advance python

#### chapter 12 – Advanced Python 1

##### ****Exception Handling in Python****

There are many built-in exceptions that are raised in Python when something goes wrong.

Exceptions in Python can be handled using a try statement. The code that handles the exception is written in except clause.

'''

try:

#code #Code that might throw an exception

except Exception as e:

print(e)

'''

When the exception is handled, the code flow continues without program interruption.

We can also specify the exceptions to catch like below:

'''

try:

#code

except ZeroDivisionError:

#code

except TypeError:

#code

except:

#code (All other exceptions are handled here)

'''

Copy

****Raising Exceptions****

We can raise custom exceptions using the raise keyword in python.

****try with else clause****

Sometimes we want to run a piece of code when try was successful.

'''

try:

#some code

except:

#some code

else:

#Code (This is executed only if the try was successful)

'''

****try with finally****

Python offers a finally clause which ensures execution of a piece of code irrespective of the exception.

'''

try:

#some code

except:

#some code

finally:

#some code (executed regardless of error!)

'''

##### ****if \_\_name\_\_==’\_\_main\_\_’ in Python****

\_\_name\_\_ evaluates to the name of the module in Python from where the program is ran.

If the module is being run directly from the command line, the \_\_name\_\_ is set to string “\_\_main\_\_”.

Thus this behavior is used to check whether the module is run directly or imported to another file.

****The global keyword****

global keyword is used to modify the variable outside of the current scope.

****enumerate function in Python****

The enumerate function adds counter to an iterable and returns it.

for i, item in list1:

print(i, item) #Prints the items of list1 with index!

****List comprehensions****

List comprehensions is an elegant way to create lists based on existing lists.

list1 = [1, 7, 12, 11, 22]

list2 = [i for item in list1 if item>8]

#### Chapter 13 – Advanced Python 2

##### ****Virtual Environment****

An environment that is same as the system interpreter but is isolated from the other python environments on the system.

****Installation****

To use virtual environments, we write

pip install virtualenv          #Installs the package

We create a new environment using:

virtualenv myprojectenv             #Creates a new venv

The next step after creating the virtual environment is to activate it.

We can now use this virtual environment as a separate python installation.

****pip freeze command****

pip freeze returns all the packages installed in a given python environment along with the versions.

“pip freeze > requirements.txt”

The above command creates a file named requirements.txt in the same directory containing the output of pip freeze.

We can distribute this file to other users and they can recreate the same environment using:

pip install –r requirements.txt

****Lambda functions****

Functions created using an expression using the lambda keyword

Syntax:

lambda arguments: expressions (can be used as a normal function)

Example:

Square = lambda x: x\*x

Square(6) #returns 36

Sum = lambda a,b,c: a+b+c

Sum(1,2,3) #returns 6

****bin method(Strings)****

Creates a string from iterable objects

l = [“apple”, “mango”, “banana”]

“,and,”.join(l)

The above line will return “apple, and, mango, and, banana”

****Format method(Strings)****

Formats the values inside the string into the desired output

template.format(p1, p2, …)        #p1, p2 … are the arguments

The syntax for format looks like:

“{} is a good {}”.format(“Harry”,”boy”) – 1

“{1} is a good {0}”.format(“Harry”, “boy”) – 2

Output for 1:

Harry is a good boy

Output for 2:

boy is a good Harry

##### ****Map, Filter & Reduce****

Map applies a function to all the items in an input\_list.

****Syntax:****

map(function, input\_list)             #function can be lambda function

Filter creates a list of items for which the function returns true.

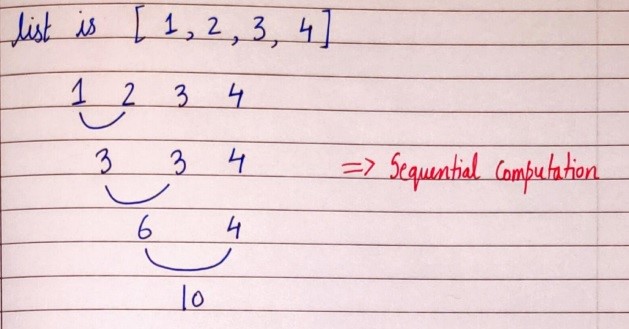
list(filter(function))            #function can be a lambda function

