Copy of Python_Basics

August 12, 2023

[71]:
```
import matplotlib.pylab as plt
import numpy as np
```

Python Basics

1. Data Structures

Lists

A list in Python is a sequence of anything!

[72]:
```
my_list=[1,'a','hello',[1,1,"hello"],(1,3,5),{1,2,3}]

#accessing the elements of a list
print(my_list[1])
print(my_list[3][0])
print(my_list[3][2][0])
print(my_list[-1])
print(my_list[-2])
```

```
a
1
h
{1, 2, 3}
(1, 3, 5)
```

List Slicing

[73]:
```
print(my_list[:2])
print(my_list[2:])
print(my_list[2:4])
print(my_list[-4:-2])
```

```
[1, 'a']
['hello', [1, 1, 'hello'], (1, 3, 5), {1, 2, 3}]
['hello', [1, 1, 'hello']]
['hello', [1, 1, 'hello']]
```

Lists are mutable; you can insert and delete elements

[74]:
```
my_list[0]=2
print(my_list)

my_list=my_list+[2]
print(my_list)

my_list=my_list+[3,4]
print(my_list)

del my_list[6]
print(my_list)

del my_list[6:8]
print(my_list)
```

```
[2, 'a', 'hello', [1, 1, 'hello'], (1, 3, 5), {1, 2, 3}]
[2, 'a', 'hello', [1, 1, 'hello'], (1, 3, 5), {1, 2, 3}, 2]
[2, 'a', 'hello', [1, 1, 'hello'], (1, 3, 5), {1, 2, 3}, 2, 3, 4]
[2, 'a', 'hello', [1, 1, 'hello'], (1, 3, 5), {1, 2, 3}, 3, 4]
[2, 'a', 'hello', [1, 1, 'hello'], (1, 3, 5), {1, 2, 3}]
```

List Methods

[75]:
```
my_list.append("append") #to add only a single element at the end of the list
print(my_list)
```

```
[2, 'a', 'hello', [1, 1, 'hello'], (1, 3, 5), {1, 2, 3}, 'append']
```

[76]:
```
my_list.extend([5,6,7])   #to add several items to the list
print(my_list)
```

```
[2, 'a', 'hello', [1, 1, 'hello'], (1, 3, 5), {1, 2, 3}, 'append', 5, 6, 7]
```

[77]:
```
print(my_list.pop(2))         #the item at the given index is removed
print(my_list)                 #the return value of this function is the␣
 ↪indicated item
print(my_list.pop())         #if no argument is passed the last item is removed␣
 ↪and returned
print(my_list)
```

```
hello
[2, 'a', [1, 1, 'hello'], (1, 3, 5), {1, 2, 3}, 'append', 5, 6, 7]
7
[2, 'a', [1, 1, 'hello'], (1, 3, 5), {1, 2, 3}, 'append', 5, 6]
```

There are many such methods that you can perform on lists...

[78]:
```
numlist=[12,2142,1,101,105]
numlist.sort()
```

```
print(numlist)
numlist.sort(reverse=True)
print(numlist)
```

```
[1, 12, 101, 105, 2142]
[2142, 105, 101, 12, 1]
```

There is another in-built fn "sorted" which takes the list as one of the arguments and it returns the sorted list the other two arguments are similar to that of the sort()

[79]:
```
print(sorted(numlist))
```

```
[1, 12, 101, 105, 2142]
```

ALIASING LISTS

[80]:
```
A=["cocoa",1,2]    #A is a variable which refers to the list
B=A                   #B is yet another variable which refers to the same list

print("A is ",A)
print("B is ",B)
```

```
A is  ['cocoa', 1, 2]
B is  ['cocoa', 1, 2]
```

[81]:
```
A[0]="manjul"
print("A is ",A)
print("B is ",B)
```

```
A is  ['manjul', 1, 2]
B is  ['manjul', 1, 2]
```

This will change the list which is refered to by A which means B is changed as well

MAKING A CLONE

[82]:
```
C=A[:]                #This creates a copy of the list and hence C now refers␣
 ↪to a different list

print("A is ",A)
print("C is ",C)
```

```
A is  ['manjul', 1, 2]
C is  ['manjul', 1, 2]
```

[83]:
```
A[0]="cocoa"        #this will not change the list referred by C
print("A is ",A)
print("C is ",C)
```

```
A is  ['cocoa', 1, 2]
C is  ['manjul', 1, 2]
```

[84]:
```
#We could even do it this way
D=A.copy()
print(D)
```

```
['cocoa', 1, 2]
```

LIST COMPREHENSION

[85]:
```
listy=[2**x for x in range(0,10)]
print(listy)

listy=[2**x for x in range(0,10) if x<5]        #if statement here is called a␣
 ↪filter
print(listy)
```

```
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
[1, 2, 4, 8, 16]
```

Strings

[86]:
```
my_string="hello!"
mystr='''dfgdfg\
dgdgdgdg'''                    #multi line strings are defined using triple␣
 ↪inverted commas

print(my_string)
print(mystr)
```

```
hello!
dfgdfgdgdgdgdg
```

More slicing examples: Guess what each of them would give before printing...

