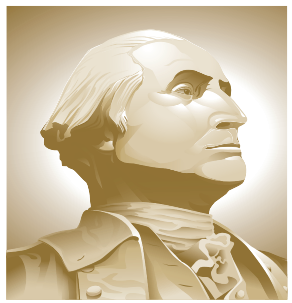
Introduction to Natural Language Processing

Natural Language Processing
Lecture 1-b



THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

Python

Why Python?

- Python is heavily used in NLP area for the following reasons:

- Easy to learn
- Fast Enough
- Object oriented
- Widely used
- Fairly portable
- NLP Packages
- Deep Learning Framework Wrappers

Objects and types

- We use the term object to refer to any entity in a python program.
- Every object has an associated type, which determines the properties of the object.
- Python defines six main types of built-in objects:
  - Number 10 or 2.71828
  - String ”NLP”
  - List [1,2,3] or [’A’, ’b’]
  - Tuple (1,2) or (’AB’, ’C’)
  - Dictionary {’Name’: ’Amir’, ’Age’: 99
  - File

Strings

- A string type object is a sequence of characters

```
1  s = "foo"
2  print(s)
3  s = 'Foo'
4  print(s)
5  # s = "foo'
6  String= "ABCD"
7  print(String[0])
8  print(String[1])
9  print(String[-1])
10 print(String[-2])
```

- Each string is stored in computer array of characters.
- A string type object is a sequence of characters
- You can access individual characters by using indices in square brackets

- Accessing substrings by indexing it.

```
1 S = '0123456'
2 print(S[1:3])
3 print(S[:3])
4 print(S[2:])
5 print(S[2:3])
6 print(S[:])
7 print(S[::-1])
```

- Special characters.

```
1 print('The backslash is used for special char \\ \' \"')
2 print("This is for a new line \n and the we can \t tab")
```

More String Functionality

- String Methods.

```
1  S1 = 'Asegmentofastringiscalledaslice.'
2  print(len(S1))
3  print ("*" * 10)
4  print('ab' in 'absent')
5  S2 = S1[10:15]
6  print(S2)
7  print(S1.find('i'))
8  print(S1.find('slice'))
9  print(S1.lower)
10 print(S1.upper())
11 print(S1.replace('slice','****'))
12 print(S1.startswith('a'))
```

- Strings can not be modified.

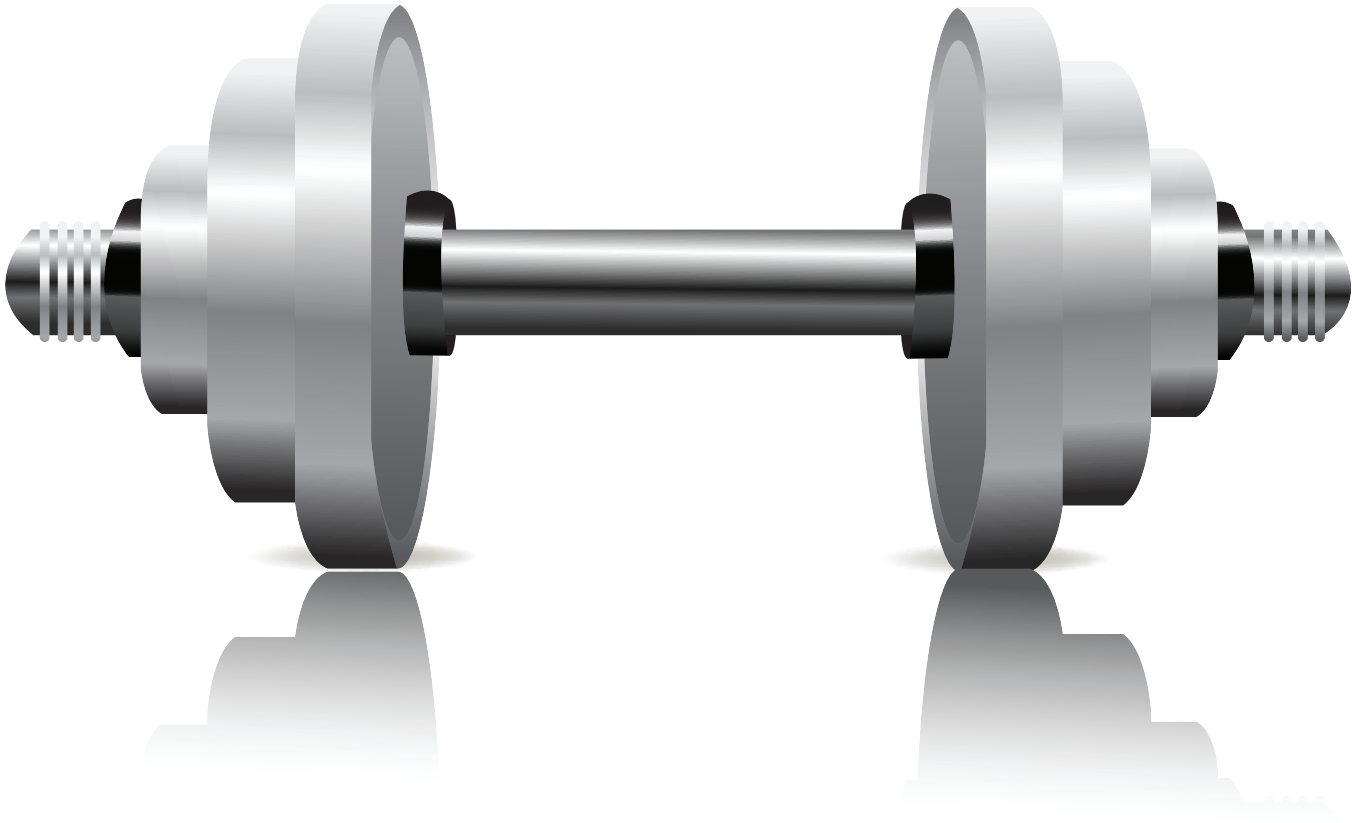
```
1  # S1[0] = 'A'
2  S3 = 'a' + S1[1:]
3  print(S3)
```

