

**DATA SCIENCE**

ASSIGNMENT - 1

NAME : NATARAJ.C

REGISTER NO : 121012012756

DEGREE : B.TECH

BRANCH : CSE –3rd YEAR

SPECIALIZATION : DATA SCIENCE

SUBJECT : APPLIED ARTIFICAL INTELLIGENCE

SUB.CODE : XCSHD3

1 . 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.**
* **Python Lambda Functions** are anonymous function means that the function is without a name. the *def* keyword is used to define a normal function in Python. Similarly, the *lambda* keyword is used to define an anonymous function in Python.

## Syntax

lambda arguments : expression

### Example

# Code to demonstrate how we can use a lambda function for adding 4 numbers

add = **lambda** num: num + 4

**print**( add(6) )

**Output :** 10

lambda is a keyword in Python for defining the anonymous function.

argument(s) is a placeholder, that is a variable that will be used to hold the value you want to pass into the function expression. A lambda function can have multiple variables depending on what you want to achieve.

expression is the code you want to execute in the lambda function.

## Why do we need a Lambda Function?

* When compared to a normal Python function written using the **def** keyword, lambda functions require fewer lines of code. However, this is not quite true because functions defined using def can be defined in a single line. But, **def**functions are usually defined on more than one line.
* They are typically employed when a function is required for a shorter period (temporary), often to be utilized inside another function such as filter, map, or reduce.
* You can define a function and call it immediately at the end of the definition using the lambda function. This is not possible with def functions.

## Use a Lambda Function?

The lambda function to create simple expressions. For example, expressions that do not include complex structures such as if-else, for-loops, and so on.

So, for example, if you want to create a function with a for-loop, you should use a user-defined function.

## Difference between def and Lambda functions :

| **def defined functions** | **lambda functions** |
| --- | --- |
| Easy to interpret | Interpretation might be tricky |
| Can consists of any number of execution statements inside the function definition | The limited operation can be performed using lambda functions |
| To return an object from the function, return should be explicitly defined | No need of using the return statement |
| Execution time is relatively slower for the same operation performed using lambda functions | Execution time of the program is fast for the same operation |
| Defined using the keyword def and holds a function name in the local namespace | Defined using the keyword lambda and does not compulsorily hold a function name in the local namespace |

## Appropriate Uses of Lambda Expressions

* Issues with readability
* The imposition of a functional way of thinking
* Heavy syntax with the lambda keyword

2 .The map() function

* The map() function allows you to iterate over each item in an iterable. Map(), on the other hand, operates independently on each item rather than producing a single result.
* Finally, the map() function can be used to perform mathematical operations on two or more lists. It can even be used to manipulate any type of array.
* The map() function’s time complexity= O (n)

### Syntax

map(function, iterable)

### Parameters

* **function** − The function to be used in the code.
* **iterable** − This is the value that is iterated in the code.

### Algorithm (Steps)

* Create a function with the name **multiplyNumbers** that returns the multiplication result of the number passed to it.
* Return the given number multiplied by 3, inside the function.
* Use the **map()** function for applying the multiplyNumbers() function for each element of the list by passing the function name, and list as arguments to it.
* Print the resultant list items after multiplying them with 3.

### Code

# creating a function that returns multiplication result

def multiplyNumbers(givenNumbers):

# returning number after multiplying with 3

return givenNumbers\*3

# map() function applies the multiplyNumbers function

# for each element of the list

givenNumbers = map(multiplyNumbers, [1, 3, 5, 2, 6])

# Printing the resultant list items

print("Multiplying list elements with 3:")

for element in givenNumbers:

print(element)

### Output

Multiplying list elements with 3:

3

9

15

6

18

The filter() function

* The filter() function creates a new iterator that filters elements from a previously created one (like a list, tuple, or dictionary).
* The filter() function checks whether or not the given condition is present in the sequence and then prints the result.
* The filter() function’s time complexity= O (n)

### Syntax

filter(function, iterable)

### Parameters

* **function** − The function to be used in the code.
* **iterable** − This is the value that is iterated in the code.

