ASSIGNMENT 1

NAME	: S AKSHAYA SARAVANAN
REG NO	: 121011012719
COURSE NAME	: APPLIED AI
COURSE CODE	: XCSHDO3
DEPT	: BTech CSE

What is Lambda Function?

A lambda function is an anonymous function (i.e., defined without a name) that can take any number of arguments but, unlike normal functions, evaluates and returns only one expression. They are typically used when you need a small, one-time function and don't want to define a separate named function.

The syntax for a lambda function is

```
lambda arguments: expression
```

The breakdown of the above syntax is :

- lambda : The keyword that indicates you're creating a lambda function.
- arguments : The input parameters for the function.
- expression: The single expression that the function evaluates and returns.

A example to implement lambda function that takes two arguments and returns their sum .

```
add = lambda a, b: a+b

result = add (3,5)

At last this will assign '8' to the 'result' variable
```

Why the use of the Lambda Function?

The lambda functions in python is mini – functions we create on the spot for fast and simple tasks. They're like quick calculations we can use right away without naming it. They're not meant for complex tasks, so at that time it is better to define a regular named function using 'def' keyword.

Applications of Lambda Function

Lambda functions are short-lived functions. Common applications of lambda function are:

- **Sorting:** Lambda functions are often used as the ‘key’ argument in sorting functions like ‘sorted ()’ to customize the sorting.

```
data = [(1,5),(3,2) ,(8,9)]
sorted_data = sorted (data, key = lambda x:x [1])
print(sorted_data)

Output:
[(3,2), (1,5), (8,9)]
```

- **Filtering**: Lambda functions can be used with functions like ‘filter ()’to select elements from a collection to satisfy a certain condition.

```
numbers = [1,2,3,4,5,6]
even_numbers = list (filter (lambda x: x % 2 == 0, numbers))
print(even_numbers)

Output:
[2,4,6]
```

- **Mapping**: Lambda functions can be applied using functions like ‘map ()’ to perform an operation on each element of a collection .

```
numbers = [1,2,3,4,5,6]
squared_numbers = list (map (lambda x: x ** 2, numbers))
print(squared_numbers)

Output:
[1, 4, 9, 16, 25]
```

Iterator in Python

In Python, an iterator is an object used to iterate over iterable objects such as lists, tuples, dictionaries, and sets. An object is called iterable if we can get an iterator from it or loop over it.

Basically an iterator is an object that allows you to loop through a collection of items like a string, one item at a time. It keeps track of the current position so that you easily move from one item to next.

Example of how an iterator works :

```
fruits = [“apple”, “banana”, “grapes”]

#Creating an iterator from the list
fruit_iterator = iter(fruits)

print(next(fruit_iterator))
print(next(fruit_iterator))
print(next(fruit_iterator))

Output:
apple
banana
grapes
```

Generators in Python

Python has a generator that allows you to create your iterator function. A generator is somewhat of a function that returns an iterator object with a succession of values rather than a single item. A yield statement, rather than a return statement, is used in a generator function.

The difference is that, although a return statement terminates a function completely, a yield statement pauses the function while storing all of its states and then continues from there on subsequent calls.

A example of generator that yields even numbers :

```
def even_numbers(limit):
       for num in range(2,limit +1, 2):
             yield num

#Creating a generator object
even_gen =  even_numbers(10)

#Using the generator to yield even numbers

for number in even_gen :
      print(number)

Output:
2
4
6
8
10
```

Module in Python

Python Module is essentially a Python script file that can contain variables, functions, and classes. Python modules help us organize our code and then reference them in other classes or Python scripts.,

A file containing Python definitions and statements is called a Python module. So naturally, the file name is the module name which is appended with the suffix .py

A example for module:

We have a file named math_operations.py that contains some mathematical functions

```
def add(a,b):
      return a+b
def subtract(a,b):
       return a-b
```

Now we are going to use this module in another Python script:

```
#main.py

import math_operations

result = math_operations.add(5,3)
print(result)

result = math_operations.subtract(10,4)
print(result)

Output:
8
6
```

Packages in Python

A package is actually a folder containing one or more module files.

Creating a directory named ‘my_package’ and placing the module inside

```
my_package/
        --init--.py
        math_operations.py

#my_package/math_operations.py

def add(a,b):
      return a+b
def subtract(a,b):
       return a-b

#main.py

from my_package import math_operations
result = math_operations.add(5,3)
print(result)

result = math_operations.subtract(10,4)
print(result)

Output: 8 & 6
```

Matrix Operations in Pandas

Matrix operations in pandas refer to the mathematical operations performed on matrices or arrays using the pandas library. These operations include addition, subtraction, multiplication, and division of matrices.

```
import pandas as pd

# Create two DataFrames representing matrices
matrix1 = pd.DataFrame([[1, 2], [3, 4]], columns=['A', 'B'])
matrix2 = pd.DataFrame([[5, 6], [7, 8]], columns=['A', 'B'])

# Display the matrices
print("Matrix 1:")
print(matrix1)
print("\nMatrix 2:")
print(matrix2)

# Matrix addition
matrix_sum = matrix1 + matrix2
print("\nMatrix Sum:")
print(matrix_sum)

# Matrix multiplication
matrix_product = matrix1.dot(matrix2)
print("\nMatrix Product:")
print(matrix_product)
```

Output:

```
Matrix 1:
   A  B
0  1  2
1  3  4

Matrix 2:
   A  B
0  5  6
1  7  8

Matrix Sum:
    A   B
0   6   8
1  10  12

Matrix Product:
    A   B
0  19  22
1  43  50
```