- More methods and its functionality.

```
1  S1= '1'
2  S2 = '1abc'
3  S3 = 'Acd'
4  S4 = 'abcd. Abcd.'
5  S5 = '     abcd'
6  S6 = ' '
7  print(S4.capitalize())
8  print(S4.count('c'))
9  print(S1.isdigit())
10 print(S2.isdigit())
11 print(S4.isalnum())
12 print(S3.encode('UTF-8'))
13 print(S3.encode('UTF-16'))
14 print(S1.center(4))
15 print(S5.strip())
16 print(S4.index('c'))
17 print(S6.isspace())
18 print(S3.istitle())
19 print('.'.join(S3))
20 print(S4.split(sep='.'))
```

Exercise - Lecture 1

- Exercise 1 to 3
- Class-Ex-Lecture1.py



Python I/O (File, Itertool, Shutil)

Files

- As far as Python is concerned, a file is just a string.
- It is stored on your file system, that you can read or write, gradually or all together.
- Directory is an old name for a folder.
- In Python, there is no need for importing external library to read and write files.
- Python provides an in built function for creating, writing, and reading files.

Open a Text File

- To open a file, you need to use the built-in open function.

```
1 f = open("sample_text.txt","w+")
2 for i in range (20):
3     f.write('This is a line {} \n'.format(i+1))
4 f.close()
```

- Append to a File in Python.

```
1 f = open("sample_text.txt", 'a+')
2 for i in range (2):
3     f.write('Append a line {} \n'.format(i+1))
4 f.close()
```

File Modes

- The following table shows the various File Modes in Python:

Mode	Description
'r'	This is the default mode. It Opens file for reading.
'w'	This Mode Opens file for writing. If file does not exist, it creates a new file. If file exists it truncates the file.
'x'	Creates a new file. If file already exists, the operation fails.
'a'	Open file in append mode. If file does not exist, it creates a new file.
't'	This is the default mode. It opens in text mode.
'b'	This opens in binary mode.
'+'	This will open a file for reading and writing (updating)

Writing and Reading to Files in Python

- In order to write into a file in Python, we need to open it in write **w**, append **a** or exclusive creation **x** mode.

```
1 with open("sample_text1.txt",'w',encoding = 'utf-8') as f:
2     f.write("Thi is my first file created. \n")
3     f.write("This is the second line file\n\n")
4     f.write("Last but not least\n")
5     f.close()
```

- To read a file in Python, we must open the file in reading **r** mode.

```
1 f = open("sample_text1.txt",'r',encoding = 'utf-8')
2 f.read(4)     # read the first 4 data
3 f.read(4)     # read the next 4 data
4 f.read()      # read in the rest till end of file
5 f.read()      # further reading returns empty sting
6 f.tell()      # get the current file position
7 f.seek(0)     # bring file cursor to initial position
8 print(f.read())
9 f.close()
```

More on Reading to Files in Python

- In this program, we used for loop, readfile() and readlines() methods.

```
1  f = open("sample_text1.txt",'r',encoding = 'utf-8')
2  for line in f:
3     print(line)
4  f.seek(0)
5  print(f.readline())
6  print(f.readline())
7  print(f.readline())
8  print(f.readline())
9  f.seek(0)
10 print(f.readlines())
11 f.close()
```

- Lastly, the readlines() method returns a list of remaining lines of the entire file.

