National University of Computer and Emerging Sciences



Lab Manual 02

CL461-Artificial Intelligence Lab

|  |  |
| --- | --- |
| Course Instructor | Dr. Kashif Zafar |
| Lab Instructor (s) | Saddam Khalil  Ahmad Abdullah |
| Section | D |
| Semester | Spring 2021 |

Department of Computer Science

FAST-NU, Lahore, Pakistan

Table of Contents

[1 Objectives 3](#_Toc67399671)

[2 Task Distribution 3](#_Toc67399672)

[3 Python Lists 3](#_Toc67399673)

[3.1 Indexing & Slicing Examples 4](#_Toc67399674)

[3.2 List Modification Examples 4](#_Toc67399675)

[3.3 List functions 4](#_Toc67399676)

[3.4 List comprehensions 5](#_Toc67399677)

[4 Python Tuples 6](#_Toc67399678)

[4.1 Tuple Functions 6](#_Toc67399679)

[5 Python Dictionaries 6](#_Toc67399680)

[5.1 Dictionary Modification Examples 8](#_Toc67399681)

[5.2 Dictionary Formatting Example 8](#_Toc67399682)

[5.3 Dictionary Functions 8](#_Toc67399683)

[6 Exercise (35 Marks) 9](#_Toc67399684)

[6.1 Verbing (5 Marks) 9](#_Toc67399685)

[6.2 Not Bad (5 Marks) 9](#_Toc67399686)

[6.3 Front Back (10 Marks) 9](#_Toc67399687)

[6.4 Remove Adjacent (5 Marks) 9](#_Toc67399688)

[6.5 Linear Merge (10 Marks) 9](#_Toc67399689)

[7 Submission Instructions 10](#_Toc67399690)

# Objectives

After performing this lab, students shall be able to understand Python data structures which includes:

* Python lists
* Python tuples
* Python dictionaries

# Task Distribution

| **Total Time** | **170 Minutes** |
| --- | --- |
| Python Lists | 20 Minutes |
| Python Tuples | 10 Minutes |
| Python Dictionaries | 20 Minutes |
| Exercise | 110 Minutes |
| Online Submission | 10 Minutes |

# Python Lists

Everything is Python is treated as an object. Lists in Python represent ordered sequences of values. Lists are "mutable", meaning they can be modified "in place". You can access individual list elements with square brackets. Python uses zero-based indexing, so the first element has index 0.

Here are a few examples of how to create lists:

# List of integers

primes = [2, 3, 5, 7]

# We can put other types of things in lists

planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']

# We can even make a list of lists

hands = [

['J', 'Q', 'K'],

['2', '2', '2'],

['6', 'A', 'K'], # (Comma after the last element is optional)

]

# A list can contain a mix of different types of variables:

my\_favourite\_things = [32, 'AI Lab, 100.25]

## Indexing & Slicing Examples

Consider our list of planets created above:

planets[0] # 'Mercury'

planets[1] # 'Venus'

planets[-1] # 'Neptune'

planets[-2] # 'Uranus'

# List Slicing

# first three planets

planets[0:3] # ['Mercury', 'Venus', 'Earth']

planets[:3] # ['Mercury', 'Venus', 'Earth']

# All the planets from index 3 onward

planets[3:] # ['Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']

# All the planets except the first and last

planets[1:-1] # ['Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus']

# The last 3 planets

planets[-3:] # ['Saturn', 'Uranus', 'Neptune']

## List Modification Examples

Working with the same planets list:

# Rename Mars

planets[3] = 'Malacandra'

# ['Mercury', 'Venus', 'Earth', 'Malacandra', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']

# Rename multiple list indexes

planets[:3] = ['Mur', 'Vee', 'Ur']

['Mur', 'Vee', 'Ur', 'Malacandra', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']

## List functions

Python has several useful functions for working with lists.

len(planets) # 8

# The planets sorted in alphabetical order

sorted(planets)

# ['Earth', 'Jupiter', 'Mars', 'Mercury', 'Neptune', 'Saturn', 'Uranus', 'Venus']

primes = [2, 3, 5, 7]

sum(primes) # 17

max(primes) # 7

# Let’s add Pluto to the planets list

planets.append('Pluto')