[87]:
```
string="abcdefgh"

print("string[2:6:2]",string[2:6:2])

print("string[-1::-1]",string[-1::-1])

print("string[-1::]",string[-1::])

print("string[::]",string[::])

print("string[::-1]",string[::-1])
```

```
string[2:6:2] ce
string[-1::-1] hgfedcba
string[-1::] h
string[::] abcdefgh
string[::-1] hgfedcba
```

slicing is very much similar to that of lists

strings are IMMUTABLE i.e. they cannot be changed and only entire deletion is possible

Tuples

[88]:
```
#creating tuples
my_tuple="hello",1,2
print(my_tuple)
my_tuple=("hello",1,3)
print(my_tuple)
```

```
('hello', 1, 2)
('hello', 1, 3)
```

indexing and slicing are same as that of lists

TUPLES ARE IMMUTABLE

Sets

DEFINITION:an unordered collection of items which are unique(no duplicates) and immutable.

[89]:
```
#sets cannot have lists,sets or dictionaries as its elements
#Creating sets
my_set={1,'a','hola',(2,3,4),1}

print(my_set)
```

```
{1, 'a', (2, 3, 4), 'hola'}
```

Dictionaries

DEFINITION: Dictionaries are also unordered collection like sets , for each item it has a key and its corresponding value pair. key:value

Values can be any datatypes

But KEYS must be IMMUTABLE like numbers or strings or tuples(inside which also only immutable data types are allowed) and further the keys are to be UNIQUE

[90]:
```
#creating dictionaries
my_dict={1:23,'a':'hello',"hi":{1:'a',2:'e'},(1,2):[(1,3),'s']}
```

[91]:
```
my_dict1 = dict([(1,'apple'), (2,'ball')])        #converted a list(in proper␣
 ↪format) to dictionary
```

Please feel free to experiment these data structures and their methods. And then you can move to the exercises below.

2. Functions

Exercise #1 : Fibonacci sequence

A Fibonacci sequence is the integer sequence of 0, 1, 1, 2, 3, 5, 8. The first two terms are 0 and 1. All other terms are obtained by adding the preceding two terms.This means to say the nth term is the sum of (n-1)th and (n-2)th term.

Your Task: Implement the functions `recur_fibo`

[92]:
```
def recur_fibo(n):

  #Implement the recursive call escape statement
  ## YOUR CODE STARTS HERE
  if n<2: return n

  ## YOUR CODE ENDS HERE
  return recur_fibo(n-1)+recur_fibo(n-2)

nterms = 10 # display the Fibonacci sequence up to n-th term

# check if the number of terms is valid
for i in range(1,nterms+1):
  print(f'{i}th term is {recur_fibo(i)}')
```

```
1th term is 1
2th term is 1
3th term is 2
4th term is 3
5th term is 5
6th term is 8
7th term is 13
8th term is 21
9th term is 34
10th term is 55
```

Exercise #2: Lambda function

A lambda function is a small anonymous function that can take any number of arguments, but can only have one expression.

Your Task: Implement Lambda function for `sigmoid_lambda` and `relu_lambda` whose definition is given below:

$$sigmoid(x) = \frac{1}{1+e^x}$$

$$ReLU(x) = x^+ = max(0, x)$$

[93]:
```
def sigmoid(x):
  sigmoid = 1 / (1 + np.exp(-x))
  return sigmoid

## Implement the above sigmoid function using python lambda functions
```

```
## YOUR CODE STARTS HERE
sigmoid_lambda= lambda x: 1/(1+np.exp(-x))
## YOUR CODE ENDS HERE

assert sigmoid(10)==sigmoid_lambda(10)
```