Python File Methods

Method	Description
close()	Closes an opened file. It has no effect if the file is already closed.
detach()	Separates the underlying binary buffer from the TextIOBase and returns it.
fileno()	Returns an integer number (file descriptor) of the file.
flush()	Flushes the write buffer of the file stream.
isatty()	Returns True if the file stream is interactive.
read(n)	Reads at most n characters from the file. Reads till end of file if it is negative or None.
readable()	Returns True if the file stream can be read from.
readline(n=-1)	Reads and returns one line from the file. Reads in at most n bytes if specified.

Method	Description
readlines(n=-1)	Reads and returns a list of lines from the file. Reads in at most n bytes/characters if specified.
seek(offset,from=SEEK_SET)	Changes the file position to offset bytes, in reference to from (start, current, end).
seekable()	Returns True if the file stream supports random access.
tell()	Returns the current file location.
truncate(size=None)	Resizes the file stream to size bytes. If size is not specified, resizes to current location.
writable()	Returns True if the file stream can be written to.
write(s)	Writes the string s to the file and returns the number of characters written.
writelines(lines)	Writes a list of lines to the file.

- **Accumulate**: This function makes an iterator that returns the results of a function.

```
1  import itertools
2  import operator
3  data = [1, 2, 3, 4, 5]
4  states = ['Newyork', 'Virginia', 'DC', 'Texas']
5  [print(each) for each in states]
6  result = itertools.accumulate(data, operator.mul)
7  for each in result:
8      print(each)
9  print(operator.mul(1,9))
10 print(operator.pow(2,4))
11 print(help(operator))
```

- **Combinations:** This function takes an iterable and a integer. This will create all the unique combination that have **r** members.

```
1  result = itertools.combinations(states, 2)
2  for each in result:
3      print(each)
4  for i in itertools.count(10,3):
5      print(i)
6      if i > 20:
7          break
```

- **Count**: Makes an iterator that returns evenly spaced values starting with number start.

```
1 import itertools
2 for i in itertools.count(1,2):
3     print(i)
4     if i > 20:
5         break
```

- **Cycle:** This function cycles through an iterator endlessly.

```
1 states = ['Newyork', 'Virginia', 'DC', 'Texas']
2 for index, city in enumerate(itertools.cycle(states)):
3     print(city)
4     if index==10:
5         break
```

- **Chain:** This function takes a series of iterables and return them as one long iterable

```
1 S1 = ['A', 'B', 'C', 'D', 'E']
2 S2 = ['F', 'G', 'H', 'I']
3 result = itertools.chain(S1, S2)
4 for each in result:
5     print(each)
```

- **Compress**: This function filters one iterable with another.

```
1 import itertools
2
3 S1 = ['A', 'B', 'C', 'D']
4 selections = [True, False, True, False]
5 result = itertools.compress(S1, selections)
6 for each in result:
7     print(each)
```

- **Islice:** This function is very much like slices. This function allows you to cut out a piece of an iterable.

```
1 S2 = itertools.islice(S1, 2)
2 for each in S2:
3     print(each)
```

- **Permutations:** This function permutes the elements.

```
1 S3 = itertools.permutations(S1)
2 for each in S3:
3     print(each)
```

- **Repeat**: This function will repeat an object over and over again. Unless, there is a times argument.

```
1 import itertools
2 for i in itertools.repeat("spam", 10):
3     print(i)
```

- **Zip longest:** This function makes an iterator that aggregates elements from each of the iterables.