# Pop removes and returns the last element of the list

planets.pop() # ‘Pluto’

# Remove an item from a list given its index instead of its value

a = [-1, 1, 66.25, 333, 333, 1234.5]

del a[0] # [1, 66.25, 333, 333, 1234.5]

# Remove slices from the list

del a[2:4] # [1, 66.25, 1234.5]

planets.index('Earth') # 2

# Is Earth a planet?

"Earth" in planets # True

# Is Pluto a planet?

"Pluto" in planets # False (We removed it remember)

# Finally to find all the methods associated with Python list object

help(planets)

## List comprehensions

List comprehensions are one of Python's most unique features. List comprehensions combined with functions like min, max, and sum can lead to impressive one-line solutions for problems that would otherwise require several lines of code. The easiest way to understand them is probably to just look at a few examples:

# With list comprehension

squares = [n\*\*2 for n in range(10)] # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

# Without list comprehension

squares = []

for n in range(10):

squares.append(n\*\*2)

# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

# List comprehensions are great of filtering and transformations

short\_planets = [planet for planet in planets if len(planet) < 6]

# ['Venus', 'Earth', 'Mars']

[

planet.upper() + '!'

for planet in planets

if len(planet) < 6

]

# ['VENUS!', 'EARTH!', 'MARS!']

# One line solution

def count\_negatives(nums):

# False + True + True + False + False equals to 2.

# return len([num for num in nums if num < 0])

return sum([num < 0 for num in nums])

count\_negatives([5, -1, -2, 0, 3])

# Python Tuples

Tuples are almost exactly the same as lists. They differ in just two ways.

1. The syntax for creating them uses parentheses instead of square brackets.
2. They cannot be modified (they are immutable).

Tuples are often used for functions that have multiple return values.

t = (1, 2, 3)

t = 1, 2, 3 # equivalent to above

t[0] = 100 # TypeError: 'tuple' object does not support item assignment

# Classic Python Swapping Trick

a = 1

b = 0

a, b = b, a # 0 1

## Tuple Functions

There are only two tuple methods count() and index() that a tuple object can call.

thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)  
x = thistuple.count(5) # 2

thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)

x = thistuple.index(8) # 3

# Python Dictionaries

Dictionaries and lists share the following characteristics:

* Both are mutable.
* Both are dynamic. They can grow and shrink as needed.
* Both can be nested. A list can contain another list. A dictionary can contain another dictionary. A dictionary can also contain a list, and vice versa.

Dictionaries differ from lists primarily in how elements are accessed:

* List elements are accessed by their position in the list, via indexing.
* Dictionary elements are accessed via keys not by numerical index.

Duplicate keys are not allowed. A dictionary key must be of a type that is immutable. E.g. a key cannot be a list or a dict.

Here are a few examples to create dictionaries:

MLB\_team = {

'Colorado' : 'Rockies',

'Boston' : 'Red Sox',

'Minnesota': 'Twins',

'Milwaukee': 'Brewers',

'Seattle' : 'Mariners'

}

# Can also be defined as:

MLB\_team = dict([

('Colorado', 'Rockies'),

('Boston', 'Red Sox'),

('Minnesota', 'Twins'),

('Milwaukee', 'Brewers'),

('Seattle', 'Mariners')

])

# Another way

tel = dict(sape=4139, guido=4127, jack=4098)

# dict comprehensions can be used to create dictionaries from arbitrary key and value expression

{x: x\*\*2 for x in (2, 4, 6)} # {2: 4, 4: 16, 6: 36}

# Building a dictionary incrementally – if you don’t know all the key-value pairs in advance

person = {}

person['fname'] = 'Joe'

person['lname'] = 'Fonebone'

person['age'] = 51

person['spouse'] = 'Edna'

person['children'] = ['Ralph', 'Betty', 'Joey']

person['pets'] = {'dog': 'Fido', 'cat': 'Sox'}

# {'fname': 'Joe', 'lname': 'Fonebone', 'age': 51, 'spouse': 'Edna',

'children': ['Ralph', 'Betty', 'Joey'], 'pets': {'dog': 'Fido', 'cat': 'Sox'}}

## Dictionary Modification Examples

A few examples to access the dictionary elements, add new key value pairs, or update previous value:

