

```

1  #Learning python from scratch
2  SAGAR KALAUNI
3  southern Illinois University Edwardsville
4  statistics || Data Science
5  Teaching Assistant @SIUE
6
7  Chapter-1:5 PYTHON PROGRAMMING BASICS
8  ~~~~~
9
10 # How to print the command in the python
11 print("Hello world, I am sagar")
12
13 # how to assign the value in python
14 x=1
15 print(x)
16
17 x=x+1
18 print(x)
19 ~~~~~
20
21 # Adding two numbers and two stings using python
22 # adding two numbers
23 a=2+3
24 print(a)
25
26 # adding two strings
27 Name= "sagar" + " " + "Kalauni"
28 print(Name)
29 ~~~~~
30
31 # how to look our variable data type in python
32 x=32
33 type(x)
34
35 y="Sagar"
36 type(y)
37 ~~~~~
38
39 # Data type conversion using python
40 x=35
41 float(x)
42
43 sval="123"
44 type(sval)
45
46 ival=int(sval)
47 type(ival)
48 ~~~~~
49
50 # There are 4 basic type of coding practise, out of which we will discuss about first 3
51 now, they are
52 # Sequential coding, conditional coding and Repeted coding. and fourth one is store and
53 reuse
54
55 # sequential code
56 # will be excuted in a sequence
57 x=2
58 print(x)
59 x=x+2
60 print(x)
61 ~~~~~
62
63 # conditional steps code
64 x=5
65 if x<10:

```

```

60     print("smaller")
61     if x>20:
62         print("greater")
63     print("finish")
64
65     -----
66
67     # Repeted step coding
68     n=5
69     while n>0:
70         print(n)
71         n=n-1
72     print("done")
73
74     # A big and nice programming code containg all these above in one
75     -----
76
77     # Modulo operator in python
78     # let's not talk about simple addition and subtraction operator in python, instead
79     # directly talk about the modulo operator in python
80     y=53
81     z= y % 2 # this will return the reminder when divided by 2, can be used to pick the
82     # small number from big, and other code development ideas
83     -----
84
85     #operator precedence in python: paranthesis--power--multiplication--addition-Left to
86     # right.
87     -----
88
89     # INPUT FUNCTION in python
90     # We can instruct python to pause and read data from the user using the input()
91     # The input function always return strings
92
93     name=input("who are you?")
94     print("Welcome", name)
95     -----
96
97     # Time to write our very first code on the basis of the knowledge above
98     # Europe floor to US floor conversion in Elevator
99
100     europe_floor=input("Europe floor")
101     us_floor= int(europe_floor) + 1
102     print("The equivalent Us floor is:", us_floor)
103     -----
104
105     # Homework-1
106     # Write a program to prompt the user for hour and rate per hour to compute a gross pay?
107
108     Hour=input("Enter Hour: ")
109     Rate=input("Enter Rate per Hour: ")
110     Gross_pay= float(Hour) * float(Rate)
111     print(Gross_pay)
112     -----
113
114     # Try and except in python
115
116     string="sagar"
117     try:
118         print("this lines runs")
119         print("this also")
120         print("this too")
121         integer=int(string) # At this line boom happens, so after that it
122         # will directly go to except part of the code

