PERIYAR
MANIAMMAI
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University)
Established Under Sec. 3 of UGC Act, 1956 • NAAC Accredited

think • innovate • transform

NAME : A.M.AKSHAYA

REG NO : 121011012718

COURSE NAME : APPLIED AI

COURSE CODE : XCSHDO3

DEPARTMENT : B.TECH(CSE)

1.Lambda Function

Definition:

A lambda function in Python is a small, inline, and anonymous function that can have any number of arguments, but can only have one expression. These functions are primarily used for simple operations where defining a full function using the def keyword might be overly verbose.

Syntax:

lambda arguments: expression

Example:

```
# Regular function
def square(x):
    return x ** 2
# Equivalent lambda function
square_lambda = lambda x: x ** 2
print(square(5))                # Output: 25
print(square_lambda(5))    # Output: 25
```

2.Map and Filters:

Definition:

map() is a built-in Python function that takes a function and an iterable (like a list) and applies the function to each element of the iterable, returning an iterator with the results.

filter() is a built-in Python function that takes a function and an iterable, and returns an iterator containing only the elements from the iterable that satisfy the given condition in the function.

Syntax:

map(function, sequence)

filter(function, sequence)

Example:

```
numbers = [1, 2, 3, 4, 5]

# Using map to double each number

doubled = list(map(lambda x: x * 2, numbers))

# Using filter to get even numbers

evens = list(filter(lambda x: x % 2 == 0, numbers))

print(doubled)    # Output: [2, 4, 6, 8, 10]

print(evens)      # Output: [2, 4]
```

3.Iterators and Generators:

Definition:

An iterator is an object in Python that implements the methods __iter__() and __next__(). The __iter__() method returns the iterator object itself, and the __next__() method returns the next value from the iterator. Iterators are used to enable the iteration over a collection of items.

A generator is a special type of iterator that is created using a function with the yield keyword. When the function is called, it returns a generator object that can be iterated over using the next() function or a for loop. Generators are useful for creating iterators with minimal code.

Syntax (Iterator):

```
class MyIterator:

    def __iter__(self):

        return self
```

```
        def __next__(self):


  # Define iteration logic
        pass
```

Syntax (Generator):

```
def my_generator():
    yield value
    # More yields
gen = my_generator()
```

Example:

```
# Iterator example
class MyRange:
    def __init__(self, start, end):
        self.current = start
        self.end = end
      def __iter__(self):
        return self
        def __next__(self):
        if self.current >= self.end:
            raise StopIteration
        value = self.current
        self.current += 1
        return value


# Generator example
def countdown(n):
    while n > 0:
        yield n
```

```
        n -= 1

# Using the iterator
my_range = MyRange(1, 4)
for num in my_range:
    print(num)  # Output: 1, 2, 3

# Using the generator
for num in countdown(3):
    print(num)  # Output: 3, 2, 1
```

4. Modules and Packages:

Definition:

A module in Python is a single file containing Python code, which can include variables, functions, and classes. A package is a collection of related modules grouped together within a directory. Packages help organize and structure larger codebases.

Example:

Suppose you have a package named my_package containing two modules: module1.py and module2.py.

```
my_package/
    __init__.py
    module1.py
    module2.py
```

You can access functions/classes from these modules like this:

```
from my_package import module1
from my_package.module2 import some_function
```

module1.function_from_module1()

some_function()

5. Matrix Operations in Pandas:

Definition:

Pandas is a powerful data manipulation library in Python that provides data structures and functions for efficiently working with structured data, such as tables.

Example:

In the provided example:

Creating a DataFrame

data = {'A': [1, 2, 3], 'B': [4, 5, 6]}

df = pd.DataFrame(data)

Here, a pandas DataFrame df is created using a dictionary data. The keys of the dictionary become the column names, and the values become the data in the respective columns.

Transposing the DataFrame

transposed = df.T

The .T attribute transposes the DataFrame, swapping rows and columns.

Performing matrix multiplication

matrix_product = df.dot(transposed)

The .dot() method is used to perform matrix multiplication between two DataFrames.

Adding a new row to the DataFrame

new_row = pd.Series([7, 8], index=['A', 'B'])

df = df.append(new_row, ignore_index=True)

Differences

The provided differences in the initial response are concise, but let's elaborate further:

Lambda vs. Regular Functions:

Lambda functions are often used for quick, simple operations.

Regular functions are more versatile and can include complex logic and multiple expressions.

Map vs. Filter:

map() applies a function to every item in an iterable and returns an iterable of the results.

filter() creates an iterable of items from the original iterable that satisfy a condition specified by the function.

Iterators vs. Generators:

Iterators are implemented using classes and can store state between iterations.

Generators are implemented using functions and are memory-efficient due to lazy evaluation.

Modules vs. Packages:

Modules contain Python code in a single file.

Packages are directories containing multiple modules and an __init__.py file. They enable hierarchical organization of code.