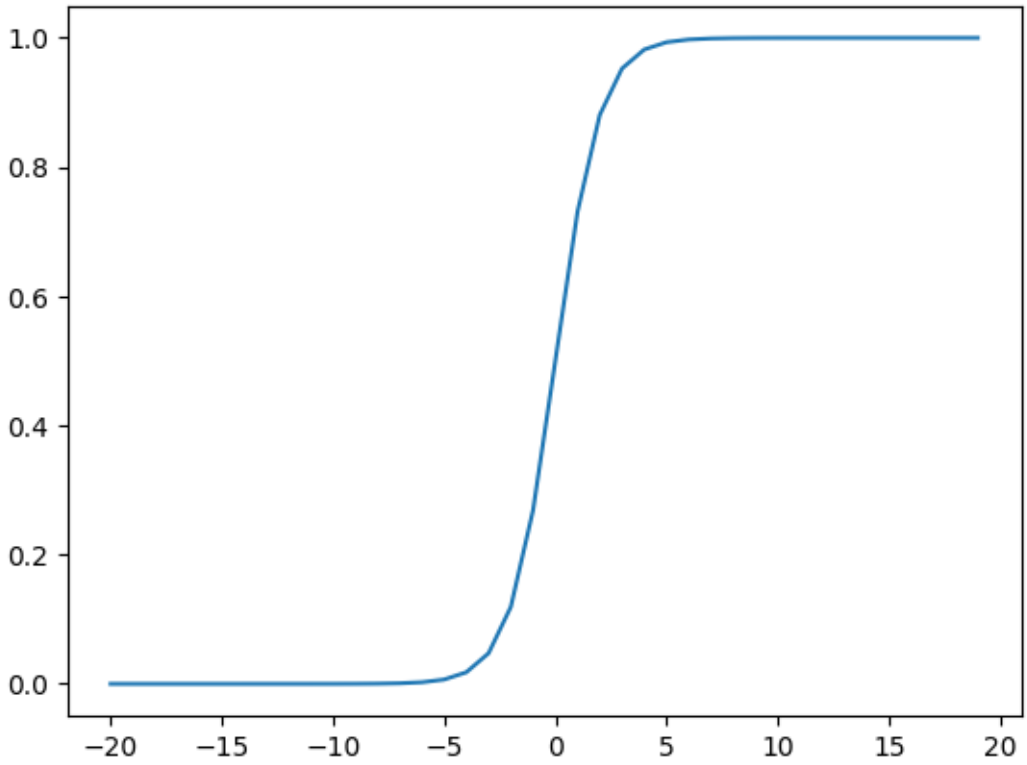
[94]:
```
def relu(x):
  relu = x if x>0 else 0
  return relu

## Implement the above sigmoid function using python lambda functions
## YOUR CODE STARTS HERE
relu_lambda=lambda x: max(x,0)
## YOUR CODE ENDS HERE
assert relu(10)==relu_lambda(10)
```

Exercise #3: Python map keyword

Your Task: Using the lambda functions and python's map keyword apply `sigmoid` and `relu` on a list of numbers from -20 to 20. Store the output in a list and print it.