```

```

116     print("let's see this line runs or not")
117 except:
118     integer= 32
119
120 print(string, integer)
121
122 -----
123 # Homework-2
124
125 # Rewrite your pay program using try and except so that your program can handles non-
126 numeric input gracefully by printing a message and exiting the program?
127
128 Hour= input("Enter hour: " )
129 Rate= input("Enter Rate: " )
130 try:
131     float_Hour=float(Hour)          # the code blows at this point then will not even look
132     at bellow part and directly go to the except part of the code
133     float_rate=float(Rate)
134 except:
135     print("Enter the Numeric Values!")
136     quit()
137 gross_pay= float_Hour * float_rate
138 print(gross_pay)
139
140 -----
141
142 # Now here we will discuss about our fourth coding way i.e store and Reuse.
143 # For doing this we will create our own function having certain code, so that we did not
144 have to type the same chunk of code again and again for the next time
145
146 def thing():
147     print("do your stuff over here")
148     print("print your stuff over here")          # till this part we have just created
149     our own customized function
150
151 print("-----below I will call my thing() function-----")
152 thing()          # it's time to call our function
153 print("-----did you just see, I did not wrote code to print above stuff
154 just call my thing function-----")
155
156 -----
157
158 # building a function and calling a function are two completely different things.
159 # once we built our own function, it will not give any output untill we call the function
160
161 # building a function[store part]
162 def song_lyrics():
163     print("give me some sun shine \n give me some rain \n give me another chanse
164           \n I wanna gorw up once again")
165
166 # Calling a function[reuse part]
167 song_lyrics()
168
169 -----
170
171 # Program to find the relationship
172 # defining a function
173 def relation(friend):
174     if friend=="kashi":
175         print("Friend")
176     elif friend=="biru":
177         print("Brother")
178     elif friend=="yogesh":
179         print("Nephew")
180     elif friend=="owen":
181         print("classmate")
182     else:

```

```

173         print("Unknown")
174 # calling our function back
175 relation('yogesh')
176 -----
177
178 # talking about return in our own customized function
179
180 def greet():
181     return "The number you have dialed is currently busy\nplease dial later"
182 def system(call):
183     if call=="busy":
184         print(greet())
185
186 # calling the function back
187 system("busy")
188
189 #print(greet(), "sagar")
190 -----
191
192 # LOOPS AND ITERATIONS IN python
193 # while and for are the keywords used for doing this [definite and indefinite loop]
194
195 # infinite loop
196 # this may happen because of error in the code or sometimes you may need it, but
197 # definitely stay away from this if your computer is not good enough
198 # Its never a good idea to put your computer in a infinite loop
199 n=10
200 while n>0:
201     print("Hie hello world")
202     n=n+1 # for infinite loop
203
204 # Zero loop code
205 n=0
206 while n>0:
207     print("hie hello world")
208     Print("loop is working")
209 print("I am outside of the loop")
210 -----
211
212 # How to go out of the loop and execute the code after loop
213 # we can use the break function to exit loop
214 # let's write an infinite loop and and get out of the loop using break statement
215
216 # infinite loop with break to come out of the loop
217 while True:
218     password=input("Enter your birth city: ")
219     if password=="baitadi":
220         break
221 print("Wellcome to your mobilephone")
222 -----
223
224 # continue statement in Python
225 # the continue statement ends the current iteration and jumps to the top of the loop and
226 # start the next iteration.
227 # break completely exit the loop while continue just stop at that point go back to the
228 # top to do next iteration. (does not move forward in ceratin iteration)
229
230 while True:
231     password=input("Enter a password:")
232     if password[0]=="#":
233         continue
234     elif password=="baitadi":
235         break
236 print(your password)
237 -----

```

```

232 # definite loops in python/ looping using for
233
234 for i in [1,2,3,4]:
235     print(i)
236 print("done")
237
238 for name in ["sagar", "dipendra", "suraj", "bijay"]:
239     print(name)
240 -----
241
242 # Using the loop ideas to solve our problems
243 # 3,    41,    12,    9,    74,    15 find the largest among these numbers
244
245 # writing the code for find the lagrest number
246 largest_num_till_now=-1
247 print("befor", "max:-",largest_num_till_now)
248 for num in [1,5,96,2,17,99]:
249     if largest_num_till_now< num:
250         largest_num_till_now=num
251         print(largest_num_till_now, num)
252 print("after", "max:-",largest_num_till_now)
253 -----
254
255 # Loop Idioms
256 # [1]counting the number of items in the list using the for loop
257
258 count=0
259 print("befor", count)
260 for item in [0,9,65,85,66,38,36,649,2165,5,32]:
261     count= count + 1
262 print("after", count)
263 -----
264
265 # [2]adding the number of items in the list using the for loop
266
267 sum=0
268 print("befor", sum)
269 for items in [0,9,65,85,66,38,36,649,2165,5,32]:
270     sum= sum + items
271     Running_sum=sum
272     print("Running sums",Running_sum)
273 print("after", sum)
274 -----
275
276 # Homework-3
277 # [3]finding the average of items in the list using the for loop
278 # Just remember the average is total_sum/ total_count
279
280 total_count=0
281 total_sum=0
282 for item in [0,9,65,85,66,38,36,649,2165,5,32]:
283     total_count= total_count + 1
284     total_sum= total_sum + item
285 print("the average is" total_sum/total_count)
286 -----
287
288 # [4]filtering the item from the list using the for loop
289 # Let's find the items which are greater then 100 from our list
290
291 count=0
292 for item in [0,9,65,85,66,38,36,649,2165,5,32]:
293     if item > 100:
294         count=count + 1
295         print(count," greater then 100 number:-", item)
296 -----

