Advanced Python Programming

Sidheswar Routray
Department of Computer Science & Engineering
School of Technology

Course Objectives

- To develop skills in file handling and exception handling.
- To build applications for automating tasks.
- To build web scraping applications.
- To create applications for data analysis and visualization.

Prerequisite: Knowledge of **Python**

Course Contents

Unit 1. Mastering File Input/Output with Python **10 Hrs.**

I/O operations: Reading and writing files using functions, Different modes for files, reading and writing files in different formats | Working with file objects: Different methods and attributes of file objects| Best practices for file handling:Error handling, buffering, encoding, handling exceptions in file, compressing and decompressing files using libraries; Working with directories and files: create, delete, move, and copy directories and files, and navigating the file system

Unit 2. Automating Tasks with Python **10Hrs.**

Automation with Python: Benefits of automating tasks with Python, Overview of the tools and libraries used for automation, Automating file manipulation tasks, Automated emails and adding attachments, Automating text processing tasks, Advanced automation techniques, Best practices for writing maintainable and scalable automation code

Mid-sem exam syllabus (Tentative)

Unit 3. Web Scraping with Python **9 Hrs.**

Web scraping: Basics of web scraping, Python for web scraping, Different Python libraries for performing web scraping, Web scraping applications, Using bots to extract content and data from websites, Best practices for web scraping.

Unit 4. Data analytics and Visualization with Python **10 Hrs.**

Data analytics and its importance: Overview of Python for data analytics, Handling array operations with Numpy, Data Cleaning and Preparation, Performing statistical analysis using Python libraries, Data visualization

REFERENCE BOOKS

1. "Python Cookbook", By David Beazley and Brian K. Jones, O'Reilly.
2. "Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython" by Wes McKinney, O'Reilly Media.
3. “Automate the Boring Stuff with Python”, by Al Sweigart, No Starch Press.

Course Outcomes

CO 1: Study fundamental Python programming concepts.

CO 2: Understand file handling and exception handling in Python.

CO 3: Compose automated applications.

CO 4: Build web scraping applications and bots.

CO 5: Design Python programs for data analysis and data visualization.

CO 6: Create Python applications for real-world problems.

Examination Scheme

Assessment Method	Assessment Tool	Description	Marks	Mapping with CO	Contribution to CO's
Direct	Mid-sem	MCQ/Analytical/ Output-based/ questions on syllabus covered from Unit I, Unit II	50	CO1/CO2/CO3 CO4/CO5/CO6	It fractionally contributes to 50% weightage of Direct Assessment to CO attainment. (50/2)
Direct	Quiz	MCQ/Output-based/ Theoretical questions on syllabus covered	25	CO1/CO2/CO3 CO4/CO5/CO6	It contributes to 100% weightage of Direct Assessment to CO attainment.
Direct	End-Sem Examination	Unit I, II, III, IV	100	CO1,CO2, CO3,CO4, CO5, CO6	It contributes to 50% weightage of Direct Assessment to CO attainment. (100/2)
	Total 100 Marks				

LAB Work and Evaluation

Assessment Method	Assessment Tool	Description	Marks	Mapping with CO	Contribution to CO's
Direct	Continuous Evaluation	Continuous Project Evaluation in the LABs + Report	50	CO1/CO2/CO3 CO4/CO5/CO6	It fractionally contributes to 100% weightage of Direct Assessment to CO attainment.
Direct	End-Sem Examination	Practical + Viva	50	CO1/CO2/CO3 CO4/CO5/CO6	It contributes to 100% weightage of Direct Assessment to CO attainment.
	Total 100 Marks				

Let’s Recall Basic Python

```
x=(~5)-4
print(x)
```

- In Python, the ~ operator is a bitwise NOT operator. It inverts all the bits of the number. For a given integer n, ~n is equivalent to -(n+1).
- So, let's break down the expression (~5) - 4 step-by-step:
- Calculate ~5:
  - ~5 is -(5 + 1) which equals -6.
- Subtract 4 from the result:
  - -6 - 4 equals -10.
- Thus, the value of x is -10.

```
bool("False")
```

In Python, the bool() function is used to convert a value to a Boolean value (True or False). When using bool() with a string, any non-empty string is considered True, regardless of its content.
So, bool("False") will evaluate to True because "False" is a non-empty string.

```
M = [[1,2,3],[4,5,6],[7,8,9]]
print(M[1])
```

```
M = [[1,2,3],[4,5,6],[7,8,9]]
print(len(M))
```

```
M = [[1,2,3],[4,5,6],[7,8,9]]
N=M
N[2]=20
print(M)
```

```
M = [[1,2,3],[4,5,6],[7,8,9]]
N=M[:]
N[2]=20
print(M)
```

```
y=[20]
y.append([3,4,5])
print(y)
```

```
y=[20]
y.extend([3,4,5])
print(y)
```

Lists

Lists in Python represent ordered sequences of values. A list can be defined as a collection of values or items of different types.

Here is an example of how to create list:

In:	`primes = [2, 3, 5, 7]` `print(primes)`

We can put other types of things in lists:

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

We can even make a list of lists:

In:

```
hands = [
    ['J', 'Q', 'K'],
    ['2', '2', '2'],
    ['6', 'A', 'K'], # (Comma after the last element is optional)
]
#I could also have written this on one line, but it can get hard to
read)
hands = [['J', 'Q', 'K'], ['2', '2', '2'], ['6', 'A', 'K']]
```

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

In:	list1 = ["abc", 34, True, 40, "male"] my_favourite_things = [32, 'raindrops on roses', help] # (Yes, Python's help function is *definitely* one of my favourite things)

Allow Duplicates

Since lists are indexed, lists can have items with the same value

In:	`thislist = ["apple", "banana", "cherry", "apple", "cherry"]` `print(thislist)`
Out:	`['apple', 'banana', 'cherry', 'apple', 'cherry']`

List Length

To determine how many items a list has, use the **len() function**

In:	`thislist = ["apple", "banana", "cherry"]` `print(len(thislist))`
Out:	3

type()

From Python's perspective, lists are defined as objects with the data type 'list'

In:	`mylist = ["apple", "banana", "cherry"]` `print(type(mylist))`
Out:	`<class 'list'>`

It is also possible to use the list() constructor when creating a new list.

In:	`#Using the list() constructor to make a List`
	`thislist = list(("apple", "banana", "cherry")) # note the double round-brackets`
	`print(thislist)`

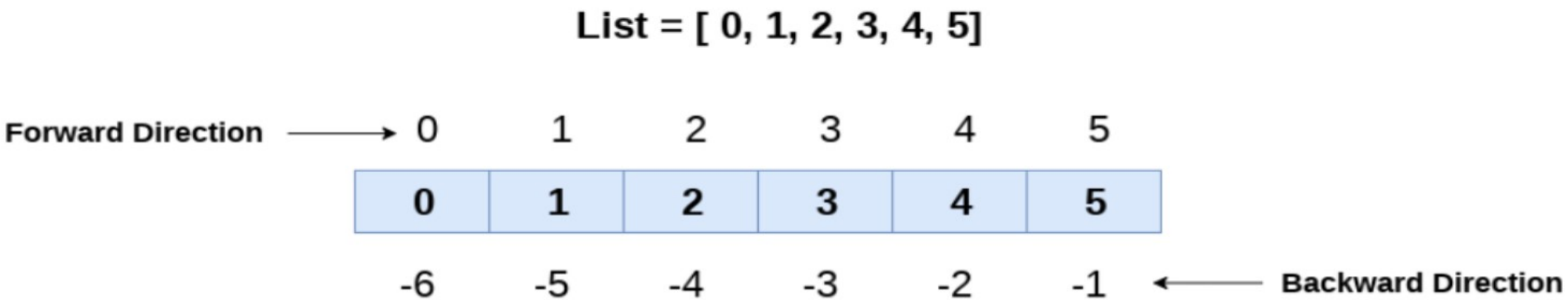
Out:	`['apple', 'banana', 'cherry']`

Indexing

You can access individual list elements with square brackets.

