AI in Software Engineering-460

Semester 4th

Name: _____

Roll Number: _____

Computer Science Department
University Of Karachi

University Of Karachi

Course Name (Course Code) _____ Semester _____ Batch _____ Name of Student: _____Roll No. _____

Lab	**Description & Score**				
1.					
	Software Handling ( )/2	*Programming/ Simulations ( )/5*	*Results ( )/2*	*Lab Report ( )/1*	*Score ( )/10*
2.					
	Software Handling ( )/2	*Programming/ Simulations ( )/5*	*Results ( )/2*	*Lab Report ( )/1*	*Score ( )/10*
3.					
	Software Handling ( )/2	*Programming/ Simulations ( )/5*	*Results ( )/2*	*Lab Report ( )/1*	*Score ( )/10*
4.					
	Software Handling ( )/2	*Programming/ Simulations ( )/5*	*Results ( )/2*	*Lab Report ( )/1*	*Score ( )/10*
5.					
	Software Handling ( )/2	*Programming/ Simulations ( )/5*	*Results ( )/2*	*Lab Report ( )/1*	*Score ( )/10*
6.					
	Software Handling ( )/2	*Programming/ Simulations ( )/5*	*Results ( )/2*	*Lab Report ( )/1*	*Score ( )/10*
7.					
	Software Handling ( )/2	*Programming/ Simulations ( )/5*	*Results ( )/2*	*Lab Report ( )/1*	*Score ( )/10*
8.					
	Software Handling ( )/2	*Programming/ Simulations ( )/5*	*Results ( )/2*	*Lab Report ( )/1*	*Score ( )/10*

University Of Karachi

9.					
	Software Handling ( )/2	*Programming/ Simulations ( )/5*	*Results ( )/2*	*Lab Report ( )/1*	*Score ( )/10*
10.					
	Software Handling ( )/2	*Programming/ Simulations ( )/5*	*Results ( )/2*	*Lab Report ( )/1*	*Score ( )/10*
11.					
	Software Handling ( )/2	*Programming/ Simulations ( )/5*	*Results ( )/2*	*Lab Report ( )/1*	*Score ( )/10*
12.					
	Software Handling ( )/2	*Programming/ Simulations*	*Results ( )/2*	*Lab Report ( )/1*	*Score ( )/10*
		( )/5			
13.					
	Software Handling ( )/2	*Programming/ Simulations ( )/5*	*Results ( )/2*	*Lab Report ( )/1*	*Score ( )/10*
14.					
	Software Handling ( )/2	*Programming/ Simulations ( )/5*	*Results ( )/2*	*Lab Report ( )/1*	*Score ( )/10*
TOTAL SCORE = 140				**OBTAINED SCORE**	

Overall Score: _____ out of 50 ***Examined by: _____***

Overall Formula= (Obtained Score / Total Score) x 50 ***(Name and Signature of Concerned)***

University Of Karachi

Artificial Intelligence-LAB (3+1)

Instructor: **Dr. Humera** Office Hours: ()

Prerequisite: **Data Structures and Algorithms**

Objectives:

This Lab aims to Develop and Understand Artificial Intelligence by using Python.

Contents:

Artificial Intelligence has emerged as one of the most significant and promising areas of computing. This course focuses on the foundations of AI and its basic techniques like Symbolic manipulations, Pattern Matching, Knowledge Representation, Decision Making and appreciating the differences between Knowledge, Data and Code. AI programming language Lisp has been proposed for the practical work of this course.

Learning Outcomes:

Mapping of CLOs and PLOs

Sr. No	Course Learning Outcomes	PLOs	Blooms Taxonomy
CLO_1	Understand the fundamental constructs of Python PLO_1 C2 (Understand) Programming language.		
CLO_2	Understand key concepts in the field of artificial intelligence	PLO_5	C2 (Understand)
CLO_3	Implement artificial intelligence techniques and case studies	PLO_3	C3 (Apply)

Recommended Book:

1. Stuart Russell and Peter Norvig, Artificial Intelligence. A Modern Approach, 3rd edition, Prentice Hall, Inc., 2010. ***Reference Books:***

1. **Artificial Intelligence Engines: A Tutorial Introduction to the Mathematics of Deep Learning By – James V Stone**
2. Hart, P.E., Stork, D.G. and Duda, R.O., 2001. Pattern classification. John Willey & Sons.
3. Luger, G.F. and Stubblefield, W.A., 2009. AI algorithms, data structures, and idioms in Prolog, Lisp, and Java. Pearson Addison-Wesley.