```

```

-----
294 # [5]searching for the particular item in the list using the boolean variable and for
    loop.
295 # boolean is the another variable type beside int, float and string and it has its value
    true or false
296
297 found=False
298 #print("before", found)
299 for item in [0,9,65,85,66,38,36,649,2165,5,32]:
300     if item==649:
301         found=True
302 print("List contain the number:-", found)
303
-----
304 # quiz- like we use largest_num_till_now=-1 in the code for finding the maximum number
    in the list. how can we modify that code to find the smallest number in the list
305 # It is way harder to say this is the maximum number so we introduce the concept of new
    number type call None.
306
307 smalles_num_till_now=None
308 for value in [55,9,65,85,66,38,36,649,2165,5,32]:
309     if smalles_num_till_now is None:
310         smalles_num_till_now= value
311     elif value < smalles_num_till_now:
312         smalles_num_till_now= value
313     print(smalles_num_till_now, value)
314 print("smallest number in our list is:-", smalles_num_till_now)
315
-----
316 Homework-4
317 #Write a program that repeatedly prompts a user for input numbers until the user enters
    'done'.
318 #Once 'done' is entered, print out the total sum, total number and average of the numbers
319 #If the user enters anything other than a valid number catch it with a try/except
320 #and put out an appropriate message and ignore the number.
321 #Enter 6, 5, bob, and 1 and match the output below.
322 # total_sum: 12, total_num: 3, average: 4
323
324 # infinite loop with break to come out of the loop
325 total_sum=0
326 total_count=0
327 while True:
328     sval=input("Enter the number:-")
329     if sval=="done":
330         break
331     try:
332         fval= float(sval)
333     except:
334         print("Invalid Input")
335         continue
336     total_sum=total_sum + fval
337     total_count= total_count + 1
338     #print(total_sum,total_count)
339 print("total sum: ", total_sum, "total num: ",total_count, "average: ", total_sum/
    total_count)
340
341
342
343 Homework-5
344 #Write a program that repeatedly prompts a user for integer numbers until the user
    enters 'done'.
345 #Once 'done' is entered, print out the largest and smallest of the numbers.
346 #If the user enters anything other than a valid number catch it with a try/except
347 #and put out an appropriate message and ignore the number.
348 #Enter 7, 2, bob, 10, and 4 and match the output below.
349 #Invalid input
350 #Maximum is 10