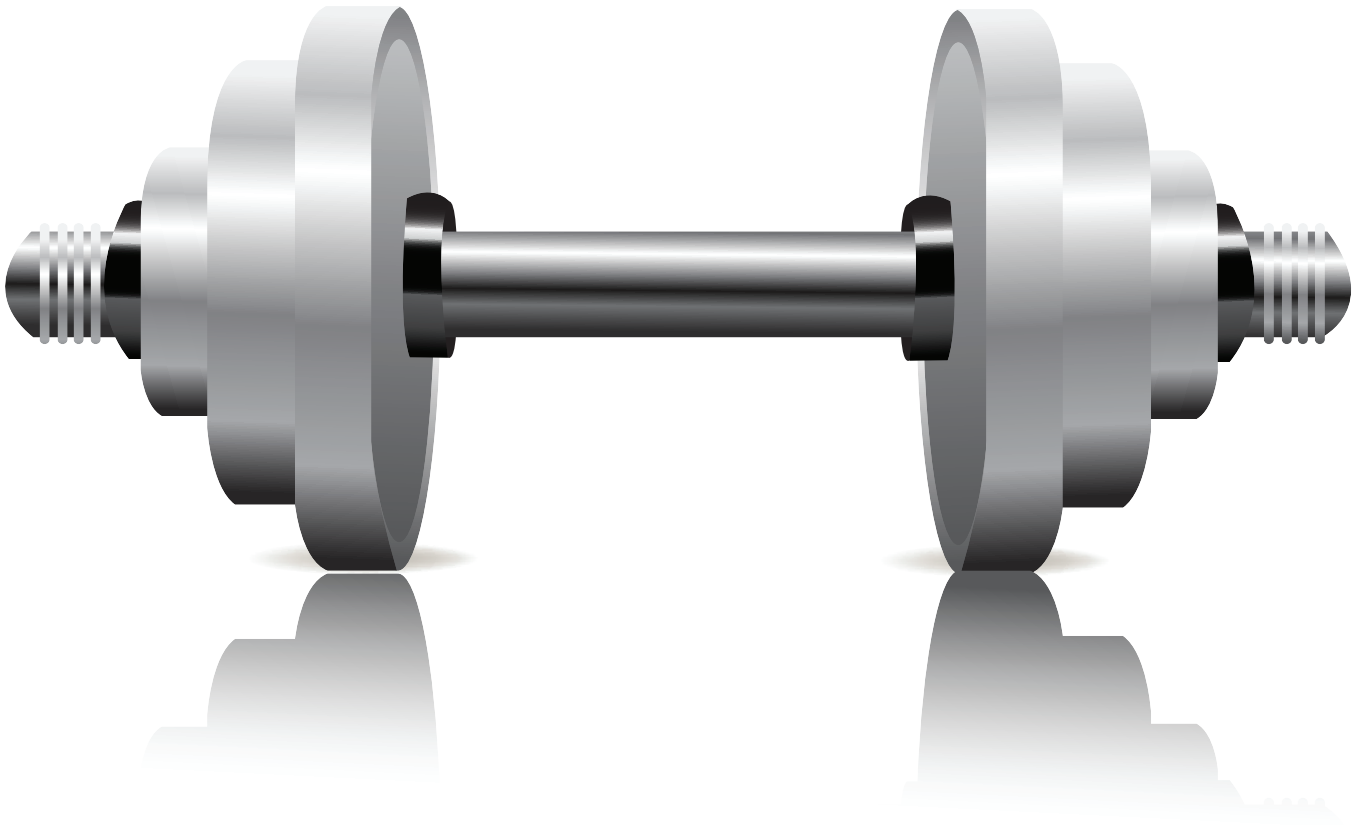
```
1 S1 = ['A', 'B', 'C', 'D', 'E',]
2 data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,]
3 for each in itertools.zip_longest(S1, data, fillvalue=None):
4     print(each)
```

- **Combinations with replacement:** This one allows individual elements to be repeated more than once.

```
1 result = itertools.combinations_with_replacement(S1, 2)
2 for each in result:
3     print(each)
```

Exercise - Lecture 1

- Exercise 4 to 9
- Class-Ex-Lecture1.py



Python (OS, System, Shutil and Counter)

Python Operating System Task

- Python has extensive support for operating system tasks, such as file and folder management
- The great advantage of doing operating system tasks is
* Python code works uniformly on Unix/Linux, Windows, and Mac

OS Important Commands

- **Make a folder**: Create a new folder

```
import os
os.mkdir('test_dir')
```

- **Working on directories:** Find a working directory and change the directory.

```
orig_dir = os.getcwd()
print(orig_dir)
os.chdir(orig_dir + "\\\\" + 'test_dir')
new_dir = os.getcwd()
print(new_dir)
os.chdir(orig_dir)
```

