools.reduce(lambda x,y: x if x>y else y, L1)
# taking out the max value
```

Out[47]:

```
58
```

In [48]:

```
functools.reduce(lambda x,y: x if x<y else y, L1)
# taking out the minimum value
```

Out[48]:

```
11
```

List Comprehensions

- List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

In [49]:

```
L=[1,2,3,4,5,6,7]
L
```

Out[49]:

```
[1, 2, 3, 4, 5, 6, 7]
```

In [50]:

```
L1=[ x*2 for x in L]
L1
```

Out[50]:

```
[2, 4, 6, 8, 10, 12, 14]
```

In [51]:

```
# we want list with numbers which are
# square of the numbers in range upto 10
L2=[i**2 for i in range(10)]
L2
```

Out[51]:

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In [52]:

```
# we want square of odd numbers
L3=[i**2 for i in range(10) if i%2!=0]
L3
```

Out[52]:

```
[1, 9, 25, 49, 81]
```

In [53]:

```
# Based on a list of fruits, you want a new list,
# containing only the fruits with the letter "a" in the name.
# Without list comprehension you will have to write
# a for statement with a conditional test inside:
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
    if "a" in x:
    newlist.append(x)

print(newlist)
```

```
  File "C:\Users\gadha\AppData\Local\Temp\ipykernel_9896\2678920804.py", l
ine 10
    newlist.append(x)
    ^
IndentationError: expected an indented block
```

In [54]:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)
```

```
['apple', 'banana', 'mango']
```

In [55]:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
L4=[fruit for fruit in fruits if fruit[-1]=='a']
L4
```

Out[55]:

```
['banana']
```

Dictionary comprehensions

- Like List Comprehension, Python allows dictionary comprehensions. We can create dictionaries using simple expressions. A dictionary comprehension takes the form {key: value for (key, value) in iterable}

In [56]:

```
D={"name":"himanshu","gender":"male","age":23}
D.items()
# this will give dictionary items in the tuple form stored in the list
```

Out[56]:

```
dict_items([('name', 'himanshu'), ('gender', 'male'), ('age', 23)])
```

In [57]:

```
D1={key:value for key,value in D.items() if len(key)>3}
D1
```

Out[57]:

```
{'name': 'himanshu', 'gender': 'male'}
```

In [58]:

```
# Python code to demonstrate dictionary comprehension

# Lists to represent keys and values
keys = ['a','b','c','d','e']
values = [1,2,3,4,5]

# but this line shows dict comprehension here
myDict = { k:v for (k,v) in zip(keys, values)}
myDict
```

Out[58]:

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
```

In [59]:

```
# We can use below too
myDict = dict(zip(keys, values))

print (myDict)
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
```

In [60]:

```
# Using dictionary comprehension make dictionary

L=[1,2,3,4,5]

# Python code to demonstrate dictionary
# creation using list comprehension

myDict = {item: item**2 for item in L }
print (myDict)
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

In [61]:

```
a='coding'
sDict = {character.upper(): character*3 for character in a}
print (sDict)
```

```
{'C': 'ccc', 'O': 'ooo', 'D': 'ddd', 'I': 'iii', 'N': 'nnn', 'G': 'ggg'}
```

In [62]:

```
# Using conditional statements in dictionary comprehension

# This example below maps the numbers
# to their cubes that are not divisible by 4.
# normal syntax without use of if

newdict1 = {item: item**3 for item in range(10) }
print(newdict1)

# comprehension using if
newdict = {item: item**3 for item in range(10)  if item**3 % 4 == 0}
print(newdict)
```

```
{0: 0, 1: 1, 2: 8, 3: 27, 4: 64, 5: 125, 6: 216, 7: 343, 8: 512, 9: 729}
{0: 0, 2: 8, 4: 64, 6: 216, 8: 512}
```