```

```

351 #Minimum is 2
352
353 largest=None
354 smallest=None
355
356 while True:
357     sval= input("inter a number:")
358     if sval=="done":
359         break
360     try:
361         int_val=int(sval)
362         if smallest is None:
363             smallest= int_val
364         if int_val < smallest:
365             smallest= int_val
366         if largest is None:
367             largest= int_val
368         if int_val > largest:
369             largest= int_val
370     except:
371         print("Invalid input")
372         continue
373 print(smallest, largest)
374 -----chapter-1 The End
-----
375 #####
376 #####
377 #Learning python from scratch
378 SAGAR KALAUNI
379 southern Illinois University Edwardsville
380 statistics || Data Science
381 Teaching Assistant @SIUE
382
383 Chapter-6:10 PYTHON DATA STRUCTURE
384 ~~~~~
385 ~~~~~
386 # Looking inside the string
387 # How to access to character inside the string
388 # python start with zero position, keep that in mind before doing any position accessing
389 # coding
390
391 fruit= "Apple"
392 fruit[3]          # can be numeric
393
394 x=5
395 fruit[x-2]        # can be equation
396 -----
397 -----
398 # length function in python
399
400 fruit= "pineapple"
401 len(fruit)        # it returns the number of character in the variable
402
403 # quiz- Write your own function to find the length of the character string
404 def length_count(variable):
405     count=0
406     for i in variable:
407         count = count + 1
408     return count
409
410 length_count('apple')
411 -----
412 -----
413 # Looping through the index
414
415 fruit= "Pineapple"

```

```

411 index= 0
412 while index < len( fruit):
413     letter= fruit[index]
414     print(index, letter)
415     index= index + 1
416 -----
417 # accessing the letters of the string using the for loop
418 fruit= "Pineapple"
419 for letter in fruit:
420     print(letter)
421 -----
422 # Homework-6
423 # Counting the numbers of letters (certain) in the words
424
425 fruit= "Pineapple"
426 count=0
427 for letter in fruit:
428     if letter=="p":
429         count= count + 1
430 print(count)
431 -----
432 # Slicing of the string in the python
433 # [a:b] --> this means start with a and go upto b but not does not include b. {b is not
434 included}
435
436 string= "Hello my name is sagar"
437
438 string[0:4] # this means from H and go upto o but do not include it. {in simple 4th
439 character is not included}
440
441 string[5:7]
442
443 string[5:1000000] # this is also okay for python but it will take more memory
444
445 string[ :4] # from starting and up ot 4.
446
447 string[5: ] # from 5th place till the end
448
449 string[ : ] # whole string
450 -----
451
452 # string concatenation in python
453
454 name = "sagar"
455 full_name= name + "kalauni"
456 print(full_name)
457
458 full_name= "Sagar" + " " + "Kalauni"
459 print(full_name)
460 -----
461
462 # Using in as a logical operator in python
463 fruit = "Pineapple"
464
465 'p' in fruit
466
467 "s" in "sagar"
468
469 "t" in "sagar"
470
471 " " in "sagar"
472 -----

```



```

471 # quiz- Write a program that will look for certain letter in the string and print out
472 found is the letter is in the string
473
474 string= " Mahabharat"
475 if 'M' in string:
476     print("found it")
477 else:
478     print("not found")
479
480 # string comparsion
481 # comparing the string means comparing the each letter of the string with other string
482 # a=1 and z=26 is the value for them, we will go in detail later so 'z' > 'a'; bigger
483 letter bigger number
484
485 'sagar' > 'bharat'
486
487 'suraj' < 'sagar'
488
489 # quiz- short the name Ram, shyam and hari using string comparsion
490
491 name= input("Enter name")
492 if name == "Ram":
493     print("my refrence name is:-", name)
494
495 if name > "Ram":
496     print("this name", name, "comes after Ram")
497
498 if name < "Ram":
499     print("this name", name, "comes before Ram")
500
501 # Lower and upper function for string
502
503 greet= " Hello world"
504 greet.lower() # original is not changed, original is still same
505
506 greet # cross checked
507
508 converted= greet.lower()
509 converted
510
511 # for upper
512
513 greet= " Hello world"
514 greet.upper() # original is not changed, original is still same
515
516 greet # cross checked
517
518 converted= greet.upper()
519 converted
520
521 # Similarly there are a lot of other libraries which we can use in the string, to know
522 more check out theses
523
524 stuff= "hellow world"
525 type(stuff)
526
527 dir(stuff)
528 # you can play with all of these string libraries, i will some few of them, below
529
530 "AAAA".capitalize() # make first letter capitalize