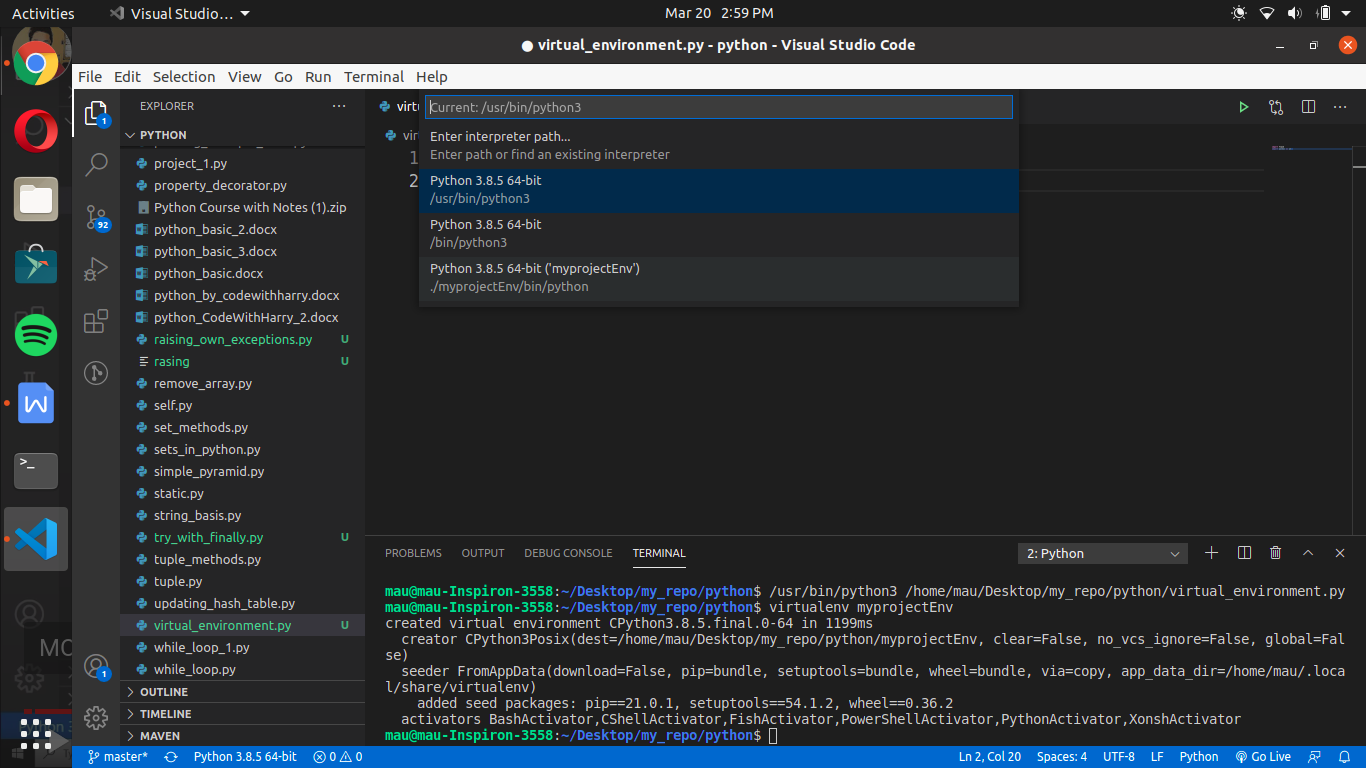
Reduce applies a rolling computation to sequential pair of elements.