### Algorithm (Steps)

* Create a function with the name **votingAge** that returns the eligibility ages for voting from the list.
* Use the **if** conditional statement to check whether the number passed to the function is greater than or equal to 18.
* If the above statement is true Return the number.
* Create a variable to store the input list.
* Use the **filter()** function by passing the function name, and input list as arguments to it to filter the ages greater than or equal to 18 from the list. Here it applies the votingAge() function to every element of the list and the result stores only the values of the list that are returned by the votingAge() function(Here votingAge() function returns the number if it is greater than 18).
* Print the filter object
* Use the **list()** function(returns a list of an iteratable), to convert the above filter object into a list and print it.

### Example

# creating a function that returns the eligibility ages for voting from the

list

def votingAge(givenNumumber):

# checking whether the number is greater than or equal to 18

if givenNumumber>=18:

# returning number

return givenNumumber

# input list

inputList = [3, 20, 18, 6, 14, 25, 19]

# Getting only values of above list which are greater than or equal to 18

result\_filterObj = filter(votingAge, inputList)

# printing the filter object

print(result\_filterObj)

# converting into a list

print("Eligibility ages for voting from the input list:", list(result\_filterObj))

### Output

<filter object at 0x7fcd3ad14280>

Eligibility ages for voting from the input list: [20, 18, 25, 19]

### Difference between Map() vs Filter()

|  |  |
| --- | --- |
| **map() function** | **filter() function** |
| map() function always passes all the elements/iterables through the transform function. | Filter() function always check the condition in terms of Boolean values and then passes elements through the function. |
| Applied to all the iterables and return the same number of the iterator without filtration. | The result is printed in the form of a filter object, and those filter objects can be converted into a list. |
| The result is printed in the form of a map object for the map() function. We can convert those map objects into the list. | The map() function is the best alternative for looping in python. The mapping technique is implemented in the map() function. |
| Transform function can take N number of arguments. | The function used for checking conditions in the filter function must take only one argument. |
| The map() function is used as the best alternative for looping in python. The mapping technique is implemented in the map() function. | The filter function is used as a filter in python. |

3.ITERATORS:

Iterators are something that help to loop over different objects in Python. Iterator goes over all the values in the list. Iterators apply the iterator protocol that contains basically two methods:

* **\_iter\_()**
* **\_next\_()**

The difference between iterators vs iterable.  
Iterable objects are objects which you can iterate upon. Examples of iterables are tuples, dictionaries, lists, strings, etc.  
These iterables use **iter**() method to fetch the iterator.

Here is an example of **iter**() method fetching an iterator from a tuple.

## Code:

atuple = ('avocado', 'beetroot', 'berries')

myiter = iter(atuple)

print(next(myiter))

print(next(myiter))

print(next(myiter))

Output:  
avocado  
beetroot  
berries

An iterator actually represents a stream of data. Iterators follow the iterator protocol.  
Iterator protocol has **iter**() and **next**() methods. Both iterables and iterators have **iter**() method that fetches an iterator.  
The thing that differentiates between iterators and iterables is the **next**() method, that is part of just iterators.  
The next() method is used to fetch the next value of the iterable, whenever we use the print function with next.

Code :

class Sequence():

def \_\_init\_\_(self):

self.num = 1

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

value = self.num

if value > 10:

raise StopIteration

self.num += 2

return value

ite = Sequence()

print(next(ite))

print(next(ite))

print(next(ite))

print(next(ite))

**Output:**  
1  
3  
5  
7

The **\_\_init\_\_**() method is the first method that executes when any class gets called. It is used to assign values to the class variables during the program. I have declared num variable with one here.  
Next and iter are the methods that make this class an iterator.  
The iter method fetches the iterator and initiates the iteration. The class sequence is an iterator, thus it returns itself.  
The next method fetches the current value from the iterator and moves to the next state when the next call happens. We update the num var by two to make the output odd numbers.

PYTHON GENERATORS:

* Generators provide a better way to create iterators in Python. This can be done by defining a proper function instead of using a return statement that uses a yield keyword. Let’s see this with the help of an example.

**Code:**

def subjects():

yield "machine learning"

yield "business analytics"

yield "java"

yield "python"

for i in subjects():

print(i)

**Output:**  
machine learning  
business analytics  
java  
python

The yield function is similar to a return statement, but with some additional functionality – it actually remembers the state of the function.  
So next time when the generator is called it won’t start from scratch, rather from where it was last called.

## PERFORMANCE OF A GENERATOR:

The improvement in the performance of a generator is due to the yield function, as it leads to the on-demand creation of values. We can generate the values one by one. Unlike when return terminates the state of function, yield pauses the state and continues where it was left off.This yield function also makes the generator memory-efficient when compared with list comprehension.