```

```

530 -----
531 # find function in string libraries
532
533 fruit= "Pineapple"
534
535 fruit.find('l')
536 position=fruit.find('l')
537
538 print(position)
539
540 fruit.find('z') # if you look for something that is not there you will get -1
541
542 # find function can take two arguments: first one is what to find and second one is from
    where should I start
543 fruit.find('a', 'i')
544 -----
545 # search and replace function in the string library
546 # search and replace function works in such a way that, first argument we have to give is
    the item which need to be searched from the string and second argument is what is need
    ot be replaced with
547
548 string= "Hello Sagar"
549 string.replace('Sagar', 'Bharat')
550
551 string.replace('S', 'B') # search for S in our string and replace it with B
552 -----
553 # Strapping whitespace from the string using the strip() function in string library
554
555 word="    hello    sagar    "
556 word.lstrip()
557 word.rstrip()
558 word.strip()
559 -----
560 # startswith() function in the string library
561 # sometime we need to find weather the line that start with particular word. we can use
    it over there
562
563 line= "Breadth there the man whose soul so die"
564 line.startswith('Breadth')
565
566 line.startswith('P')
567 -----
568 # Homework-7
569 # bellow is the line given from the line only slice and extract the university short
    format
570
571 line= "From Augustin.Marcus@siue.edu Jan 4 2024 06:32PM"
572
573 # since we want the slice siue only from the above line, first we will find (look) for
    the starting position
574
575 spos = line.find('@')
576
577 # now secondly we need to find the ending position of the line
578
579 lpos = line.find('.', spos) # this code will look for the '.' starting from spos
    value in the line
580
581 # findally we get the exact slice we just have to slice it with proper start and end
    point
582 line[spos + 1 : lpos]
583 -----

```

```

584 # Homework-8
585 # Take the following python code that store a string: string = "X-DSPAM-Confidence:
0.8475";
586 #Write code to extract the number at the end of the line below and convert the extracted
value to a floating point number and print it out.
587
588 string = "X-DSPAM-Confidence:    0.8475"
589
590 spos= string.find(':')
591 lpos=string.find('5', spos)
592
593 sliced= string[spos + 1: lpos + 1]
594
595 num = sliced.strip()
596
597 num=float(num)
598 print(num)
599 -----taking data from our computer
-----
600 # new line character in python,
601 print('x\ny')
602
603 len('x\ny')    # new line character is a single character \n
604 -----
-----
605 # How to read file using python
606 # impnote, python treat filehandle as a sequence of lines
607 # file handle act as sequence of string where eaach line in the file is a string in the
sequence
608 # open() does not actually read the file, it kind of give us the protal where we can
look for the file
609 fhandle= open("C:/Users/Dell/Desktop/Learn with Friedns/Learning python from Zero/Data
folder/mbox.txt")
610
611 print(fhandle) # it is a python object, we will talk about it later, for now just
remember open does nto actually read the file but only give us protal to look at it
612 -----
-----
613 # WORKING DIRECTORY IN Python
614 # How to look for what is my working directory right now in python
615 pwd()
616
617 import os
618 os.getcwd()    # full form of cwd is current working directory
619
620 # how to change the working directory in python
621
622 import os
623 os.chdir('INside this write the path you want to set working directory to')
624 -----
-----
625
626 # counting the number of lines in the paragraph
627
628 fhandle= open("C:/Users/Dell/Desktop/Learn with Friedns/Learning python from Zero/Data
folder/mbox.txt")
629
630 count=0
631 for line in fhandle:
632     count= count + 1
633 print(count)
634 -----
-----
635
636 # how to read file using python
637

