Lab-03
To Get familiar with Advance concepts in Python

Objectives:

- Python Lambda , python Arrays
- File I/O In Python(Python File Handling,Read Files, Write/Create Files,Delete File
- Python Modules: Built-in Modules In Python,OS,Math,Random,Datetime

Apparatus

- Hardware Requirement
  Personal computer
- Software Requirement
  Anaconda

Theory:

PYTHON LAMBDA AND ARRAYS:

A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.

Syntax

lambda *arguments* : *expression*
The expression is executed and the result is returned:

Example
A lambda function that adds 10 to the number passed in as an argument, and print the result:

```
x = lambda a : a + 10
print(x(5))
```

Lambda functions can take any number of arguments:

Example
A lambda function that multiplies argument a with argument b and print the result:

```
x = lambda a, b : a * b
print(x(5, 6))
```

Example

A lambda function that sums argument a, b, and c and print the result:

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

Note: Python does not have built-in support for Arrays, but Python Lists can be used instead.

Arrays

Note: This page shows you how to use LISTS as ARRAYS, however, to work with arrays in Python you will have to import a library, like the NumPy library.

Arrays are used to store multiple values in one single variable:

Example

Create an array containing car names:

```
cars = ["Ford", "Volvo", "BMW"]
```

Try it Yourself »

What is an Array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
car1 = "Ford"
car2 = "Volvo"
car3 = "BMW"
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Access the Elements of an Array

You refer to an array element by referring to the *index number*.

Example

Get the value of the first array item:

```
x = cars[0]
```

Try it Yourself »

Example

Modify the value of the first array item:

```
cars[0] = "Toyota"
```

Try it Yourself »

The Length of an Array

Use the len() method to return the length of an array (the number of elements in an array).

Example

Return the number of elements in the cars array:

```
x = len(cars)
```

Try it Yourself »

Note: The length of an array is always one more than the highest array index.

Looping Array Elements

You can use the for in loop to loop through all the elements of an array.

Example

Print each item in the cars array:

```
for x in cars:
  print(x)
```

Try it Yourself »

FILE INPUT OUTPUT IN PYTHON:

Python File Open

File handling is an important part of any web application.

Python has several functions for creating, reading, updating, and deleting files.

File Handling

The key function for working with files in Python is the open() function.

The open() function takes two parameters; *filename*, and *mode*.

There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

"t" - Text - Default value. Text mode

"b" - Binary - Binary mode (e.g. images)

Syntax

To open a file for reading it is enough to specify the name of the file:

```
f = open("demofile.txt")
```

The code above is the same as:

```
f = open("demofile.txt", "rt")
```

Because "r" for read, and "t" for text are the default values, you do not need to specify them.

Python File Write

Write to an Existing File

To write to an existing file, you must add a parameter to the open() function:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

Example

Open the file "demofile2.txt" and append content to the file:

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```

Run Example »

Example

Open the file "demofile3.txt" and overwrite the content:

```
f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()

#open and read the file after the appending:
f = open("demofile3.txt", "r")
print(f.read())
```

Run Example »

Note: the "w" method will overwrite the entire file.

Create a New File

To create a new file in Python, use the open() method, with one of the following parameters:

"x" - Create - will create a file, returns an error if the file exist

"a" - Append - will create a file if the specified file does not exist

"w" - Write - will create a file if the specified file does not exist

Example

Create a file called "myfile.txt":

```
f = open("myfile.txt", "x")
```

Result: a new empty file is created!

Example

Create a new file if it does not exist:

```
f = open("myfile.txt", "w")
```

PYTHON MODULES:

What is a Module?

Consider a module to be the same as a code library.

A file containing a set of functions you want to include in your application

Create a Module

To create a module just save the code you want in a file with the file extension .py:

Example

Save this code in a file named mymodule.py

```
def greeting(name):
  print("Hello, " + name)
```

Use a Module

Now we can use the module we just created, by using the import statement:

Example

Import the module named mymodule, and call the greeting function:

```
import mymodule

mymodule.greeting("Jonathan")
```

Run Example »

Variables in Module

The module can contain functions, as already described, but also variables of all types (arrays, dictionaries, objects etc):

Example

Save this code in the file mymodule.py

```
person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

Example

Import the module named mymodule, and access the person1 dictionary:

