In [ ]:
```
m1 = <output> <for loop>
m2 = <if output> <for loop> <if condition>
m3 = <if output> <if condition> <else> <else output> <for loop>
m4 = <if output> <if cond> <else> <if output> <if cond> <else> <else output> <fo
```

2-Dec

In [1]:
```
def mul(a):
    return(a*a)

ans = mul(10)
ans
```

Out[1]:
```
100
```

- lambda is a keyword used to write function in a single line
- like list comprehension we will write in a single line
- it is same analogy
- it will reduced the time complexity

- A lambda function is a small anonymous function written in one line. It's often used with functions like map(), filter(), or for quick logic

$pattern - 1$

- function with a one argument

In [ ]:
```
# Syntax

# function name = lambda <argument name> : <return output>
```

In [ ]:
```
# in above
# function name : mul
# argument name : a
# return output : a*a
```

In [3]:
```
mul = lambda a: a*a
mul
```

Out[3]:
```
<function __main__.<lambda>(a)>
```

In [4]:
```
mul = lambda a: a*a
mul()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[4], line 2
      1 mul = lambda a: a*a
----> 2 mul()

TypeError: <lambda>() missing 1 required positional argument: 'a'
```

In [5]:
```
mul = lambda a: a*a
mul(10)
```

Out[5]: 100

In [6]:
```
def cube(x):
    return(x**3)

cube(10)
```

Out[6]: 1000

In [4]:
```
cube = lambda x: x**3
cube(10)
```

Out[4]: 1000

$patterns - 2$

- function with two arguments

In [ ]:
```
# syntax
<fun_name> = lambda <arg1>,<arg2>: return (output)
```

In [9]:
```
def add (a,b):
    return(a+b)
add(100,200)
```

Out[9]: 300

In [10]:
```
add1 = lambda a,b: a+b
add1(200,300)
```

Out[10]: 500

In [14]:
```
def avg(a,b,c):
    return(round((a+b+c)/3,2))

avg(10,15,25)
```

Out[14]: 16.67

In [17]:
```
avg = lambda a,b,c: round((a+b+c)/3,2)
avg(10,20,30)
```

Out[17]: 20.0

$pattern - 3$

- default arguments

In [18]:
```
avg2 = lambda a,b,c=500: round((a+b+c)/3,2)
avg2(109,229)
```

Out[18]: 279.33

$pattern - 4$

- if else

In [19]:
```
def max(a,b):
    if a>b:
        return(a)
    else:
        return(b)

max(10,20)
```

Out[19]: 20

In [ ]:
```
# syntax
# function name = lambda <arg1>,<arg2>: <if out> <if cond> <else> <else out>
```

In [21]:
```
maximum = lambda a,b: a if a>b else b
maximum(23,30)
```

Out[21]: 30

$Pattern - 5$

- List cases

In [23]:
```
list1 = ['hyd','chennai','mumbai','pune']

ans = []
for i in list1:
    ans.append(i.capitalize())

ans
```

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

In [ ]:
```
lambda <variable>: <op>
lambda i : i.capitalize()
```

In [24]:
```
lambda i : i.capitalize()
```

Out[24]: <function __main__.<lambda>(i)>

In [ ]:
```
lambda <variable>: <op>,<iterator>
```

In [25]:
```
lambda i : i.capitalize(),list1
```

Out[25]: (<function __main__.<lambda>(i)>, ['hyd', 'chennai', 'mumbai', 'pune'])

In [26]:
```
# Step-1: lambda <variable>: <output>
lambda i : i.capitalize()

#step-2: lambda <variable>:<output>,<iterator>
lambda i : i.capitalize(), list1
```

Out[26]:
```
(<function __main__.<lambda>(i)>, ['hyd', 'chennai', 'mumbai', 'pune'])
```

map

- map is used to combine the function and iterator

In [28]:
```
# step-3:
# map(lambda <variable>: <output>, <iterator>)
map(lambda i:i.captalize(),list1)
```

Out[28]:
```
<map at 0x278f37c76d0>
```

In [34]:
```
# Step-4: when we apply map the output will store at memory address
# so apply the list or tuple to see the answer
# list(map(labda <variable>: <op>,<iterator>))
```

In [33]:
```
list(map(lambda i:i.capitalize(),list1))
```

Out[33]:
```
['Hyd', 'Chennai', 'Mumbai', 'Pune']
```

In [32]:
```
tuple(map(lambda i:i.capitalize(),list1))
```

Out[32]:
```
('Hyd', 'Chennai', 'Mumbai', 'Pune')
```

In [35]:
```
lambda i : i.capitalize()
lambda i : i.capitalize(),list1
map(lambda i : i.capitalize(),list1)
list(map(lambda i : i.capitalize(),list1))
```

Out[35]:
```
['Hyd', 'Chennai', 'Mumbai', 'Pune']
```

In [ ]:
```
# Step-1: lambda <variable>: <output>
#step-2: lambda <variable>:<output>,<iterator>
# step-3: map(lambda <variable>: <output>, <iterator>)
# step-4: list(map(lambda <variable>: <output>, <iterator>))
```