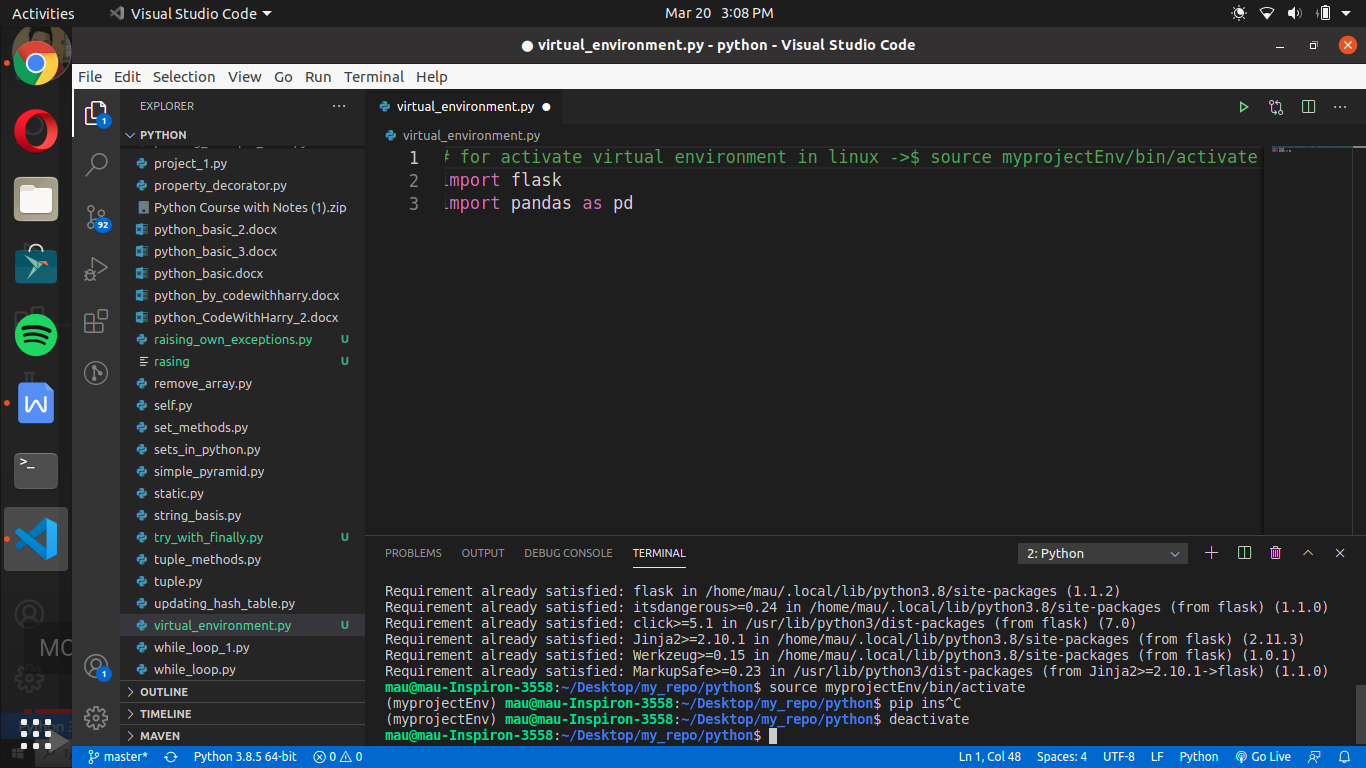
from functools import reduce

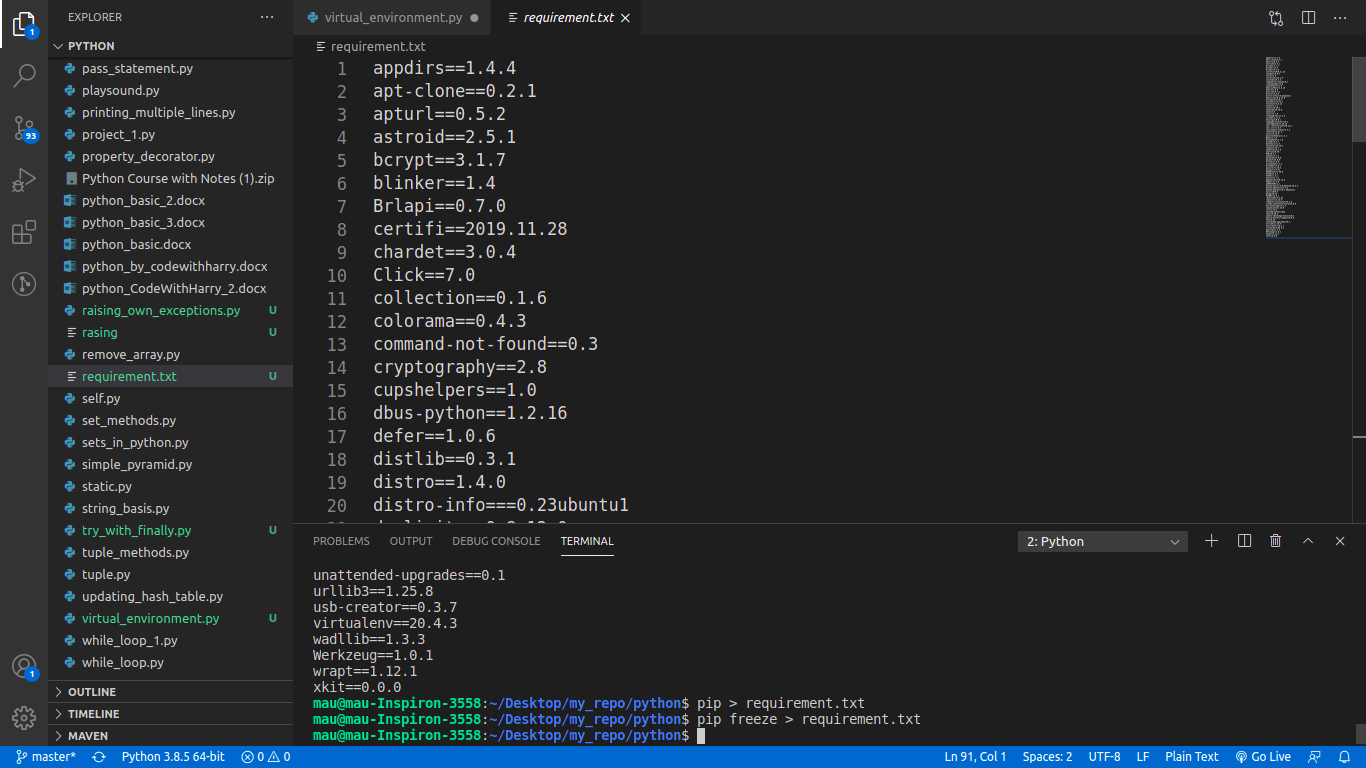
val = reduce(function, list1)        #function can be a lambda function

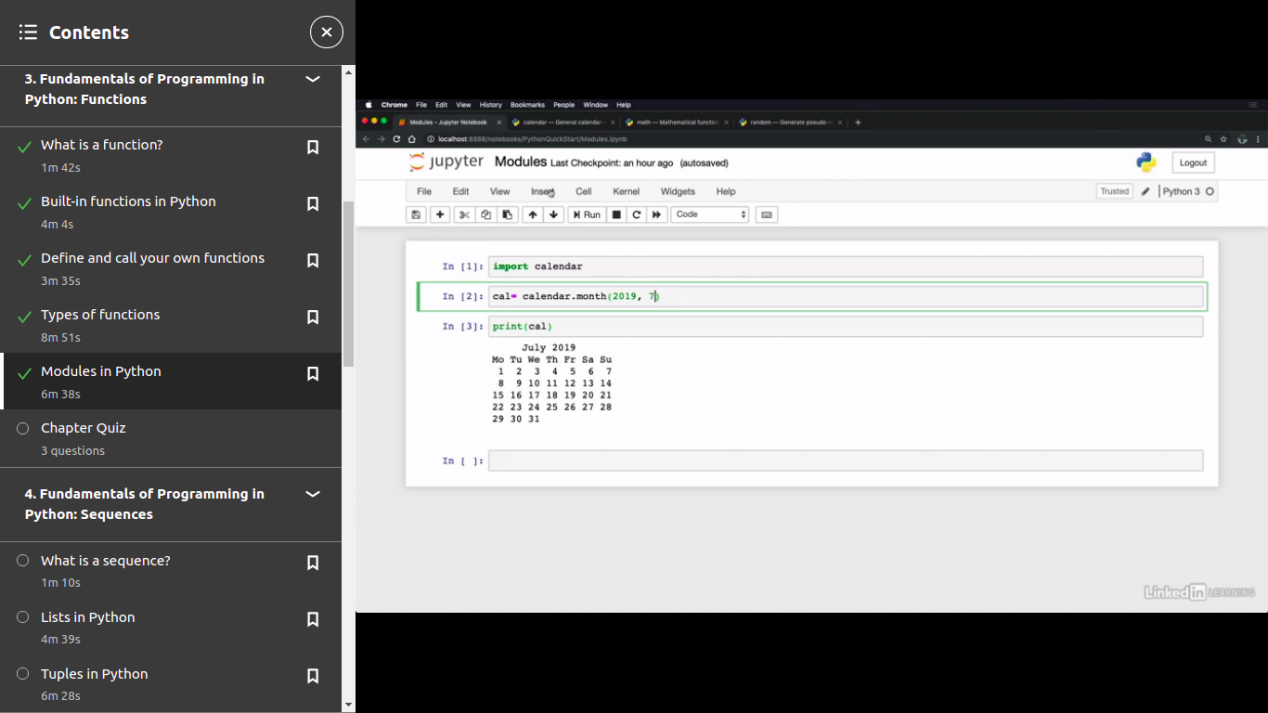
If the function computes sum of two numbers and the list is [1, 2, 3, 4]

