Experiment No.: 2

Title: Write a Python program to demonstrate lambda technique

Objectives: 1. Implement the python programs using lambda functions.

Theory:

Python Lambda Functions

Python Lambda Functions are anonymous function means that the function is without a name. As we already know that 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

Simply put, a lambda function is just like any normal python function, except that it has no name when defining it, and it is contained in one line of code.

```
lambda argument(s): expression
```

A lambda function evaluates an expression for a given argument. You give the function a value (argument) and then provide the operation (expression). The keyword lambda must come first. A full colon (:) separates the argument and the expression.

- This function can have any number of arguments but only one expression, which is evaluated and returned.
- One is free to use lambda functions wherever function objects are required.
- You need to keep in your knowledge that lambda functions are syntactically restricted to a single expression.
- It has various uses in particular fields of programming besides other types of expressions in functions.

In the example code below, x is the argument and x+x is the expression.

```
#Normal python function
def a_name(x):
   return x+x#Lambda function
lambda x: x+x
```

1. Scalar values

This is when you execute a lambda function on a single value.

```
(lambda x: x*2)(12)
###Results
24
```

In the code above, the function was created and then immediately executed. This is an example of an immediately invoked function expression

2. Lists

Filter(). This is a Python inbuilt library that returns only those values that fit certain criteria. The syntax is filter(function, iterable). The iterable can be any sequence such as a list, set, or series object (more below).

The example below filters a list for even numbers. Note that the filter function returns a 'Filter object' and you need to encapsulate it with a list to return the values.

```
list_1 = [1,2,3,4,5,6,7,8,9]
filter(lambda x: x%2==0, list_1)
### Results
<filter at 0xf378982348>

list(filter(lambda x: x%2==0, list_1))
###Results
[2, 4, 6, 8]
```

Map(). This is another inbuilt python library with the syntax map(function, iterable).

This returns a modified list where every value in the original list has been changed based on a function. The example below cubes every number in the list.

```
list_1 = [1,2,3,4,5,6,7,8,9]
cubed = map(lambda x: pow(x,3), list_1)
list(cubed)
###Results
[1, 8, 27, 64, 125, 216, 343, 512, 729]
```

3. Series object

A Series object is a column in a data frame, or put another way, a sequence of values with corresponding indices. Lambda functions can be used to manipulate values inside a Pandas dataframe.

Let's create a dummy dataframe about members of a family.

```
import pandas as pddf = pd.DataFrame({
   'Name': ['Luke','Gina','Sam','Emma'],
   'Status': ['Father', 'Mother', 'Son', 'Daughter'],
   'Birthyear': [1976, 1984, 2013, 2016],
})
```

Lambda with Apply() function by Pandas. This function applies an operation to every element of the column.

To get the current age of each member, we subtract their birth year from the current year. In the lambda function below, x refers to a value in the birthyear column, and the expression is 2021(current year) minus the value.

```
df['age'] = df['Birthyear'].apply(lambda x: 2021-x)
```

Lambda with Python's Filter() function. This takes 2 arguments; one is a lambda function with a condition expression, two an iterable which for us is a series object. It returns a list of values that satisfy the condition.

```
list(filter(lambda x: x>18, df['age']))
###Results
[45, 37]
```

Lambda with Map() function by Pandas. Map works very much like apply() in that it modifies values of a column based on the expression.

```
#Double the age of everyonedf['double_age'] =
df['age'].map(lambda x: x*2)
```

We can also perform **conditional operations** that return different values based on certain criteria.

The code below returns 'Male' if the Status value is father or son, and returns 'Female' otherwise. Note that apply and map are interchangeable in this context.

```
#Conditional Lambda statementdf['Gender'] = df['Status'].map(lambda x: 'Male' if x=='father' or
x=='son' else 'Female')
```

4. Lambda on Dataframe object

I mostly use Lambda functions on specific columns (series object) rather than the entire data frame, unless I want to modify the entire data frame with one expression.

For example rounding all values to 1 decimal place, in which case all the columns have to be float or int datatypes because round() can't work on strings.

```
df2.apply(lambda x:round(x,1))##Returns an error if some
##columns are not numeric
```

In the example below, we use apply on a dataframe and select the columns to modify in the Lambda function. Note that we *must* use `axis=1` here so that the expression is applied column-wise.

```
#convert to lower-casedf[['Name','Status']] =
df.apply(lambda x: x[['Name','Status']].str.lower(), axis=1)
```

- Write implementation steps