Lambda Functions

- A lambda function is a small anonymous function.
- A lambda function can take any number of arguments, but can only have one expression.

$syntax$:

In [ ]:
```
lambda arguments : expression
```

Pattern-1

- Function with one argument

In [2]:
```
#example:1
mul=lambda n:n*10
mul(3)
```

Out[2]: 30

In [5]:
```
#example:2
cube=lambda n:n**3
cube(3)
```

Out[5]: 27

Pattern-2

- Function with two arguments

In [6]:
```
add=lambda n1,n2:n1+n2
add(3,4)
```

Out[6]: 7

In [8]:
```
average=lambda a,b,c:round((a+b+c)/3,2)
average(1,4,6)
```

Out[8]: 3.67

Pattern-3

- Default Arguments

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

Out[10]: 42

In [11]:
```
avg=lambda c,b=30,a=10:round((a+b+c),2)
avg(10)
```

Out[11]: 50

In [12]:
```
avg=lambda c=1,b=30,a=10:round((a+b+c),2)
avg()
```

Out[12]: 41

Pattern-4

- using if-else condition

In [13]:
```
maxx=lambda a,b:a if a>b else b
maxx(3,-1)
```

Out[13]: 3

In [17]:
```
maxx=lambda a,b,c:a if (a>=b and a>=c) else (b if b>=c else c)
maxx(3,-1,8)
```

Out[17]: 8

Pattern-5

- Using List

In [18]:
```
l=['hyd','chennai','mumbai']
lambda i:i.capitalize(),l
```

Out[18]: (<function __main__.<lambda>(i)>, ['hyd', 'chennai', 'mumbai'])

Map

- The map function in python takes in a functions and list as an argument.
- The map function is used to apply the transformation to an iterable object like a list,tuple or set

Syntax of map

In [ ]:
```
map(lambda arguments: expression, iterable)
```

In [19]:
```
#example-1:
l=['hyd','chennai','mumbai']
map(lambda i:i.capitalize(),l)
```

Out[19]:
```
<map at 0x19b8c44cb80>
```

In [20]:
```
#apply list to get values
list(map(lambda i:i.capitalize(),l))
```

Out[20]:
```
['Hyd', 'Chennai', 'Mumbai']
```

filter

- The filter() is a built-in function that is used to filter the elements from the iterable object like list, set, tuple, etc. This function will directly filter the elements in an iterable by taking the condition as a function

- The filter() function in Python is used to filter elements from an iterable based on a specified condition. It takes a function and an iterable as arguments and returns an iterator containing only the elements for which the function returns True.

Syntax

In [ ]:
```
filter(function, iterable)
```

In [22]:
```
#example-1:
n=[1,2,3,4,5,6,7,8,9,10]
filter(lambda i:i%2==0,n)
```

Out[22]:
```
<filter at 0x19b8c457220>
```

In [26]:
```
list(filter(lambda i:i%2==0,n))
```

Out[26]:
```
[2, 4, 6, 8, 10]
```

In [24]:
```
tuple(filter(lambda i:i%2==0,n))
```

Out[24]:
```
(2, 4, 6, 8, 10)
```

In [27]:
```
tuple(filter(lambda i:i%2!=0,n))
```

Out[27]:
```
(1, 3, 5, 7, 9)
```

In [28]:
```
list(filter(lambda i:i%2!=0,n))
```

Out[28]:
```
[1, 3, 5, 7, 9]
```

Reduce

- All inbuilt functions are achieve by reduce
- Reduce is available from functools package

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

Out[29]: 15

In [30]:
```
from functools import reduce
l=[1,2,3,4,5]
reduce(lambda summ,i:summ+i,l)
```

Out[30]: 15

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

Out[31]: 15

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

Out[32]: 120

In [ ]: