Lambda functions

- Lambda functions represnts function concept
- But we can write in a single line
- Like list comprehension, lambda functions similar only
- It will decrease the time complexity
- always remember if we use many for loops or many conditions using multiple line , the time complexity will increase

In [ ]:
```
**It will use a keyword lambda**

# lambda <arguments>: <expression>
```

In [ ]:
```
What is the interview process for a data science position with 3 years of experi

1) coding+sql
2) based on ML DL
3) Techn: 2,3

bond : 1.5 bonus

ds2 === 2 2ppl
ds3 ====

do you know
```

In [ ]:
```
strings
list
tuple set
dictionary
landad
file handling

oops : kwargs

Sir do we have right to break bond with company
```

In [1]:
```
def summ(n):
    return(n+10)
summ(10)
```

Out[1]:
```
20
```

$Pattern-1$

Function with only one argument

- function name
- argument name

- return output

In [ ]:
```
# syntax: <function name>= lambda <argument name>: <return output>
def summ(n):
    return(n+10)
summ(10)

# function name= summ
# argument name = n
# return output= n+10
```

In [4]:
```
summ= lambda n:n+10
summ(100)
```

Out[4]: 110

In [5]:
```
def cube(n):
    return(n**3)
cube(10)
```

Out[5]: 1000

In [7]:
```
cube= lambda n:n**3
cube(10)
```

Out[7]: 1000

pattern-2

Two arguments

In [8]:
```
# syntax :  <function name>= lambda <arg1>,<arg2>: <return output>
def add(a,b):
    return(a+b)
add(50,50)

# Function name: add
# arg1=a
# arg2=b
# return= a+b
```

Out[8]: 100

In [10]:
```
add=lambda n1,n2: n1+n2
add(60,50)
```

Out[10]: 110

In [13]:
```
average=lambda a,b,c:round((a+b+c)/3,2)
average(10,202,30)
```

Out[13]: 80.67

Pattern-3

Default arguments

In [14]:
```
average=lambda a,b,c=500:round((a+b+c)/3,2)
average(10,202)
```

Out[14]:
```
237.33
```

Pattern-4

if-else

In [15]:
```
def max(a,b):
    if a>b:
        return(a)
    else:
        return(b)
max(10,20)
```

Out[15]:
```
20
```

In [17]:
```
# syntax : function name = lambda <arg1>,<arg2>: <list comprehension>
# syntax : function name = lambda <arg1>,<arg2>: <if_out><if_con><else><else_out
maxx=lambda a,b: a if a>b else b
maxx(30,20)
```

Out[17]:
```
30
```

Pattern-5

using List

In [18]:
```
l=['hyd','chennai','mumbai']
# op=['Hyd','Chennai','Mumbai']
op=[]
for i in l:
    op.append(i.capitalize())
op
```

Out[18]:
```
['Hyd', 'Chennai', 'Mumbai']
```

In [ ]:
```
lambda <variable>:<op>
# variable:i
# op: i.capitalize()
lambda <variable>:<op>,<iterator>
# Qn: from where you are getting 'i'
# <iterator>: list
```

map

- the function and iterator are available now
- we need to map both

In [21]:
```
l=['hyd','chennai','mumbai']
lambda i: i.capitalize(),l
```

Out[21]:
```
(<function __main__.<lambda>(i)>, ['hyd', 'chennai', 'mumbai'])
```

In [22]:
```
l=['hyd','chennai','mumbai']
map(lambda i: i.capitalize(),l)
```

Out[22]:
```
<map at 0x227209e9cf0>
```

In [23]:
```
# apply the list to see the values
l=['hyd','chennai','mumbai']
list(map(lambda i: i.capitalize(),l))
```

Out[23]:
```
['Hyd', 'Chennai', 'Mumbai']
```

In [24]:
```
l=['hyd','chennai','mumbai']
tuple(map(lambda i: i.capitalize(),l))
```

Out[24]:
```
('Hyd', 'Chennai', 'Mumbai')
```

In [ ]:
```
# step1: Write your normal expression
#        ex: lambda <var>: <op>===>lambda i: i.capitalize()
# step2: add the iterator
#        ex: lambda <var>: <op>,<list>===>lambda i: i.capitalize(),list1
# Step-3: Map the both
#        ex: map(lambda <var>: <op>,<list>)===>map(lambda i: i.capitalize(),list
# Step-4: save the values in a list,
#       ex: list(map(lambda <var>: <op>,<list>))===>list(map(lambda i: i.capital

#Note: Those who are getting list object not callable use tuple
```

In [ ]:
```
- Case-1: Function call with One argument

    - lambda arguments :  Exression

    - lambda variables :  return output

- Case-2: Function call with Two arguments

    - lambda arg1,arg2 :  Exression

    - lambda var1,var2 :  return output

- Case-3: Function call with Default arguments

    - lambda arg1,arg2=500 :  Exression

    - lambda var1,var2=500 :  return output

- Case-4: Function call with Two arguments and if-else statement

    - lambda arg1,arg2 :  Expression

    - lambda var1,var2 :  if_output  if_con  else  els_op  forloop

- Case-5: Lambda operations using List

      - lambda arg:  Expression,iterator

      - map(lambda var: operation,list)

      - list(map(lambda var: operation,list))
```