In:	`planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']`
	`planets[0]`
	`# Elements at the end of the list can be accessed with negative numbers, starting from -1:`
	`planets[-1]`
	`planets[-2]`

Out:	`'Mercury'`
	`'Neptune'`
	`'Uranus'`



Slicing

What are the first three planets? We can answer this question using slicing:

In:

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

# planets[0:3] is our way of asking for the elements of planets
starting from index 0 and continuing up to but not including index 3.
planets[0:3]

#If I leave out the end index, it's assumed to be the length of the
list.
planets[3:]

#We can also use negative indices when slicing:
# All the planets except the first and last
planets[1:-1]

# The last 3 planets
planets[-3:]
```

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

In:	list = [1,2,3,4,5] print(list[-1]) print(list[-3:]) print(list[:-1]) print(list[-3:-1])
Out:	5 [3, 4, 5] [1, 2, 3, 4] [3, 4]

Lists are "mutable", meaning they can be modified "in place".

One way to modify a list is to assign to an index or slice expression.

In:	`planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']`
	
	`planets[3] = 'Malacandra'`
	`planets`

Out:	`['Mercury',`
	` 'Venus',`
	` 'Earth',`
	` 'Malacandra',`
	` 'Jupiter',`
	` 'Saturn',`
	` 'Uranus',`
	` 'Neptune']`

List methods

list.append() modifies a list by adding an item to the end:

In:	`planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']`
	`planets.append('Pluto')`
	`planets`
	
	`#Add a list to a list:`
	`a = ["apple", "banana", "cherry"]`
	`b = ["Ford", "BMW", "Volvo"]`
	`a.append(b)`
	`print(a)`

Out:	`['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune','Pluto']`
	
	`['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']`
	
	`['apple', 'banana', 'cherry', ['Ford', 'BMW', 'Volvo']]`

Python List extend() Method

Add the elements of a list (or any iterable), to the end of the current list.

In:	`# Add the elements of cars to the fruits list`
	`fruits = ['apple', 'banana', 'cherry']`
	`cars = ['Ford', 'BMW', 'Volvo']`
	`fruits.extend(cars)`
	`print(fruits)`

Out:	`['apple', 'banana', 'cherry', 'Ford', 'BMW', 'Volvo']`

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

In:	`t = (1, 2, 3)`
	`t = 1, 2, 3 # equivalent to above`
	`t`

Out:	`(1, 2, 3)`

2: They cannot be modified (they are immutable).

In:	`t[0] = 100`

Out:	`Traceback (most recent call last):`
	
	`  File "<ipython-input-3-3382f43ca263>", line 1, in <module>`
	`  t[0]=100`
	
	`TypeError: 'tuple' object does not support item assignment`

In:	`thistuple = ("apple", "banana", "cherry")` `print(thistuple)`
Out:	`('apple', 'banana', 'cherry')`

The tuple() Constructor

It is also possible to use the tuple() constructor to make a tuple.

In:	`thistuple = tuple(("apple", "banana", "cherry"))`
	`# note the double round-brackets`
	`print(thistuple)`

Out:	`('apple', 'banana', 'cherry')`

Access Tuple Items

You can access tuple items by referring to the index number, inside square brackets

In:	`thistuple = ("apple", "banana", "cherry")` `print(thistuple[1])`
Out:	`banana`

Check if Item Exists

To determine if a specified item is present in a tuple use the in keyword.

In:	`# Check if "apple" is present in the tuple`
	`thistuple = ("apple", "banana", "cherry")`
	`if "apple" in thistuple:`
	`  print("Yes, 'apple' is in the fruits tuple")`

Out:	`Yes, 'apple' is in the fruits tuple`

Python - Update Tuples

Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are unchangeable, or immutable as it also is called.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

In:	`# Convert the tuple into a list to be able to change it`
	`x = ("apple", "banana", "cherry")`
	`y = list(x)`
	`y[1] = "kiwi"`
	`x = tuple(y)`
	
	`print(x)`

Out:	`('apple', 'kiwi', 'cherry')`

Add Items

Since tuples are immutable, they do not have a build-in append() method, but there are other ways to add items to a tuple.

1. **Convert into a list:** Just like the workaround for changing a tuple, you can convert it into a list, add your item(s), and convert it back into a tuple.

In:	`# Convert the tuple into a list, add "orange", and convert it back into a tuple`
	`thistuple = ("apple", "banana", "cherry")`
	`y = list(thistuple)`
	`y.append("orange")`
	`thistuple = tuple(y)`
	`print(thistuple)`

Out:	`('apple', 'banana', 'cherry', 'orange')`

Introduction to dictionary and set

Dictionary

- A dictionary is a set of key-value pairs referenced by a single name
- **syntax:**
  dictionaryName = {“keyOne” : “valueOne”, “keyTwo”: “valueTwo”}