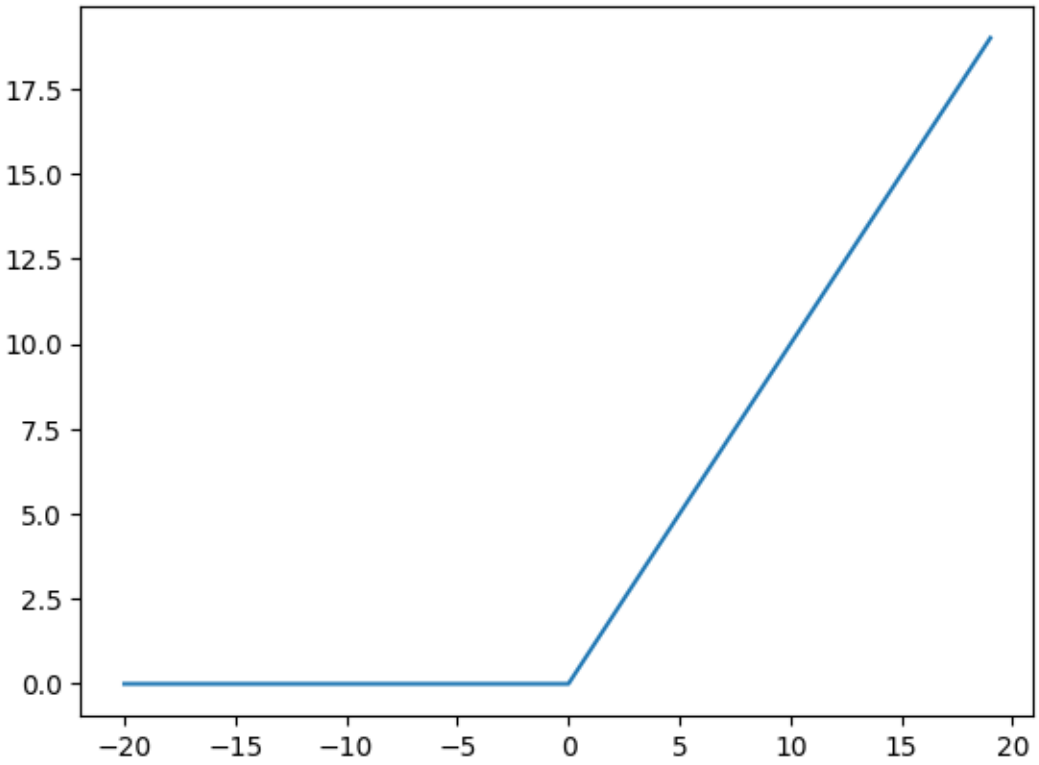
[95]:
```
x_list = [*range(-20,20)]
## YOUR CODE STARTS HERE
sigmoid_out=list(map(sigmoid_lambda, x_list))
## YOUR CODE ENDS HERE
print(sigmoid_out)
plt.plot(x_list,sigmoid_out);
```

```
[2.0611536181902037e-09, 5.602796406145941e-09, 1.522997951276035e-08,
4.1399375473943306e-08, 1.12535162055095e-07, 3.059022269256247e-07,
8.315280276641321e-07, 2.2603242979035746e-06, 6.144174602214718e-06,
1.670142184809518e-05, 4.5397868702434395e-05, 0.00012339457598623172,
0.0003353501304664781, 0.0009110511944006454, 0.002472623156634743,
0.006692850924284855 4, 0.01798620996209156, 0.04742587317756678,
0.11920292202211755, 0.2689414213699951, 0.5, 0.7310585786300049,
0.8807970779778823, 0.9525741268224334, 0.9820137900379085, 0.9933071490757153,
0.9975273768433653, 0.9990889488055994, 0.9996646498695336, 0.9998766054240137,
0.9999546021312976, 0.999983298578152, 0.9999938558253978, 0.999997739675702,
0.9999991684719722, 0.999999694097773, 0.9999998874648379, 0.9999999586006244,
0.9999999847700205, 0.9999999943972036]
```



[96]:
```
x_list = [*range(-20,20)]
## YOUR CODE STARTS HERE
relu_out=list(map(relu_lambda, x_list))
## YOUR CODE ENDS HERE
print(relu_out)
plt.plot(x_list,relu_out);
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 5,
6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```



Exercise #4: On Data structures

Your Task: You are given a paragraph as a string. Write a function to return a sorted dictionary with keys as the set of words and the corresponding values as their frequency in the paragraph

[97]:
```
import re

def count_word_frequency(paragraph):
    # Convert the paragraph to lowercase to ensure case-insensitivity
    paragraph = paragraph.lower()
    paragraph=re.sub(r'[^a-z\s]', '', paragraph)

    # Split the paragraph into words
    words = paragraph.split()

    # Initialize an empty dictionary to store word frequencies
    word_freq = {}

    # Write a for loop to update the frequencies of each word
    ## YOUR CODE STARTS HERE
    for word in words:
      if word in word_freq: word_freq[word]+=1
      else: word_freq[word]=1
```

```
    ## YOUR CODE ENDS HERE

    return dict(sorted(word_freq.items(), key=lambda item: item[1],␣
 ↪reverse=True))
```

[98]:
```
# Test the function
paragraph = """
Language is a fundamental aspect of human communication. It allows us to convey␣
 ↪ideas, emotions, and information to one another. From spoken words to␣
 ↪written texts, language takes various forms, each with its unique beauty.␣
 ↪The evolution of languages has been a fascinating journey, shaped by␣
 ↪cultural interactions and historical events.

English, one of the most widely spoken languages globally , has a rich history.␣
 ↪It originated in England and gradually spread across the world due to␣
 ↪colonization and globalization. Today, it serves as a lingua franca for␣
 ↪international business, diplomacy, and academia. Its flexibility and␣
 ↪adaptability have contributed to its widespread adoption.

Languages are not just a means of communication; they also shape the way we␣
 ↪think and perceive the world. Linguists study the intricate structures and␣
 ↪grammar of languages to understand how they influence cognition. Each␣
 ↪language reflects the culture and values of its speakers, carrying the␣
 ↪weight of their history.

As technology advances , language continues to evolve. New words and phrases␣
 ↪emerge, reflecting modern trends and innovations. The internet , with its␣
 ↪global reach, has played a significant role in disseminating languages and␣
 ↪creating new digital communication styles.