Administrative Instructions:

- According to institute policy, 75% attendance is *mandatory* to appear in the final examination but 100% will be expected. Approved leaves will not be considered towards attendance.
- Every student should bring calculator, book and manual in each lab.
- Every student is expected to be in lab before schedule starting time.
- In any case there will be no rescheduling and makeup of labs.

Lab-01

Introduction to Python Objectives:

The purpose of this lab is to get you familiar with Python and its IDE

Apparatus:

Hardware Requirement Personal computer.

Software Requirement

Anaconda.

Python Installation:

To get started working with Python 3, you'll need to have access to the Python interpreter. There are several common ways to accomplish this:

- Python can be obtained from the **Python Software Foundation** website at python.org. Typically, that involves downloading the appropriate **installer** for your operating system and running it on your machine.
- Some operating systems, notably Linux, provide a **package manager** that can be run to install Python.
- On macOS, the best way to install Python 3 involves installing a package manager called **Homebrew**. You'll see how to do this in the relevant section in the tutorial.
- On mobile operating systems like Android and iOS, you can install apps that provide a Python programming environment. This can be a great way to practice your coding skills on the go.

Alternatively, there are several websites that allow you to access a Python interpreter online without installing anything on your computer at all.

It is highly unlikely that your Windows system shipped with Python already installed. Windows systems typically do not. Fortunately, installing does not involve much more than downloading the Python installer from the python.org website and running it. Let's take a look at how to install Python 3 on Windows:

Step 1: Download the Python 3 Installer

1. Open a browser window and navigate to the Download page for Windows at python.org.
2. Underneath the heading at the top that says **Python Releases for Windows**, click on the link for the **Latest Python 3 Release - Python 3.x.x**. (As of this writing, the latest is Python 3.6.5.)
3. Scroll to the bottom and select either **Windows x86-64 executable installer** for 64-bit or **Windows x86 executable installer** for 32-bit. (See below.)

Sidebar: 32-bit or 64-bit Python?

For Windows, you can choose either the 32-bit or 64-bit installer. Here's what the difference between the two comes down to:

- If your system has a 32-bit processor, then you should choose the 32-bit installer.
- On a 64-bit system, either installer will actually work for most purposes. The 32-bit version will generally use less memory, but the 64-bit version performs better for applications with intensive computation.
- If you're unsure which version to pick, go with the 64-bit version.

Note: Remember that if you get this choice "wrong" and would like to switch to another version of Python, you can just uninstall Python and then re-install it by downloading another installer from python.org.

Step 2: Run the Installer

Once you have chosen and downloaded an installer, simply run it by double-clicking on the downloaded file. A dialog should appear that looks something like this:

Important: You want to be sure to check the box that says **Add Python 3.x to PATH** as shown to ensure that the interpreter will be placed in your execution path.

Then just click **Install Now**. That should be all there is to it. A few minutes later you should have a working Python 3 installation on your system.

PyCharm is available in three editions: Professional, Community, and Educational (Edu). The Community and Edu editions are open-source projects and they are free, but they has less features. PyCharm Edu provides courses and helps you learn programming with Python. The Professional edition is commercial, and provides an outstanding set of tools and features. For details, see the editions comparison matrix.

To install PyCharm

1. Download PyCharm for your operating system.
2. Do the following depending on your operating system: o

Windows installation:

1. Run the PyCharm-*.exe file you've downloaded.
2. Follow the instructions in the installation wizard.

THEORY

Operators

The Python interpreter can be used to evaluate expressions, for example simple arithmetic expressions. If you enter such expressions at the prompt (>>>) they will be evaluated and the result will be returned on the next line.

```
>>> 1 + 1   2
>>> 2 * 3
6
```

Boolean operators also exist in Python to manipulate the primitive True and False values.

```
>>> 1==0
False
>>> not (1==0)
True
>>> (2==2) and (2==3)    False
>>> (2==2) or
(2==3)    True
```

Strings

Like Java, Python has a built in string type. The + operator is overloaded to do string concatenation on string values.

```
>>> 'artificial' + "intelligence"  'artificialintelligence'
```