100% comes in interview all above 4 step and lambda function

In [36]:
```
list1 = ['hyd','chennai','mumbai','pune']

ans = []
for i in list1:
        ans.append(i.upper())

ans
```

Out[36]:
```
['HYD', 'CHENNAI', 'MUMBAI', 'PUNE']
```

In [40]:
```
list(map(lambda i : i.upper(),list1))
```

Out[40]: ['HYD', 'CHENNAI', 'MUMBAI', 'PUNE']

In [6]:
```
list1 = ['hyd','chennai','mumbai','pune']

ans = []

for i in list1:
    ans.append(list1.index(i)*10)
ans
list(map(lambda i : list1.index(i)*10, list1 ))
tuple(map(lambda i : list1.index(i)*10, list1 ))
```

Out[6]: (0, 10, 20, 30)

In [52]:
```
list(map(lambda i : list1.index(i)*10,list1))
```

Out[52]: [0, 10, 20, 30]

In [67]:
```
# str1 = 'hello hai how are you'
str1 = 'hello hai how are you'
# ans = ['Hello','Hai','How','Are','You']
ans=[]
for i in str1:
    print(i.title())
```

```
H
E
L
L
O

H
A
I

H
O
W

A
R
E

Y
O
U
```

In [55]:
```
ans = list(map(lambda i : i.title(),str1.split()))
ans
```

Out[55]: ['Hello', 'Hai', 'How', 'Are', 'You']

The below quetion ask in congnizant company by using lambda

In [69]:
```
#Que6:- ['hyd','chen#ai','mu#bai','pune']
# ans=['chen#ai', 'mu#bai']

l1 = ['hyd','chen#ai','mu#bai','pune']
```

```
for i in l1:
    if "#" in i:
        print(i)
```

```
chen#ai
mu#bai
```

In [70]:
```
lambda i : if "#" in i, l1
```

```
  Cell In[70], line 1
    lambda i : if "#" in i, l1
               ^
SyntaxError: invalid syntax
```

always remember using list comprehension and lambda

- when we use if after that 100% comes else otherwise it will through the error

In [ ]:
```
lambda <variable> : <condition>

# dont write if
# dont write i
```

- mistakes-1 map(lambda i : if "#" in i,l1)
  - don't write if
- mistake-2 list(filter(lambda i :i if "#" in i,l1))
  - don't write the output i
- direct attack

In [76]:
```
l1 = ['hyd','chen#ai','mu#bai','pune']

map(lambda i : "#" in i,l1)
```

Out[76]:
```
<map at 0x278f4b82530>
```

In [74]:
```
list(map(lambda i : "#" in i,l1))
```

Out[74]:
```
[False, True, True, False]
```

filter

- Whenever condition statements
- which means we are filtering the answers based on conditions
- map will give boolean outputs, True or False
- True answer we can see by applying filter only
- so use filter instead of map in above code

In [75]:
```
list(filter(lambda i : "#" in i,l1))
```

Out[75]: ['chen#ai', 'mu#bai']

In [77]:
```
l1 = [1,2,3,4,5]
summ =0
for i in l1:
    summ = summ+i

print(summ)
```

```
15
```

reduce

- reduce is a method to write all inbuilt functions using lambda method
- it is available in a package called **functools**
- Level-1: reduce(lambda sum,i: sum+i, list1)
  - in level-1 by default sum=0
- Level-2: reduce(lambda sum,i : sum+i, list1, value)
  - in level-2 the sum starts with some value

In [79]:
```
import functools
```

In [80]:
```
dir(functools)
```

Out[80]:
```
['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 [81]:
```
import functools
list1 = [1,2,3,4,5]
functools.reduce(lambda summ,i:summ+i,list1)
```

Out[81]:
```
15
```

In [83]:
```
import functools
list1 = [1,2,3,4,5]
functools.reduce(lambda summ,i:summ+i,list1,200)
```

Out[83]: 215

In [9]:
```
import functools
list1 = [1,2,3,4,5]
functools.reduce(lambda val,i:val*i,list1,3)
```

Out[9]: 360

3 way to use package

- 1. import functools
- 2. from functools import reduce
- 3. import functools as ft

In [ ]:
```
import functools
list1 = [1,2,3,4,5]
functools.reduce(lambda summ,i:summ+i,list1)
#########################################################

from functools import reduce
list1 = [1,2,3,4,5]
reduce(lambda summ,i:summ+i,list1)
#########################################################

import functools as ft
list1 = [1,2,3,4,5]
ft.reduce(lambda summ,i:summ+i,list1)
```

In [ ]:
```
lambda <variable>:<op>
lambda <variable>: <if op> <if cond> <else> <else op>
map(lambda <variable>: <op>,iterator)
filter(lambda <variable>: <con>, <iterator>)
reduce(lambda <var1>,<var2>: var1+var2, <iterator>)
```

map,filter,reduce are very very important

- Type will ask in interview 100%

In [ ]:
```

```