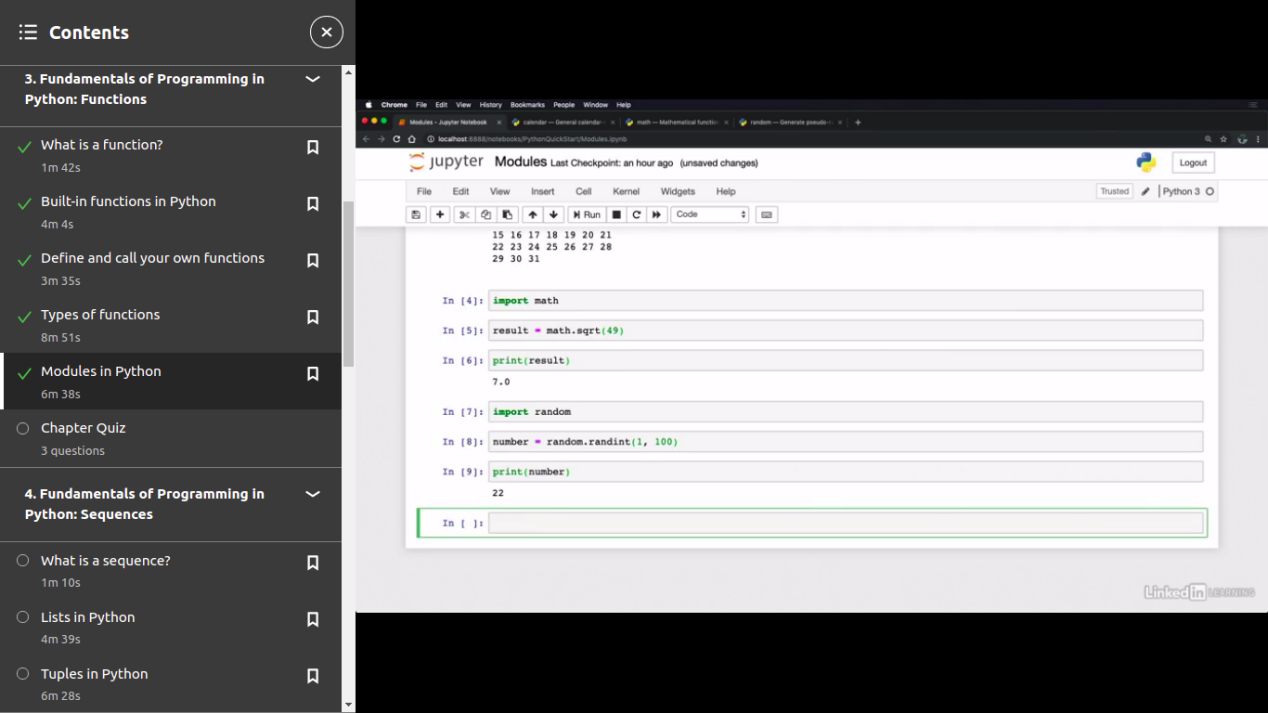


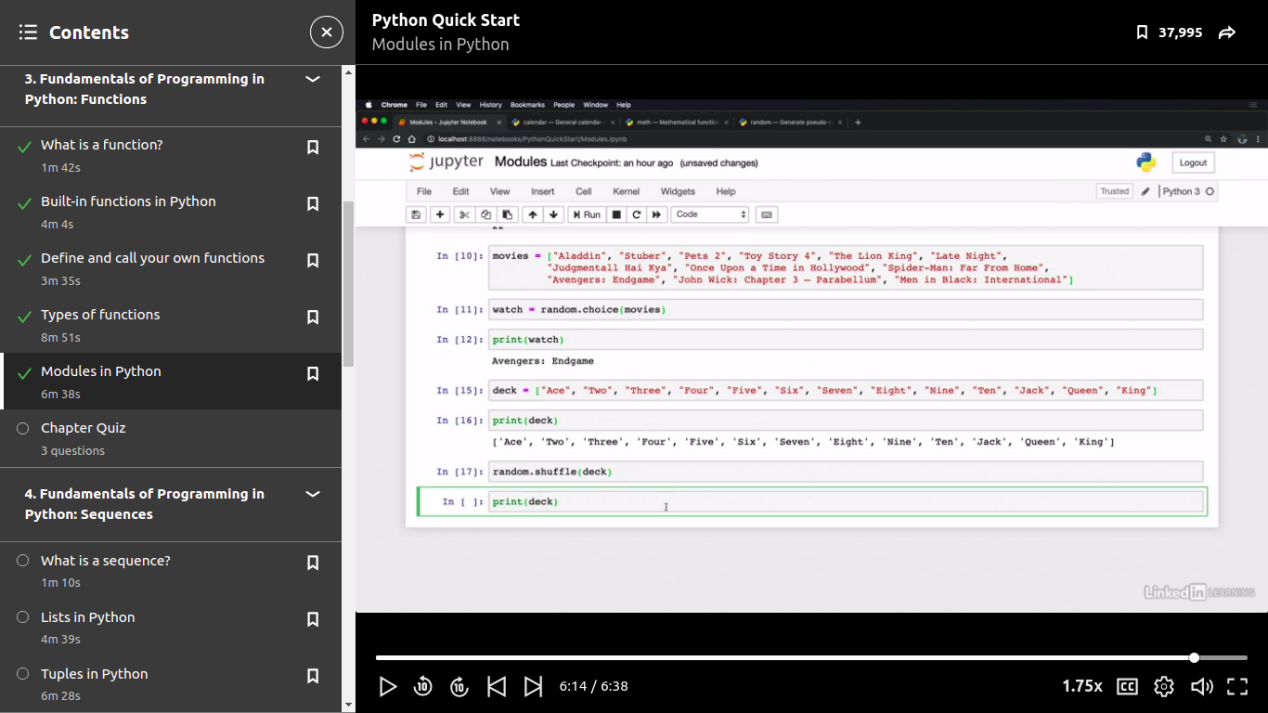


