Course 4: Python for Data Science, AI & Development

Week 1:

Lesson 1: About the Course

About This Course
- Building blocks for python programming and data collection
- Foundational skills in python programming

Types
- int
    - 11
    - All integers
- float
    - 21.213
    - Includes numbers in between integers
- str
    - “Hello Python 101”
    - Any string
- Typecasting
    - “Type” followed by ()
        - float()
        - int() (this can lose information)
        - str()
        - bool() (1 is True, 0 is False)
        - type() returns type of whatever is inside of it
- boolean
    - True
    - False

Lesson 2: Expressions and Variables

Expressions and Variables
- Mathematical operations (+ - * /)
    - // uses integer division
- Expressions can be stored in variables

Lesson 3: String Operations

String Operations

- Can use single or double quotes
- Use numbers or symbols
- Example uses name=”Michael Jackson ”
- Can access indices of a string (name[0])
- Can also access them backwards (name[-1] returns the last index of the string)
- Slicing: name[0:4] (Mich) or name[8:12] (Jack)
- Stride: name[::2] chooses every 2nd value (McalJcsn)
- Together: name[0:5:2] returns every 2nd value up to index 4 (Mca)
- len(name) = length of string
- Strings can be concatenated
- Can replicate string
    - 3 * “Michael Jackson “ = “Michael Jackson Michael Jackson Michael Jackson “
- Strings are immutable (cannot be mutated)
- Escape sequences (usage of the backslash \)
- \n (new line), \t (tab)
- String Methods
    - Strings are sequences
    - New_name = name.upper() (upper is the method)
        - New_name is now “MICHAEL JACKSON ”
    - name.replace(‘Michael’, ‘Janet’)
        - “Janet Jackson ”
    - name.find(‘el’)
        - Returns first index of the sequence (8)
        - If not found, (-1)

Week 2:

Lesson 1: Lists and Tuples

List and Tuples

- Tuples:
    - Compound data types
    - Ordered sequence
    - Comma separated elements within parenthesis
    - Strings, ints, floats
    - Can be accessed via an index
    - Can concatenate by adding them
    - Can slice them
    - Can use len (returns number of items in tuple)

        - Immutable
            - To manipulate a tuple, we must create a new tuple
        - Can nest tuples
            - Can visualize nested tuples as a tree
- Lists:
    - Also ordered sequences
    - Represented by square brackets
    - Like tuples, but lists are mutable
    - Can concatenate them
    - Can slice them
    - Methods
        - extend (adds all elements)
        - append (adds one element with all elements in it as a list)
        - del
        - split (can use a delimiter by passing it into split(‘,’) like that)
    - Cloning
        - B = [‘A’]
        - C = B
        - This will give C and B the same reference
        - To clone we instead use
        - C = B[:]

Dictionaries

- Type of collection
- Keys and Values
- Keys = index
- Values = information
- Uses {}
- Keys must be immutable & unique
- Values can be immutable, mutable, and duplicates
- Each key and value pair is separated by a comma
- Methods
    - Del
    - In
    - keys()
    - values()

Sets

- Type of collection
    - You can input different python types
- Unordered
- Only unique elements
- Uses {}
- When set is created even with duplicates, duplicates are removed

- List to set -> set(list) (typecasting)
- Set operations
    - Think of a venn diagram
    - .add()
    - .remove()
    - “example” in Set (returns True or False)
- Mathematical set operations
    - & (intersection of two sets)
    - .union() (all of both sets)
    - .issubset() (check if one set is subset of another)
    - .issubset() (check if one set is superset of another)
    - .difference() (set 1 without elements from set 2)
    - .intersection() (same as using &)

Week 3:

Lesson 1: Conditions and Branching

Conditions and Branching

- Conditions (comparison operators)
    - == checks if two values are equal
    - >, >=, <, <=, !=
- Branching
    - If / else statements
    - elif (else if) statement
- Logic Operators
    - not()
    - or
    - And

Lesson 2: Loops

Loops

- range(i, j)
    - Order sequence (iterates up to but not including second number)
- for loop
    - for … in …
    - for i in range(x, y)
    - for square in squares

        - for i,square in enumerate(squares)
            - Enumerate runs for the length of the list
- while loop
    - Runs **while** a condition is met
    - while(...)