There are many built-in methods which allow you to manipulate strings.

```
>>> 'artificial'.upper()
```

'ARTIFICIAL' >>>
'HELP'.lower()
'help' >>> len('Help')
4
Notice that we can use either single quotes ' ' or double quotes " " to surround string. This allows for easy nesting of strings.
We can also store expressions into variables.

```
>>> s = 'hello world' >>> print(s)
hello world >>>
s.upper() 'HELLO WORLD'
>>> len(s.upper())
11
>>> num = 8.0
>>> num += 2.5
>>> print(num)
10.5
```

In Python, you do not have declare variables before you assign to them. **Built-in**

Data Structures

Python comes equipped with some useful built-in data structures, broadly similar to Java's collections package. **Lists**

Lists store a sequence of mutable items:

```
>>> fruits = ['apple','orange','pear','banana'] >>> fruits[0] 'apple'
```

We can use the + operator to do list concatenation:

```
>>> otherFruits = ['kiwi','strawberry']
>>> fruits + otherFruits
>>> ['apple', 'orange', 'pear', 'banana', 'kiwi', 'strawberry']
```

Python also allows negative-indexing from the back of the list. For instance, fruits[-1] will access the last element 'banana':

```
>>> fruits[-2] 'pear'
>>> fruits.pop()
'banana'
>>> fruits
['apple', 'orange', 'pear']
>>> fruits.append('grapefruit')
>>> fruits
['apple', 'orange', 'pear', 'grapefruit']
>>> fruits[-1] = 'pineapple'
```

```
>>> fruits
['apple', 'orange', 'pear', 'pineapple']
```

We can also index multiple adjacent elements using the slice operator. For instance, fruits[1:3], returns a list containing the elements at position 1 and 2. In general fruits[start:stop] will get the elements in start, start+1, ..., stop-1. We can also do fruits[start:] which returns all elements starting from the start index. Also fruits[:end] will return all elements before the element at position end:

```
>>> fruits[0:2]
['apple', 'orange']
>>> fruits[:3]
['apple', 'orange', 'pear']
>>> fruits[2:]
['pear', 'pineapple']   >>> len(fruits)    4
```

The items stored in lists can be any Python data type. So for instance we can have lists of lists: >>>

```
lstOfLsts = [['a','b','c'],[1,2,3],['one','two','three']]
>>> lstOfLsts[1][2]   3
>>> lstOfLsts[0].pop()
'c'
>>> lstOfLsts
[['a', 'b'],[1, 2, 3],['one', 'two', 'three']]
```

Tuples

A data structure similar to the list is the *tuple*, which is like a list except that it is immutable once it is created (i.e. you cannot change its content once created). Note that tuples are surrounded with parentheses while lists have square brackets.

```
>>> pair = (3,5)
>>> pair[0]  3
>>> x,y = pair
>>> x  3
>>> y
5
>>> pair[1] = 6
TypeError: object does not support item assignment
```

The attempt to modify an immutable structure raised an exception. Exceptions indicate errors: index out of bounds errors, type errors, and so on will all report exceptions in this way.

Sets

A *set* is another data structure that serves as an unordered list with no duplicate items. Below, we show how to create a set:

```
>>> shapes = ['circle','square','triangle','circle'] >>> setOfShapes = set(shapes)
```

Another way of creating a set is shown below:

```
>>> setOfShapes = {‘circle’, ‘square’, ‘triangle’, ‘circle’}
```

Next, we show how to add things to the set, test if an item is in the set, and perform common set operations (difference, intersection, union):

```
>>> setOfShapes   set(['circle','square','triangle'])
>>> setOfShapes.add('polygon')
>>> setOfShapes
set(['circle','square','triangle','polygon'])
>>> 'circle' in setOfShapes
True
>>> 'rhombus' in setOfShapes
False
>>> favoriteShapes = ['circle','triangle','hexagon']  >>> setOfFavoriteShapes =
set(favoriteShapes) >>> setOfShapes - setOfFavoriteShapes  set(['square','polyon'])
>>> setOfShapes & setOfFavoriteShapes  set(['circle','triangle'])
>>> setOfShapes | setOfFavoriteShapes  set(['circle','square','triangle','polygon','hexagon'])
```

Note that the objects in the set are unordered; you cannot assume that their traversal or print order will be the same across machines!