```
import mymodule

a = mymodule.person1["age"]
print(a)
```

Lab tasks

Exercise 1

Perform the given operations

I. a Python program to square and cube every number in a given list of integers using Lambda.
II. a Python program to find if a given string starts with a given character using Lambda.

III. a Python program to extract year, month, date and time using Lambda.

Exercise 2

I. You have collected information about cities in your province. You decide to store each city's name, population, and mayor in a file. Write a python program to accept the data for a number of
cities from the keyboard and store the data in a file in the order in which they're entered.

II. Write a python program to create a data file student.txt and append the message "Now we are AI students"s

Exercise 3:

Plan on python modules.
(hands on experience)

Notes: these exercises will focus on importing, exploring and using some of the more useful modules available in the Python standard library. Unlike previous exercises there are no questions, the goal is to inform you of modules in the Python standard library that you might find useful in your final projects.

1. In previous exercise sets we used specific modules, for example, random to generate random numbers, math to access mathematical functions and sys for system variables. To access the contents of a module we use the import keyword. For example:

```
>>> import math
```

Modules in the standard library have a top-level help page to give you an overview of the contents of the module (remember, press 'q' to exit the help pages):

```
>>> help(math)
```

Or you can access the help page of a specific variable or function:

```
>>> help(math.sqrt)
```

In Python3 you can also see what is in a module by typing the name of the module, a dot and pressing tab twice.

```
>>> math.
```

2. Instead of importing an entire module, we can import a single function or variable using the from keyword, for example:

```
>>> from math import sqrt
>>> sqrt(9)
3.0
```

This can be useful if we know we are going to use the function alot and want our code to look neater. The downside is that we cannot have a function called, for example, sqrt() in our own code.

3. Sometimes modules change names from one version to another or have long names that we want to avoid typing all the time. Instead of using from we can rename the module using as.
For example, the English do not use the word math, but use the far more pleasing, maths. If I wanted to change that in my code I could do this:

```
>>> import math as maths
```

>>> maths.sqrt(9)
3.0
This only works with modules however, not the functions inside a module (because they are not modules, they are functions):
>>> import math.sqrt as squareroot Traceback (most recent call last):
File "<frozen importlib._bootstrap>", line 2218, in _find_and_load_unlocked
AttributeError: 'module' object has no attribute '__path__' During handling of the above exception, another exception occurred:
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ImportError: No module named 'math.sqrt'; 'math' is not a package
Instead we use a combination of the from keyword and the as keyword to rename functions or variables:
>>> from math import sqrt as squareroot >>> squareroot(9)
3.0

4. time module: if you want to time something, get the current date or pause the execution of your script, then you can use the time module.
   time.time() returns the current time in seconds since the epoch (the epoch is defined as Thursday 1st January 1970 https://en.wikipedia.org/wiki/Unix_time):
   >>> import time
   >>> time.time()
   1427791906.13485
   time.ctime(s) converts a time, s, measured in seconds past the epoch into a human-readable string. If you do not provide an argument, then it returns the current date and time:
   >>> current_time = time.time()
   >>> time.ctime(current_time)
   'Tue Mar 31 11:55:39 2015'
   >>> time.ctime()
   'Tue Mar 31 11:55:51 2015'
   time.sleep(s) waits for at least s seconds and then returns:
   >>> time.sleep(3)
   >>> time.sleep(0.5)
5. glob module: despite the strange name, the glob module is immensely useful for processing large numbers of files. It takes a single string argument that you use to specify a pattern to match files. Patterns are strings that map to file names. For example, in a useless example we can find all files called "file.py" in the current directory:
   >>> import glob
   >>> glob.glob("file.py")
   We never want to get a list of all files with a specific file name (such a list would be zero or one items long, so we may as well just use an if statement). To match all files in the current directory, we can use the asterisks character to mean "match any thing".
   >>> glob.glob("*")

To match a subset of files we can use the asterisks to match parts of file names. For example, to match all .py files:

```
>>> glob.glob("*.py")
```

Play around with this to ensure you understand how the asterisks works.

6. random module: in a previous set of exercises we used random.randint(a,b) to generate random integers between a and b, inclusive:

```
>>> import random
>>> random.randint(1, 10)
```

random.random() generates a real-valued random number in the interval [0,1):

```
>>> random.random()
```

random.shuffle(list) permutes the list into a random ordering in place, i.e. it does not return a permuted copy of the original list:

```
>>> x = [1,2,3,4,5]
>>> random.shuffle(x)
>>> print(x)
```

random.sample(list,k) randomly selects k elements from list (k cannot exceed len(list)) and returns a new list containing those elements:

```
>>> x = [1,2,3,4,5]
>>> random.sample(x, 2)
[5, 1]
```