In [7]:
```
l=['hyd','chennai','mumbai']
# op=['HYD','CHENNAI','MUMBAI']
#for i in l:
    #print(i.upper())

tuple(map(lambda i:i.upper(),l))
```

Out[7]:
```
('HYD', 'CHENNAI', 'MUMBAI')
```

Filter

- whenever if conditions are there use filter

In [12]:
```
l=['hyd','che#nnai','mum#bai','blr']
#op=['che#nnai','mum#bai']
# for i in l:
#     if '#' in i:
#         print(i)

# Mistake-1: tuple(map(lambda i:if '#' in i,l))
#            do not write if

tuple(map(lambda i:'#' in i,l))
```

Out[12]:
```
(False, True, True, False)
```

In [13]:
```
tuple(filter(lambda i:'#' in i,l))
```

Out[13]:
```
('che#nnai', 'mum#bai')
```

In [14]:
```
# numbers= [1,3,2,7,6]
# op=[2,6]

l1 = [1,2,3,4,5,6,7]
list(filter(lambda i:i%2==0,l1))
```

Out[14]:
```
[2, 4, 6]
```

In [ ]:
```
**Reduce**

- All inbulit functions can achieve by Reduce

- Reduce is avialable from functools package

- level-1: reduce(lambda summ,i:summ+i,l1)

- level-2: reduce(lambda summ,i:summ+i,l1,intial_value)

- For example we want intialize summ=0 then we choose level-1

- For exaple we want start with other than zero then we choose level-2
```

In [16]:
```
l1=[1,2,3,4,5]

# I want sum of the elements in a list

# Method-1: sum
sum(l1)

# Method-2: with out sum
summ=0
for i in l1:
    summ=summ+i

print(summ)
```

```
15
```

In [21]:
```
l1=[1,2,3,4,5]
filter(lambda summ,i:summ+i,l1)
```

Out[21]:
```
<filter at 0x1f710644280>
```

In [26]:
```
l1=[1,2,3,4,5]
max(l1)
min(l1)
len(l1)
sum(l1)
```

Out[26]:
```
15
```

In [27]:
```
import functools
```

In [28]:
```
dir(functools)
```

Out[28]:
```
['GenericAlias',
 'RLock',
 'WRAPPER_ASSIGNMENTS',
 'WRAPPER_UPDATES',
 '_CacheInfo',
 '_HashedSeq',
 '_NOT_FOUND',
 '__all__',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 '_c3_merge',
 '_c3_mro',
 '_compose_mro',
 '_convert',
 '_find_impl',
 '_ge_from_gt',
 '_ge_from_le',
 '_ge_from_lt',
 '_gt_from_ge',
 '_gt_from_le',
 '_gt_from_lt',
 '_initial_missing',
 '_le_from_ge',
 '_le_from_gt',
 '_le_from_lt',
 '_lru_cache_wrapper',
 '_lt_from_ge',
 '_lt_from_gt',
 '_lt_from_le',
 '_make_key',
 '_unwrap_partial',
 'cache',
 'cached_property',
 'cmp_to_key',
 'get_cache_token',
 'lru_cache',
 'namedtuple',
 'partial',
 'partialmethod',
 'recursive_repr',
 'reduce',
 'singledispatch',
 'singledispatchmethod',
 'total_ordering',
 'update_wrapper',
 'wraps']
```

In [29]:
```
import functools
l1=[1,2,3,4,5]
functools.reduce(lambda summ,i:summ+i,l1)
```

Out[29]:
```
15
```

In [30]:
```
#import functools
#functools.reduce

from functools import reduce
l1=[1,2,3,4,5]
reduce(lambda summ,i:summ+i,l1)
```

Out[30]: 15

In [31]:
```
import functools as ft
l1=[1,2,3,4,5]
ft.reduce(lambda summ,i:summ+i,l1)
```

Out[31]: 15

In [ ]:
```
import functools
l1=[1,2,3,4,5]
functools.reduce(lambda summ,i:summ+i,l1)

from functools import reduce
l1=[1,2,3,4,5]
reduce(lambda summ,i:summ+i,l1)

import functools as ft
l1=[1,2,3,4,5]
ft.reduce(lambda summ,i:summ+i,l1)
```

In [ ]:
```

```

In [32]:
```
import functools
l1=[1,2,3,4,5]
functools.reduce(lambda mul,i:mul*i,l1)
```

Out[32]: 120

In [ ]:
```
import functools
l1=[1,2,3,4,5]
# 1*2*3*4*5:functools.reduce(lambda x,y:x*y,l1)
# 1+2+3+4+5:functools.reduce(lambda x,y:x+y,l1)
```

In [33]:
```
# Method-2: with out sum
summ=0
for i in l1:
    summ=summ+i

print(summ)

# Other method
import math
math.sqrt(25)
```

15

Out[33]: 5.0

In [34]:
```
import numpy
numpy.mean([1,2,3,4,5])
```

Out[34]:
```
3.0
```

In [35]:
```
import functools
```

In [36]:
```
functools
```

Out[36]:
```
<module 'functools' from 'C:\\Users\\omkar\\anaconda3\\Lib\\functools.py'>
```

In [ ]:
```
# Maximum value using Reduce

# Map : Direct attack
# filter: if conditions
# Reduce : inbulit functions some intialization

# I want detailed word document with examples
```