Dictionaries

The last built-in data structure is the *dictionary* which stores a map from one type of object (the key) to another (the value). The key must be an immutable type (string, number, or tuple). The value can be any Python data type.

Note: In the example below, the printed order of the keys returned by Python could be different than shown below. The reason is that unlike lists which have a fixed ordering, a dictionary is simply a hash table for which there is no fixed ordering of the keys (like HashMaps in Java). The order of the keys depends on how exactly the hashing algorithm maps keys to buckets, and will usually seem arbitrary. Your code should not rely on key ordering, and you should not be surprised if even a small modification to how your code uses a dictionary results in a new key ordering.

```
>>> studentIds = {'knuth': 42.0, 'turing': 56.0, 'nash': 92.0 } >>> studentIds['turing']   56.0
>>> studentIds['nash'] = 'ninety-two'
>>> studentIds
{'knuth': 42.0, 'turing': 56.0, 'nash': 'ninety-two'}
>>> del studentIds['knuth']
>>> studentIds
{'turing': 56.0, 'nash': 'ninety-two'}
>>> studentIds['knuth'] = [42.0,'forty-two']
>>> studentIds
{'knuth': [42.0, 'forty-two'], 'turing': 56.0, 'nash': 'ninety-two'}
>>> studentIds.keys()
['knuth', 'turing', 'nash']
>>> studentIds.values()
```

[[42.0, 'forty-two'], 56.0, 'ninety-two']
>>> studentIds.items()
[('knuth',[42.0, 'forty-two']), ('turing',56.0), ('nash','ninety-two')] >>> len(studentIds) 3

As with nested lists, you can also create dictionaries of dictionaries.

Writing Scripts

Now that you've got a handle on using Python interactively, let's write a simple Python script that demonstrates Python's for loop. Open the file called foreach.py, which should contain the following code:

```
# This is what a comment looks like fruits
= ['apples', 'oranges', 'pears', 'bananas'] for
fruit in fruits: print(fruit + ' for sale')

fruitPrices = {'apples': 2.00, 'oranges': 1.50, 'pears': 1.75} for fruit, price in fruitPrices.items():
if price < 2.00:
print('%s cost %f a pound' % (fruit, price)) else:
print(fruit + ' are too expensive!')
```

At the command line, use the following command in the directory containing foreach.py:

```
[cs188-ta@nova ~/tutorial]$ python foreach.py apples for sale oranges
for sale pears for sale bananas for sale apples are too expensive!
oranges cost 1.500000 a pound pears cost 1.750000 a pound
```

Remember that the print statements listing the costs may be in a different order on your screen than in this tutorial; that's due to the fact that we're looping over dictionary keys, which are unordered. To learn more about control structures (e.g., if and else) in Python, check out the official Python tutorial section on this topic.

If you like functional programming you might also like map and filter:

```
>>> list(map(lambda x: x * x, [1,2,3]))
[1, 4, 9]
>>> list(filter(lambda x: x > 3, [1,2,3,4,5,4,3,2,1])) [4, 5, 4]
```

The next snippet of code demonstrates Python's *list comprehension* construction:

```
nums = [1,2,3,4,5,6] plusOneNums = [x+1 for x in nums] oddNums = [x for x
in nums if x % 2 == 1] print(oddNums) oddNumsPlusOne = [x+1 for x in
nums if x % 2 ==1] print(oddNumsPlusOne)
```

This code is in a file called listcomp.py, which you can run:

```
[cs188-ta@nova ~]$ python listcomp.py
[1,3,5]
[2,4,6]
```

Lab #01 task:

Exercise: Dir and Help

Learn about the methods Python provides for strings. To see what methods Python provides for a datatype, use the dir and help commands:

```
>>> s = 'abc'
```

>>> dir(s)

```
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__',
'__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
'__getslice__', '__gt__', '__hash__', '__init__','__le__', '__len__',
'__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__','__repr__', '__rmod__', '__rmul__', '__setattr__', '__str__',  'capitalize', 'center', 'count', 'decode',
'encode', 'endswith',
'expandtabs', 'find', 'index', 'isalnum', 'isalpha', 'isdigit', 'islower',
'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',
'replace', 'rfind','rindex', 'rjust', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate',  'upper', 'zfill']
```

>>> help(s.find)

Help on built-in function find:

find(...) method of builtins.str instance
S.find(sub[, start[, end]]) -> int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

```
>> s.find('b')
```

1 Try out some of the string functions listed in dir (ignore those with underscores '_' around the method name).

Exercise Python input /output Basic operations

(i)Write a Python program to swap 4 variables values (input four values.

