National University of Computer and Emerging Sciences

Lab Manual 02
Artificial Intelligence Lab

Department of Computer Science
FAST-NU, Lahore, Pakistan

Table of Contents

1 Objectives 3
2 Task Distribution 3
3 Functions 3
3.1 Custom Functions 3

3.1.1 Simple Function Example.......... 3
3.1.2 Function Example with Return Statement.......... 3
3.2 Built-in Functions.......... 4
3.2.1 Built-in Function Examples.......... 4
4 Python Lists.......... 5
4.1 Indexing & Slicing Examples.......... 5
4.2 List Modification Examples.......... 6
4.3 List functions.......... 6
4.4 List comprehensions.......... 7
5 Python Tuples.......... 8
5.1 Tuple Functions.......... 8
6 Python Dictionaries.......... 8
6.1 Dictionary Modification Examples.......... 9
6.2 Dictionary Formatting Example.......... 9
6.3 Dictionary Functions.......... 10
7 Exercise (35 Marks).......... 11
7.1 Check Divisible by 3 (2.5 Marks).......... 11
7.2 Front Back (5 Marks).......... 11
7.3 Transpose (2.5 Marks).......... 11
7.4 Sort tuple (5 Marks).......... 11
7.5 Multiply tuple elements (5 Marks).......... 11
7.6 Factor count (5 Marks).......... 12
7.7 Blocks of stocks (10 Marks).......... 12

1 Objectives

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

- ✔ Python loops

- ✔ Python functions
- ✔ Python conditional statements

2 Functions

1.1 Build-in Functions

Programmers can define their own functions in Python. Functions can contain all types of Python statements like variables, conditions and loops etc.

1.1.1 Simple Function Example

A function in Python starts with `def` keyword followed by the function name with round brackets. Function parameters can be passed depending on the requirement.

```
def hello(name):
     print('Hello {}'.format(name))
hello('Alice') #Hello Alice
hello('Bob') #Hello Bob
```

1.1.2 Function Example with Return Statement

A return statement consists of the following:

- The return keyword.
- The value or expression that the function should return.

```
import random #Syntax to import Python libraries
def getAnswer(answerNumber):
    if answerNumber == 1:
        return 'It is certain'
    elif answerNumber == 2:
        return 'It is decidedly so'
    elif answerNumber == 3:
        return 'Yes'
    elif answerNumber == 4:
        return 'Reply hazy try again'
    elif answerNumber == 5:
        return 'Ask again later'
    elif answerNumber == 6:
        return 'Concentrate and ask again'
    elif answerNumber == 7:
        return 'My reply is no'
    elif answerNumber == 8:
        return 'Outlook not so good'
    elif answerNumber == 9:
        return 'Very doubtful'

r = random.randint(1, 9)
fortune = getAnswer(r)
print(fortune)
```

1.2 Built-in Functions

The Python interpreter has a number of functions built into it that are always available. We have already covered a few built-in functions in the datatypes section above. Refer to this link for the complete list of Python built-in functions.

1.2.1 Built-in Function Examples

Execute the code given below in your Jupyter notebook to find the results of built-in functions.

```
# abs integer number
num = -5
print('Absolute value of -5 is:', abs(num))
# Notice print here, it is also a built-in function

# abs floating number
fnum = -1.45
print('Absolute value of 1.45 is:', abs(fnum))

# input function
x = input('Enter your name:')
print('Hello, ' + x)

# max function
number = [3, 2, 8, 5, 10, 6]
largest_number = max(number);
print("The largest number is:", largest_number)

# print usage
print('Hands-on','python','programming','lab',sep='\n')

# sum function
my_list = [1,3,5,2,4]
print "The sum of my_list is", sum(my_list)
```

2 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]
```

2.1 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']
```

2.2 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']
```

2.3 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, 123,4.5]

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)
```

2.4 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])
```

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
```

3.1 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
```

4 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'}}
```

4.1 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'
```

4.2 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'
```

4.3 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 & dictionary comprehensions.

5 Exercise (35 Marks)

- For this exercise, use the notebook *`CL461_AI-E_Lab_2_Exercise.ipynb`* provided on Google classroom.
- Open the *`CL461_AI-E_Lab_2_Exercise.ipynb`* on Google Collab by uploading it.

5.1 Check Divisible by 3 (2.5 Marks)

Create a function to check if a given element in a list is divisible by 3 or not. Return those elements from the function only. Give a one line solution.

Input:

list1 = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

Output:
```
[0, 9, 36, 81]
```

5.2 Front Back (5 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

5.3 Transpose (2.5 Marks)

Convert the following code snippet to its equivalent list comprehension one liner code. Write a function to return your output.

```
transpose = []
for i in range(3):
  transpose_row = []
  for row in matrix:
      transpose_row.append(row[i])
  transpose.append(transpose_row)
```

Input:
matrix = [[1,2,3],[4,5,6],[7,8,9]]
Output:
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]

5.4 Sort tuple (5 Marks)

Sort a tuple of tuples by 2nd item. Give a single line solution
Input:
tuple1 = (('a', 23), ('b', 37), ('c', 11), ('d',29))
Output:
(('c', 11), ('a', 23), ('d', 29), ('b', 37))

5.5 Multiply tuple elements (5 Marks)

Write a function which returns a tuple that is formed by multiplying adjacent elements in the tuple. Give a one line solution.
Input:
(1, 5, 7, 8, 10)
Output:
(5, 35, 56, 80)

5.6 Factor count (5 Marks)

Write a function which constructs a dictionary where its keys are the integer factors and values are the count of those factors when applied on a list of elements.
Input:
[2, 4, 6, 8]
Output:

{1: 4, 2: 4, 3: 1, 4: 2, 5: 0, 6: 1, 7: 0, 8: 1}

5.7 Blocks of stocks (10 Marks)

A block of stock is defined by a number of attributes, including a purchase date, a purchase price, a number of shares, a stock symbol, and a current price.

Let's assume that we have the following stocks.

Purchase Date	Purchase Price	Shares	Symbol	Current Price
25 Jan 2001	43.50	25	CAT	92.45
25 Jan 2001	42.80	50	DD	51.19
25 Jan 2001	42.10	75	EK	34.87
25 Jan 2001	37.58	100	GM	37.58

Develop a function to represent the given information in the form of tuple.

Develop a function that examines each block, multiplies shares by purchase price and shares by current price to determine the total amount gained or lost.