- **Rename:** Rename the directory.

```
os.rename('test_dir', 'new_test_dir')
```

- **List directory:** List the current or any directory.

```
l1 = os.listdir('new_test_dir')
print(l1)
l2 = os.listdir(os.curdir)
print(l2)
print(l2.sort())
```

Copy and Remove

- **Remove a folder**: Removes an entire folder tree.

```
import shutil
import os
import subprocess
shutil.rmtree('new_test_dir')
```

- **Copy:** Copy a folder.

```
os.mkdir('test')
shutil.copytree('test', 'test1')
```

- **Run system commands:** Run terminal commands using python.

```
os.system('python 01_Basic_Pattern.py')
subprocess.call('python 01_Basic_Pattern.py', shell=True)
```

- **Add and split directory:** Join path and split path.

```
path = os.path.join(os.getcwd(), 'test')
print(path)
foldername, basename = os.path.split(path)
print(foldername)
print(basename)
```

Collections Module

- Collections in Python are containers that are used to store collections of data, for example, list, dict, set, tuple etc.
- Several modules have been developed that provide additional data structures to store collections of data.
- One such module is the Python collections module.
  - Counter
  - Defaultdict
  - OrderedDict
  - deque
  - ChainMap

- Create Counter Objects use Counter.

```
1 from collections import Counter
2
3 l1 = [1,2,3,4,1,2,6,7,3,8,1]
4 print(Counter(l1))
5 l2 = ['a', 'b', 'c', 'd', 'a' , 'c', 'c']
6 print(Counter(l2))
7 cnt = Counter(l1)
8 print(cnt[1])
```

- Most Common Elements and Subtract

```
1 print(cnt.most_common())
2 cnt = Counter({1:3,2:4})
3 deduct = {1:1, 2:2}
4 cnt.subtract(deduct)
5 print(cnt)
```

Default Dictionary

- The defaultdict works exactly like a python dictionary, except for it does not throw KeyError when you try to access a non-existent key.

```
1 from collections import defaultdict
2
3 nums = defaultdict(int)
4 nums['one'] = 1
5 nums['two'] = 2
6 print(nums['three'])
```

- For example, let's say you want the count of each name in a list of names.

```
1 count = defaultdict(int)
2 names = "John Julie Jack Ann Mike John John Jack Jack Jen Smith Jen Jen"
3 list = names.split(sep=' ')
4 for names in list:
5     count[names] +=1
6 print(count)
```

Order Dictionary

- OrderedDict is a dictionary where keys maintain the order in which they are inserted, which means if you change the value of a key later, it will not change the position of the key.

```
1  from collections import OrderedDict
2  from collections import Counter
3
4  od = OrderedDict()
5  od['c'] = 1
6  od['b'] = 2
7  od['a'] = 3
8  print(od)
9  for key, value in od.items():
10     print(key, value)
```

- Following example is an interesting use case of OrderedDict with Counter.

```
1  list = ["a","c","c","a","b","a","a","b","c"]
2  cnt = Counter(list)
3  od = OrderedDict(cnt.most_common())
4  for key, value in od.items():
5      print(key, value)
```

- You can create a deque with deque() constructor. You have to pass a list as an argument.