# Retrieve a value

MLB\_team['Minnesota'] # 'Twins'

# Add a new entry

MLB\_team['Kansas City'] = 'Royals'

# Update an entry

MLB\_team['Seattle'] = 'Seahawks'

## Dictionary Formatting Example

The % operator works conveniently to substitute values from a dict into a string by name:

hash = {}

hash['word'] = 'garfield'

hash['count'] = 42

s = 'I want %(count)d copies of %(word)s' % hash # %d for int, %s for string

# 'I want 42 copies of garfield'

## Dictionary Functions

The following is an overview of methods that apply to dictionaries:

# Let’s use this dict for to demonstrate dictionary functions

d = {'a': 10, 'b': 20, 'c': 30}

# Clears a dictionary.

d.clear() # {}

# Returns the value for a key if it exists in the dictionary.

print(d.get('b')) # 20

# Removes a key from a dictionary, if it is present, and returns its value.

d.pop('b') # 20

# Returns a list of key-value pairs in a dictionary.

list(d.items()) # [('a', 10), ('b', 20), ('c', 30)]

list(d.items())[1][0] # 'b'

list(d.items())[1][1] # 20

# Returns a list of keys in a dictionary.

list(d.keys()) # ['a', 'b', 'c']

# Returns a list of values in a dictionary.

list(d.values()) # [10, 20, 30]

# Removes the last key-value pair from a dictionary.

d.popitem() # ('c', 30)

# Merges a dictionary with another dictionary or with an iterable of key-value pairs.

d2 = {'b': 200, 'd': 400}

d.update(d2) # {'a': 10, 'b': 200, 'c': 30, 'd': 400}

For more details, visit [iterate dictionary](https://realpython.com/iterate-through-dictionary-python/) & [dictionary comprehensions](https://www.datacamp.com/community/tutorials/python-dictionary-comprehension).

# Exercise (35 Marks)

* For this exercise, use the notebook *CL461\_AI-D\_Lab\_2\_Exercise.ipynb* provided on Google classroom.
* Open the *CL461\_AI-D\_Lab\_2\_Exercise.ipynb*on Google Collab by uploading it.
* The *main()* function is already defined and complete. DO NOT change it.
* Only enter your code within the space provided in the notebook.

## Verbing (5 Marks)

Given a string, if its length is at least 3, add 'ing' to its end. Unless it already ends in 'ing', in which case add 'ly' instead. If the string length is less than 3, leave it unchanged. Return the resulting string.

## Not Bad (5 Marks)

Given a string, find the first appearance of the substring 'not' and 'bad'.

If the 'bad' follows the 'not', replace the whole 'not'...'bad' substring with 'good'. Return the resulting string. So 'This dinner is not that bad!' yields: This dinner is good!

## Front Back (10 Marks)

Consider dividing a string into two halves. If the length is even, the front and back halves are the same length. If the length is odd, we'll say that the extra char goes in the front half. e.g. 'abcde', the front half is 'abc', the back half 'de'. Given 2 strings, a and b, return a string of the form:

a-front + b-front + a-back + b-back

## Remove Adjacent (5 Marks)

Given a list of numbers, return a list where all adjacent == elements have been reduced to a single element, so [1, 2, 2, 3] returns [1, 2, 3]. You may create a new list or modify the passed in list.

## Linear Merge (10 Marks)

Given two lists sorted in increasing order, create and return a merged list of all the elements in sorted order. You may modify the passed in lists. Ideally, the solution should work in "linear" time, making a single pass of both lists.

# Submission Instructions

Always read the submission instructions carefully.

* Rename your Jupyter notebook to your roll number and download the notebook as **.ipynb** extension.
* To download the required file, go to **File->Download .ipynb**
* Only submit the **.ipynb** file. DO NOT **zip** or **rar** your submission file
* Submit this file on Google Classroom under the relevant assignment.
* Late submissions will not be accepted