Lesson 3: Functions

Functions

- Piece of code you can reuse
- Uses “def”
    - def f1(input):
- Built-in functions
    - len() (takes in type sequence, returns length of sequence)
    - sum()
    - sorted() (does not change original list, creates a new one)
    - sort() (changes original list, does not create a new one)
- Making functions
    - def add1(a)
        - b=a+1
        - return b
- “”” (triple quotes) used for documentation
- pass (keyword to do nothing)
- Using * creates variatic parameters
- Scope
    - Part of the program where variables are accessible
    - Global, local
    - Can have same variables without conflict
    - If variable not assigned in function, python will check global scope
- Can define local variables as global by using “global” then variable name

Lesson 4: Exception Handling

Exception Handling

- try…except
- Once an error is met within try statement, jumps to except statement
    - Error message should print when error occurs
    - Can create custom responses for different errors using
        - except ValueError (for example)
        - except ZeroDivisionError
- else statement

        - If execution runs properly
    - finally
        - Code to run no matter what (for instance, closing a file)
    -

Lesson 5: Objects and Classes

Objects and Classes

- Types (these are all objects)
    - Int
    - Float
    - String
    - List
    - Dictionary
    - Bool
- Every object has
    - Type
    - Internal data representation
    - Set of procedures for interacting with object (methods)
- Find type by using type()
- Methods are functions that every instance of that class or type provides
    - How you interact with data of the object
- Create your own type
- Class
    - Data attributes
    - Methods
- class Circle(object): (format of creating classes. Defines class)
    - def __init__(self, radius, color): (data attributes to initialize each instance of the class)
        - __init__ is a constructor
        - self.radius = radius;
        - self.color = color;
- RedCircle = Circle(10,'red')
- Method
    - def add_radius(self, r):
        - self.radius = self.radius + r
- dir()
    - Returns list of data attributes of an object

Week 4:

Lesson 1: Reading & Writing Files with Open

Reading Files with Open

- .read()
- open(directory, mode)
    - r = reading
    - w = writing
    - a = appending
- .name()
    - String with name of file
- .close()
    - Closes file
- with open() as File1:
    - Automatically closes file object
- .readlines()
    - Output every line as an element in a list
- .readline()
    - Reads first line of file
    - Putting in characters will limit the reading to the first line
- readlines(number)
    - Gets first 4 characters from the file

Writing Files with Open

- .write()
    - If open mode is w, will overwrite file if file already exists
    - Using mode a will write to the same file and not make a new one

Lesson 2: Pandas

Loading Data with Pandas

- import pandas as pd
- df = pd.read_csv(filepath)
    - Reads csv that is given
    - Usually store csv in variable named df (dataframe)
- df.head()
    - Examine first 5 rows of dataframe
- .read_excel
- Can create a dataframe from a dictionary
    - Keys = table headers

        - Values = lists corresponding to the rows

Pandas: Working with and Saving Data

- .unique()
    - All unique elements in a column
- Can use inequality operators (<=, >=, etc)
- df[[‘Length’]]
    - New dataframe with column specified
- df[‘Length’]
    - 1-D dataframe with column specified
- df.iloc[0, 0]
    - Access unique elements in a specified row and column
- df.loc[0, ‘Length’]
    - Uses column name to access elements
- df.iloc[0:2, 0:3]
    - Slicing when selecting elements
- df.loc[0:2, ‘Artist’:’Released’]
    - Slicing by name
- df.index
    - Access index of dataframe

Lesson 3: Numpy in Python

One Dimensional Numpy

- Basics and array creation
    - List holds ata
    - Numpy array is similar
    - Fixed size and one type
    - a = np.array([0,1,2,3,4])
    - a.type()
        - numpy.ndarray
    - a.size()
    - a.ndim()
        - Number of array dimensions
    - a.shape()
        - Tuple
        - Indicates size of array in each direction
- Operations
    - Faster and require less memory than Python
    - Vector addition and subtraction
        - Python way
            - u = [1,0]
            - v = [0,1]
            - z = [0,0]