Sample input: Before
swapping
a=2,b=56,c=78,d=9
After Swapping
a=,9,b=78,c=56,d=2

(ii) Write a Python program to convert temperatures to and from celsius,
Fahrenheit.
Formula : c/5 = f-32/9 Expected
Output :
Enter temp in Celsius: 60°C
Temperature in Fahrenheit is :140

Exercise: Lists

(i)Play with some of the list functions. You can find the methods you can call on an object via the dir and get information about them via the help command:

```
>>> dir(list)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',   '__delslice__', '__doc__', '__eq__', '__ge__',
'__getattribute__',
'__getitem__', '__getslice__', '__gt__', '__hash__', '__iadd__', '__imul__',
'__init__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__',
'__rmul__', '__setattr__', '__setitem__', '__setslice__', '__str__',   'append', 'count', 'extend', 'index', 'insert', 'pop',
'remove', 'reverse',  'sort']

>>> help(list.reverse)
Help on built-in function reverse:


reverse(...)
   L.reverse() -- reverse *IN PLACE*

>>> lst = ['a','b','c']
>>> lst.reverse()
>>> ['c','b','a']
```

Note: Ignore functions with underscores "_" around the names; these are private helper methods. Press 'q' to back out of a help screen

(ii)Write a Python program to count the number of strings where the string length is 2 or more and the first and last character are same from a given list of strings. Sample List : ['abc', 'xyz', 'aba', '1221']
Expected Result : 2.

Exercise: Dictionaries

(i)Use dir and help to learn about the functions you can call on dictionaries and implement it.

(ii)Write a Python script to concatenate following dictionaries to create a new one.

Sample Dictionary :
dic1={1:10, 2:20}
dic2={3:30, 4:40}
dic3={5:50,6:60}
Expected Result : {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}

Exercise: List Comprehensions

(i)Write a list comprehension which, from a list, generates a lowercased version of each string that has length greater than five.

(ii)Write a Python program to print a specified list after removing the 0th, 4th and 5th elements
Sample List : ['Red', 'Green', 'White', 'Black', 'Pink', 'Yellow','Teapink'] Expected
Output : ['Green', 'White', 'Black']

Exercise : Operators:

Play with some Operators in Python(assignment ,bitwise ,logical, arithmetic, identity, membership)

(i) What will be the output of the given program

Identity Operators in Python x = 6 if (type(x) is int): print ("true") else: print ("false")	**Output:**
if (type(x) x = 7.2 if (type(x) is not int): print ("true") else: print ("false")	**Output:**

Membership operator: list1=[1,2,3,4,5] list2=[6,7,8,9] for item in list1: if item in list2: print("overlapping") else: print("not overlapping")	**Output:**
Floor division and Exponent and Assign a//=3 a**=5 print(“floor divide=”,a) print(“exponent=”,a)	**Output:**

Bitwise Operaotors: a = 60
/* 60 = 0011 1100 */ b =
13 /* 13 = 0000 1101 */
int c = 0
 	
c = a & b /* 12 = 0000 1100 */
print("Line 1", c)
 	
c = a | b /* 61 = 0011 1101 */
print("Line 2 ", c)
 	
c = a ^ b /* 49 = 0011 0001 */
print("Line 3 ", c)
 	
c = ~a /*-61 = 1100 0011 */
print("Line 4", c)
 	
c = a << 2 /* 240 = 1111 0000 */
printf("Line 5 ", c);
 	
c = a >> 2 /* 15 = 0000 1111 */
printf("Line 6 -", c);
	output

Exercise

Create a Python Program that perform following tasks for any problem of your choice: that must include

Task 1: Introduction
Task 2: Terminal
Task 3: Python Interpreter
Task 4: Variables
Task 5: Opertaors
Task 6: Dictionary usage
Task 7: Lists and Tuples
Task 8: Conditional Statements
Task 9: The For Loop
Task 10: User Input and the While Loop