## GENERATOR EXPRESSIONS:

Generator expressions are better at generating sequences and are memory-efficient. They are often compared to list comprehensions, as the way the code is written for both is similar.

**Code:**

1

2

3

4

5

6

7

#generator expression

Gen = (x \*\* 2

for x in range(4))

next(Gen)

next(Gen)

next(Gen)

next(Gen)

**Output:**  
0  
1  
4  
9

Generator expressions are memory and space-efficient as they do not need the whole list to be generated as in list comprehension. It can be used with large datasets.

### EXAMPLE:

* You can start a generator using yield and next functions.

4.Modules in Python

[In Python](https://www.scaler.com/topics/python/), a module can be a simple python file (**.py** extension file), i.e., a combination of numerous functions that we can use to provide different functionalities in a program.

Python Modules are essentially Python Programming Statements containing various types of Python Functions used to perform various operations in a Python Program. In the script, Python modules serve as a ready-made library available to programmers and users.

### Example :

**scalerAcad.py**

def module\_function(para):

print("Creating a new Module: " + para)

Make a new file with any name, say *test*.*py*, and paste the code below into it. Run it.

**test.py**

import scaler\_acad

scaler\_acad.module\_function("Scaler")

### Output:

Creating a new Module: Scaler

Explanation:

We create a new module named *scalerAcad*.*py* by creating a new Python file consisting of a function named **module\_function**, which takes a parameter and prints a statement.

Therefore, after creating the module, we use it in the *test*.*py* file by importing it, calling it with the parameters, and printing the statement.

### ****Syntax of Python Import****

import module

**Note:**This does not import the functions or classes directly instead imports the module only. To access the functions inside the module the dot(.) operator is used.

## Locating Modules

When you import a module, the Python interpreter searches for the module in the following sequences −

* The current directory.
* If the module isn't found, Python then searches each directory in the shell variable PYTHONPATH.
* If all else fails, Python checks the default path. On UNIX, this default path is normally /usr/local/lib/python/.

The module search path is stored in the system module sys as the **sys.path** variable. The sys.path variable contains the current directory, PYTHONPATH, and the installation-dependent default

~ >In Python, we can define a module in one of 3 ways:

* Python itself allows for the creation of modules.
* Similar to the re (regular expression) module, a module can be primarily written in C programming language and then dynamically inserted at run-time.
* A built-in module, such as the itertools module, is inherently included in the interpreter.

A module is a file containing Python code, definitions of functions, statements, or classes. An example\_module.py file is a module we will create and whose name is example\_module.

We employ modules to divide complicated programs into smaller, more understandable pieces. Modules also allow for the reuse of code.

Rather than duplicating their definitions into several applications, we may define our most frequently used functions in a separate module and then import the complete module.

Packages in Python

As we discussed earlier, to create large-scale-based real-world applications, we divide large code into smaller pieces to perform different functionalities, resulting in many modules. To collaborate with all of the modules, we create a Python package with an \_\_init\_\_.py file that informs the Python Interpreter that the given folder is a Python Package.

For any source code, a Python package serves as a user-variable interface. This functionality enables any functional runtime script to use a Python package at a specified moment, showing the main difference between the module and the package in Python.

**To import a package, we type the following:**

# import math

In the above code, **math** is a package.

Only its immediate modules are imported when we import a package, not the sub-packages. If you try to access those, it will raise an AttributeError.

## Python Package Different from Modules?

A Python package defines the code as a separate unit for each function when using a library. While the modules themselves are a distinct library with built-in functionality, the advantage of packages over modules is their **reusability**. So this is the difference between a module and a package in Python.

### Explicit Namespaces

It gives the program, which is interpreted for the first time, the default namespace. These namespaces serve as the source code for the coding's identification. However, a novice programmer can also integrate them from the library. However, it is always recommended to be familiar with general namespaces to execute code correctly.

**Code:**

# def Acad():

# para = "Scaler"

# Acad()

**Output:**

>>> Acad

<function namespace at 0x000001F91DE9FF70>

>>>

### Convenience API

Generally, this is a way to namespace specific code objects. It takes the user right to the core of the code, making it simple to see problems as well. Additionally, it aids in interpreting the codes to be used as user interface codes when needed.