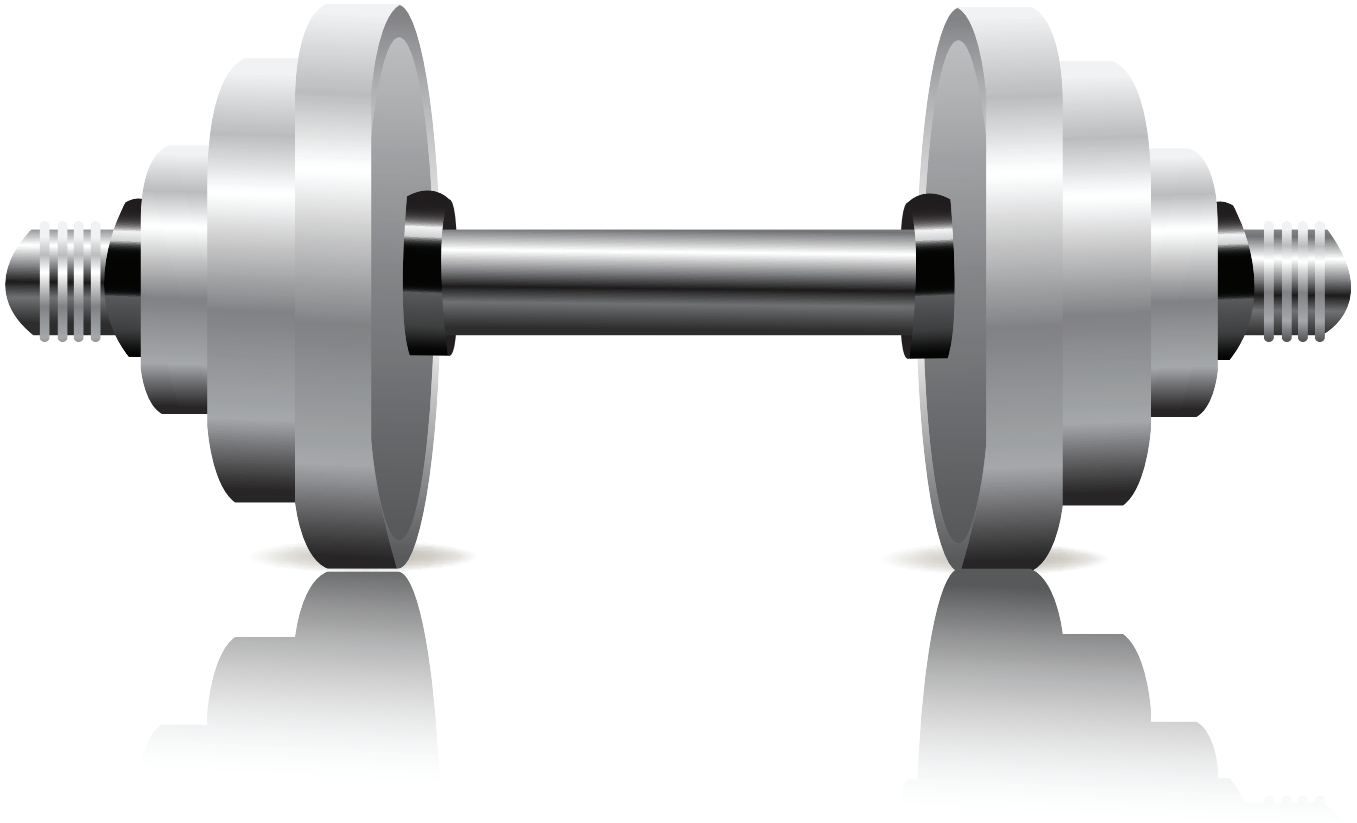
```
1 from collections import deque
2
3 list = ["a","b","c"]
4 deq = deque(list)
5 print(deq)
```

- Inserting Elements, Removing Elements, Clearing and Counting Elements

```
1 deq.append("d")
2 deq.appendleft("e")
3 print(deq)
4
5 deq.pop()
6 deq.popleft()
7 print(deq)
8 print(deq.clear())
9
10 list = ["a","b","c"]
11 deq = deque(list)
12 print(deq.count("a"))
```

Exercise - Lecture 1

- Exercise 10 to 13
- Class-Ex-Lecture1.py



Parser , Linux Terminal, Classes and Numpy

Python argparse 1

- The argparse module makes it easy to write user-friendly command-line interfaces.
- It parses the defined arguments from the sys.argv.
- The argparse module also automatically generates help and usage messages, and issues errors when users give the program invalid arguments.

- argparse optional argument
- argparse required argument
- argparse positional arguments
- argparse type
- argparse default

Python Argparse 2

- Positional argument

```
1 import argparse
2 parser = argparse.ArgumentParser()
3 parser.add_argument('name')
4 parser.add_argument('age')
5 args = parser.parse_args()
6 print('{}'.format(args.name) + ' is '  +'{}'.format(args.age) + ' years old.')
```

- Type argument

```
1 import argparse
2 import random
3
4 parser = argparse.ArgumentParser()
5 parser.add_argument('-n', type=int, required=True, help="define the number")
6 args = parser.parse_args()
7 n = args.n
8 for i in range(n):
9     print(random.randint(-100, 100))
```

- Runner

```
1 import os
2 os.system('python 21_Lemmatization.py Amir 99')
3 os.system('python 23_TF.py -n 3')
```

- Terminal_1.pdf
- Terminal_2.pdf
- Terminal_3.pdf
- git_cheatsheet.pdf

- Please check my Data Mining Repo and review them.
- Data Mining Repo

Exercise - Lecture 1

- Lets work on Linux Terminal
- Class-Ex-Lecture1.py