            - for n, m in zip(u,v):
                - z.append(n+m)
        - Numpy way
            - u = np.array([1,0])
            - v = np.array([0,1])
            - z = u+v
            - z : array([1,1])
        - Easier to type and runs much faster
    - Array Multiplication with a Scalar
        - Each component of array is affected by scalar
        - y = np.array([1,2])
        - z = 2*y
        - z : array([2,4])
    - Product of two numpy arrays
        - u = np.array([1,2])
        - v = np.array([3,2])
        - z = u*v
        - z : array([3,4])
    - Dot Product
        - u = np.array([1,2])
        - v = np.array([3,1])
        - result = np.dot(u,v)
        - result = 5
    - Adding Constant to a Numpy Array (broadcasting)
        - u = np.array([1,2, 3, -1])
        - z = u+1
        - Every element will be 1 larger
    - Universal Functions
        - a.mean()
        - b.max()
        - x = np.pi
        - y = np.sin(x)
        - np.linspace(-2,2,num=5)
            - Returns 5 evenly spaced numbers over specified interval

Two Dimensional Numpy

- Think of them as rectangular arrays
- ndim = 2
- shape = (3,3) (as an example)
    - First number says how many list
    - Second number says how many elements in each list
- Operations are identical to 1D arrays
- Matrix Multiplication
    - Columns in A must be = to rows in B

- 1st column A and 1st row B are multiplied and added together
- Uses np.dot() like before

Week 5:

Lesson 1: Simple APIs

Simple APIs
- Application Program Interfaces
- Lets two pieces of software talk to each other (middle guy)
- Pandas is an API to use my python program with the order code
- REST APIs
    - Communicate through the internet
    - REpresentational State Transfer
    - Rules regarding
        - Communication
        - Input or Request
        - Output or Response
    - Me = Client
    - Web service = resource
    - Client finds service via an endpoint from the service
    - Request through an http message
- Pandas
    - pd.to_datetime(scrambled data to be converted to a readable format)
- API Keys and Endpoints
    - Gives acces to API
    - Unique characters to authorize you to use API

Lesson 2: REST APIs, Webscraping, and Working with Files

REST APIs & HTTP Requests
- HTTP Protocol
    - Transfers information through the web
    - JSON file
    - URL - Uniform Resource Locator
    - Scheme
        - http://
    - Internet Address (used to find the location)
        - www.ibm.com
    - Route (location on web server)

        - /images/IDSNlogo.png
    - All together: http://www.ibm.com/images/IDSNlogo.png
    - Methods
        - GET
            - Retrieves data from server
        - POST
            - Submits data to the server
        - PUT
            - Updates data already on server
        - DELETE
            - Deletes data from server
- Using Requests Library
    - import requests
    - r = requests.get(url)
    - r.status_code -> 200
    - r.request.headers
    - r.request.body
    - r.headers
        - [‘date’]
        - [‘Content-Type’]
    - r.encoding -> ‘UTF-8’

Create Query string

```
url_get='http://httpbin.org/get'

payload={"name":"Joseph","ID":"123"}

r=requests.get(url_get,params=payload)

r.url:'http://httpbin.org/get?name=Joseph&ID=123'

r.request.body : None

r.status_code: 200
```

    - r.text
    - r.json

HTML for Webscraping

- Python to extract information from websites

Webscraping

- Automatically extract info from a website
- Tools
    - Python
    - Requests
    - Beautiful Soup
        - Represents html as tree-like objects

Working with DIfferent File Formats

- Reading CSV
    - import pandas as pd
    - file = “file.csv”
    - df = pd.read_csv(file)
    - First line is added as header but we don’t want that
    - df.columns = [‘Name’, ‘Phone’] (fixes the above issue)
- Reading JSON
    - import json
    - with open(‘filesample.json’, ‘r’) as openfile:
        - json_object = json.load(openfile)
    - print(json_object)
- Reading XML
    - import pandas as pd
    - import xml.etree.ElementTree as etree
    - tree = etree.parse(“fileExample.xml”)
    - root = tree.getroot()
    - columns = [“Name”, “Phone Number”, “Birthday”]
    - df = pd.DataFrame(columns = columns)
    - THEN
    - for node in root:
        - name = node.find(“name”).text
        - phonenumber = node.find(“phonenumber”).text
        - birthday = node.find(“birthday”).text
    - df = df.append(pd.Series([name, phonenumber, birthday], index = columns).., ignore_index = True)