**Code:**

# import hello

# hello.hey()

## ****Output:****

# Error

5.MATRIX OPERATIONS IN PYTHON

Matrix operation can perform some arithmetic operations like addition,subtraction and multiplication.

## MATRIX ADDITION:

These matrices can be added if (if and only if) the order of the matrices are equal, i.e. the two matrices have the same number of rows and columns.

**Adding elements of the matrix:**

# importing numpy as np

import numpy as np

# creating first matrix

A = np.array([[1, 2], [3, 4]])

# creating second matrix

B = np.array([[4, 5], [6, 7]])

print("Printing elements of first matrix")

print(A)

print("Printing elements of second matrix")

print(B)

# adding two matrix

print("Addition of two matrix")

print(np.add(A, B))

**OUTPUT FOR MATRIX ADDITION :**

Printing elements of first matrix

[[1 2]

[3 4]]

Printing elements of second matrix

[[4 5]

[6 7]]

Addition of two matrix

[[ 5 7]

[ 9 11]]

## MATRIX SUBTRACTION:

The subtraction of matrices is an operation where element-wise subtraction applies to the matrices of the same order, which essentially means that subtraction between two matrices can only happen when both of them have the same number of rows and columns.

**Subtracting elements of matrices:**

# importing numpy as np

import numpy as np

# creating first matrix

A = np.array([[1, 2], [3, 4]])

# creating second matrix

B = np.array([[4, 5], [6, 7]])

print("Printing elements of first matrix")

print(A)

print("Printing elements of second matrix")

print(B)

# subtracting two matrix

print("Subtraction of two matrix")

print(np.subtract(A, B))

**OUTPUT FOR MATRIX SUBTRACTION :**

Printing elements of first matrix

[[1 2]

[3 4]]

Printing elements of second matrix

[[4 5]

[6 7]]

Subtraction of two matrix

[[-3 -3]

[-3 -3]]

## MATRIX MULTIPLICATION :

Matrix multiplication is a binary operation that uses a pair of matrices to produce another matrix. The elements within the matrix are multiplied according to elementary arithmetic.

In the multiplication of two matrices, the row elements of the first matrix are multiplied to the column elements of the second matrix.

**Matrix Multiplication using Nested Loop :**

# Program to multiply two matrices using nested loops

# 3x3 matrix

X = [[12,7,3],

[4 ,5,6],

[7 ,8,9]]

# 3x4 matrix

Y = [[5,8,1,2],

[6,7,3,0],

[4,5,9,1]]

# result is 3x4

result = [[0,0,0,0],

[0,0,0,0],

[0,0,0,0]]

# iterate through rows of X

for i in range(len(X)):

# iterate through columns of Y

for j in range(len(Y[0])):

# iterate through rows of Y

for k in range(len(Y)):

result[i][j] += X[i][k] \* Y[k][j]

for r in result:

print(r)

**OUTPUT FOR MATRIX MULTIPLICATION :**

[114, 160, 60, 27]

[74, 97, 73, 14]

[119, 157, 112, 23]

Numpy methods for matrix manipulations

* **numpy.add()** − Add two matrices
* **numpy.subtract()** − Subtract two matrices
* **numpy.divide()** − Divide two matrices
* **numpy.multiply()** − Multiply two matrices

### Example :

import numpy as np

# Two matrices

mx1 = np.array([[5, 10], [15, 20]])

mx2 = np.array([[25, 30], [35, 40]])

print("Matrix1 =\n",mx1)

print("\nMatrix2 =\n",mx2)

# The addition() is used to add matrices

print ("\nAddition of two matrices: ")

print (np.add(mx1,mx2))

# The subtract() is used to subtract matrices

print ("\nSubtraction of two matrices: ")

print (np.subtract(mx1,mx2))

# The divide() is used to divide matrices

print ("\nMatrix Division: ")

print (np.divide(mx1,mx2))

# The multiply()is used to multiply matrices

print ("\nMultiplication of two matrices: ")

print (np.multiply(mx1,mx2))

### Output

Matrix1 =

[[ 5 10]

[15 20]]

Matrix2 =

[[25 30]

[35 40]]

Addition of two matrices:

[[30 40]

[50 60]]

Subtraction of two matrices:

[[-20 -20]

[-20 -20]]

Matrix Division:

[[0.2 0.33333333]

[0.42857143 0.5 ]]

Multiplication of two matrices:

[[125 300]

[525 800]]