Preserving endangered languages is an essential endeavor to maintain cultural␣
 ↪diversity. Many languages are at risk of disappearing as fewer speakers pass␣
 ↪them on to the next generation. Efforts are being made to document and␣
 ↪revitalize endangered languages, recognizing their significance in␣
 ↪preserving unique cultural heritage.

In conclusion , language is a powerful tool that shapes human interactions and␣
 ↪defines cultures. It is a bridge that connects people across the world,␣
 ↪fostering understanding and empathy. As we cherish and celebrate linguistic␣
 ↪diversity, we embrace the richness of humanity's collective heritage.
"""
```

[99]:
```
result = count_word_frequency(paragraph)
print(result)
```

```
{'and': 16, 'the': 12, 'to': 10, 'of': 9, 'a': 8, 'languages': 8, 'language': 5,
'its': 5, 'is': 4, 'it': 4, 'in': 4, 'as': 4, 'communication': 3, 'has': 3,
```

```
'cultural': 3, 'world': 3, 'are': 3, 'we': 3, 'human': 2, 'one': 2, 'spoken': 2,
'words': 2, 'each': 2, 'with': 2, 'unique': 2, 'interactions': 2, 'history': 2,
'across': 2, 'they': 2, 'speakers': 2, 'their': 2, 'new': 2, 'preserving': 2,
'endangered': 2, 'diversity': 2, 'heritage': 2, 'that': 2, 'fundamental': 1,
'aspect': 1, 'allows': 1, 'us': 1, 'convey': 1, 'ideas': 1, 'emotions': 1,
'information': 1, 'another': 1, 'from': 1, 'written': 1, 'texts': 1, 'takes': 1,
'various': 1, 'forms': 1, 'beauty': 1, 'evolution': 1, 'been': 1, 'fascinating':
1, 'journey': 1, 'shaped': 1, 'by': 1, 'historical': 1, 'events': 1, 'english':
1, 'most': 1, 'widely': 1, 'globally': 1, 'rich': 1, 'originated': 1, 'england':
1, 'gradually': 1, 'spread': 1, 'due': 1, 'colonization': 1, 'globalization': 1,
'today': 1, 'serves': 1, 'lingua': 1, 'franca': 1, 'for': 1, 'international': 1,
'business': 1, 'diplomacy': 1, 'academia': 1, 'flexibility': 1, 'adaptability':
1, 'have': 1, 'contributed': 1, 'widespread': 1, 'adoption': 1, 'not': 1,
'just': 1, 'means': 1, 'also': 1, 'shape': 1, 'way': 1, 'think': 1, 'perceive':
1, 'linguists': 1, 'study': 1, 'intricate': 1, 'structures': 1, 'grammar': 1,
'understand': 1, 'how': 1, 'influence': 1, 'cognition': 1, 'reflects': 1,
'culture': 1, 'values': 1, 'carrying': 1, 'weight': 1, 'technology': 1,
'advances': 1, 'continues': 1, 'evolve': 1, 'phrases': 1, 'emerge': 1,
'reflecting': 1, 'modern': 1, 'trends': 1, 'innovations': 1, 'internet': 1,
'global': 1, 'reach': 1, 'played': 1, 'significant': 1, 'role': 1,
'disseminating': 1, 'creating': 1, 'digital': 1, 'styles': 1, 'an': 1,
'essential': 1, 'endeavor': 1, 'maintain': 1, 'many': 1, 'at': 1, 'risk': 1,
'disappearing': 1, 'fewer': 1, 'pass': 1, 'them': 1, 'on': 1, 'next': 1,
'generation': 1, 'efforts': 1, 'being': 1, 'made': 1, 'document': 1,
'revitalize': 1, 'recognizing': 1, 'significance': 1, 'conclusion': 1,
'powerful': 1, 'tool': 1, 'shapes': 1, 'defines': 1, 'cultures': 1, 'bridge': 1,
'connects': 1, 'people': 1, 'fostering': 1, 'understanding': 1, 'empathy': 1,
'cherish': 1, 'celebrate': 1, 'linguistic': 1, 'embrace': 1, 'richness': 1,
'humanitys': 1, 'collective': 1}
```

[OPTIONAL Exercise] Do you observe any mistake in the output? Can u point out which part of the provided code is leading to this mistake and possibly correct it.

[100]:
```
'''Here the punctuations sometimes occur just next to a word so for ex: "and"␣
 ↪and "and,"
will be counted as 2 different words and be counted twice, but this has been␣
 ↪rectified by using re.sub'''
```

[100]:
```
'Here the punctuations sometimes occur just next to a word so for ex: "and" and
"and," \nwill be counted as 2 different words and be counted twice, but this has
been rectified by using re.sub'
```