```
[ ]  colorOfFruits = {"apple": "red", "mango": "yellow", "orange": "orange"}
     colorOfFruits
```

```
{'apple': 'red', 'mango': 'yellow', 'orange': 'orange'}
```

Introduction to dictionary and set

- Values are retrieved from a dictionary by specifying the key associated to that value.

Syntax: dictionaryName[“key”]

```
[ ]  colorOfFruits["mango"]
```

```
'yellow'
```

A value can be reassigned by making use of the key corresponding to that value.
Syntax: dictionaryName[“key”] = “New Value”

```
[ ]  colorOfFruits["apple"] = "green"
     colorOfFruits
```

```
{'apple': 'green', 'mango': 'yellow', 'orange': 'orange'}
```

Introduction to dictionary and set

- insert new elements in the dictionary

```
colorOfFruits['kiwi']="green"
colorOfFruits
```

```
{'apple': 'green', 'mango': 'yellow', 'orange': 'orange', 'kiwi': 'green'}
```

```
[5] colorOfFruits = {"apple": "red", "mango": "yellow", "orange": "orange"}
    colorOfFruits
```

```
{'apple': 'red', 'mango': 'yellow', 'orange': 'orange'}
```

```
for i in colorOfFruits.keys():
  print(i)
```

```
apple
mango
orange
```

Introduction to dictionary and set

```
[ ]  for i in colorOfFruits.values():
       print(i)
```

```
red
yellow
orange
```

```
[ ]  for i,j in colorOfFruits.items():
       print(i,j)
```

```
apple red
mango yellow
orange orange
```

Introduction to dictionary and set

- **Operations in dictionary:**
  - dictionaryName.keys(): It is used to list all the keys in a dictionary.
  - dictionaryName.values(): It is used to list all the values in a dictionary
  - dictionaryName.items(): It is used to list all keys and values in dictionary

```
[9]  colorOfFruits.keys()
```

```
dict_keys(['apple', 'mango', 'orange'])
```

```
[10]  colorOfFruits.values()
```

```
dict_values(['red', 'yellow', 'orange'])
```

```
colorOfFruits.items()
```

```
dict_items([('apple', 'red'), ('mango', 'yellow'), ('orange', 'orange')])
```

Introduction to dictionary and set

Set

- set is collection of different objects
- It doesn't hold duplicate items
- It stores elements without any order

```
[12] age={25,23,22,26,23,22}
     age
```

```
{22, 23, 25, 26}
```

```
[13] age.add(24)
     age
```

```
{22, 23, 24, 25, 26}
```

```
age.discard(23)
age
```

```
{22, 24, 25, 26}
```

```
x=10
if (x==10):
    print("A")
if (x<=20):
    print("B")
else:
    print("C")
```

```
x=10
if 2:
    print("A")
elif 20:
    print("B")
else:
    print("C")
```

```
x=20
for x in (2,3,5,7):
    x=x+1
print(x)
```

```
if 10:
    print("A")
if -10:
    print("B")
if 0:
    print("C")
else:
    print("D")
```

```
name = "snow storm"
name[5] = 'x'
print (name)
```

```
count=0
for i in range(5):
    for j in range(i):
        count+=1
print(count)
```

```
x="ABCD"
i=0
while(i<len(x)):
      y=x[i]
      i=i+2
print(y)
```

```
my_list = [3,4,9,1,2,6,8]
print(my_list[-6:4]== my_list[1:4])
print(my_list[-5:4]== my_list[2:4])
print(my_list[-1:3]== my_list[-1:4])
```

```
A=[1, 10, 8, 7, 12, 14, 13, 18, 22]
i=0
while i < len(A):
    if A[i]%2==0:
        A.remove(A[i])
    else:
        i=i+1
print(A)
```

```python
a=[i for i in range(20) if(i%2==0)]
print(a)
```