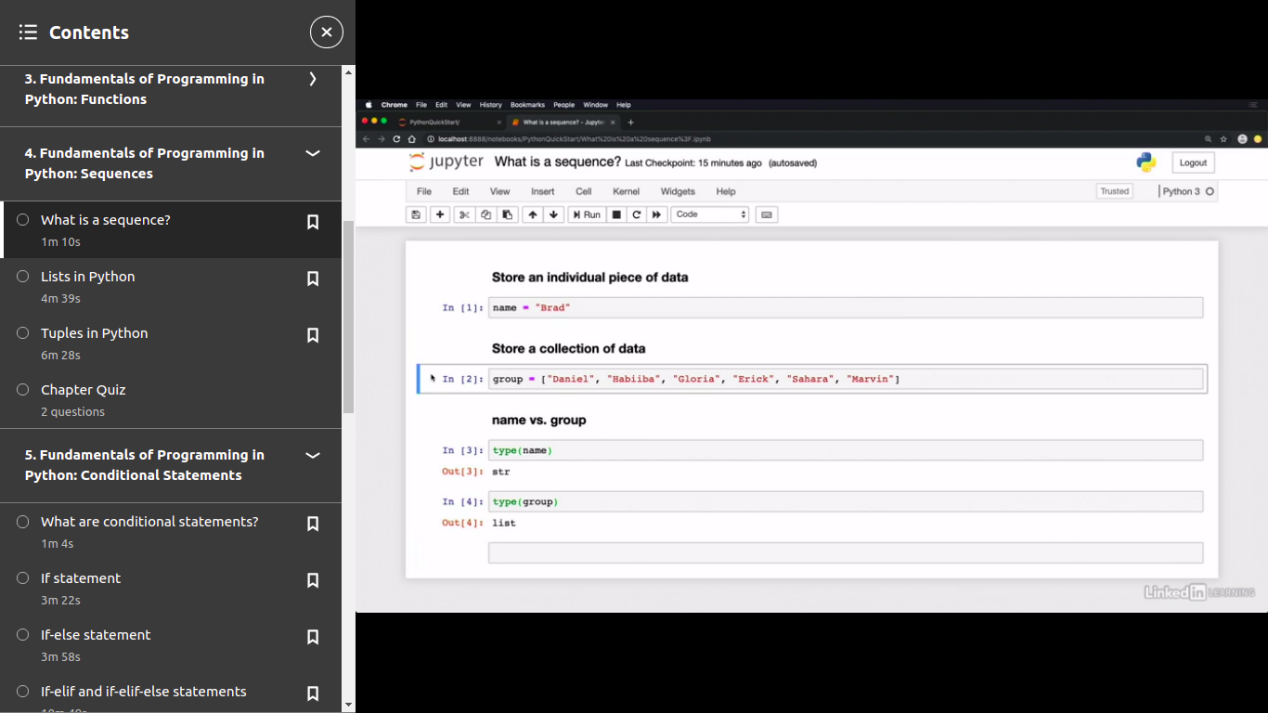


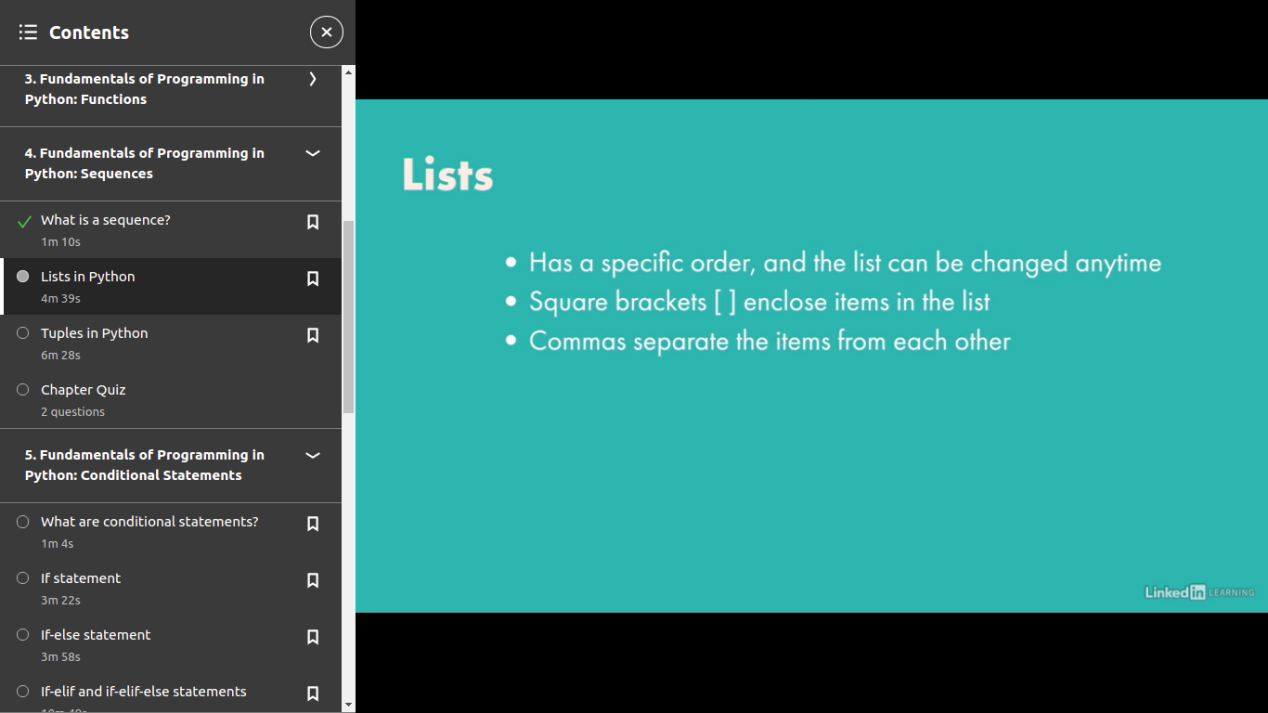












the main difference in List or tuple is list can be modified but tuples cann’t