```

```

638 fhandle= open("C:/Users/Dell/Desktop/Learn with Friedns/Learning python from Zero/Data
639 folder/mbox.txt")
640 -----
641 # searching through a file
642
643 fhandle= open("C:/Users/Dell/Desktop/Learn with Friedns/Learning python from Zero/Data
644 folder/mbox.txt")
645 for line in fhandle:
646     if line.startswith('From'):
647         print(line)
648 -----
649 # if you look at the output of the above code you can see that there is space between
650 each line. this is because print statements adds new line (every time, always true)
651 # and also each line from the file has newline at the end
652 # quiz-How to deal with them?
653 fhandle= open("C:/Users/Dell/Desktop/Learn with Friedns/Learning python from Zero/Data
654 folder/mbox.txt")
655 for line in fhandle:
656     line= line.rstrip()
657     if line.startswith("From"):
658         print(line)
659
660 # we will do this kind of coding a lot in the future class so you need to understand
661 first three line of code completely i.e loading a data file, looking through the data
662 file and removing \n
663 -----
664 # quiz- How you will write a python code that will print the exact the same output as
665 above but with different logic?
666 # Idea is we can skip all the line that does not startswith From clause.
667 # skipping with continue
668 fhandle= open("C:/Users/Dell/Desktop/Learn with Friedns/Learning python from Zero/Data
669 folder/mbox.txt")
670 for line in fhandle:
671     line = line.rstrip()
672     if not line.startswith("From"):
673         continue
674     print(line)
675 -----
676 # using in to select lines
677
678 fhandle= open("C:/Users/Dell/Desktop/Learn with Friedns/Learning python from Zero/Data
679 folder/mbox.txt")
680 for line in fhandle:
681     line= line.rstrip()
682     if "umich.edu" in line:
683         print(line)
684 -----
685 # quiz- how to get the same code with different logic
686 # exactly by skipping line with continue
687
688 fhandle= open("C:/Users/Dell/Desktop/Learn with Friedns/Learning python from Zero/Data
689 folder/mbox.txt")
690 for line in fhandle:
691     line= line.rstrip()

```

```

690     if not "umich.edu" in line:
691         continue
692     print(line)
693     -----
694     # proper coding format
695
696     fname= input("Enter a file name: ")
697     fhandle= open(fname)           # because input() always give string as the output
698
699     count= 0
700     for line in fhandle:
701         count = count + 1
702     print(count)
703     -----
704     # Homework-9
705     # Write a code to find how many lines starts with 'Subject' in mbox.txt and
706     # mbox-short.txt files?
707
708     fname= input("Enter a file name:")
709     fhandle= open(fname)
710
711     count=0
712     for line in fhandle:
713         line= line.rstrip()
714         if line.startswith("Subject"):
715             count = count + 1
716     print(count)
717     -----
718     # quiz- The above code will blow up if the user input the wrong file name, we don't want
719     # code to blow up, what can be done.
720     # Correct we can use try and except
721
722     fname= input("Enter a file name:")
723     try:
724         fhandle= open(fname)
725     except:
726         print("Invalid file name:-", fname)
727         quit()
728
729     count=0
730     for line in fhandle:
731         line= line.rstrip()
732         if line.startswith("Subject"):
733             count= count + 1
734     print(count)
735     -----
736     # Homework-10
737     # Write a program to read through a file[mbox.txt] and print the content of the file (
738     # line by line) all in upper case.
739
740     fname= input("Enter a file name:")
741     fhandle= open(fname)
742     for line in fhandle:
743         line=line.rstrip()
744         print(line.upper())
745     -----
746     # LIST in the python
747     # till now we are talking about single variable, but python data structure can contain
748     # more than one variable in the form of list and other stuffs
749     # List in the python means the collection of the variables and the variables can be any
750     # string or numeric
751     # list is denoted by []

```

```

747
748 num_list= [1,2,3,4]
749 print(num_list)
750 type(num_list)
751
752 var_list=['Ram','shyam','kanha']
753 print(var_list)
754 type(var_list)
755 -----
756 # It is not necessary that list should contain same data type
757
758 mix_list= ['ram', 32, 36, 44, 'x']
759 print(mix_list)
760 -----
761 # List inside list
762 # A list can have list inside it
763
764 new_list= [ 32, "ram", [1,3, 4], 55, 6545]
765 print(new_list)
766
767 # Empty list in python
768 emp_list=[]
769 print(emp_list)
770 -----
771 # List and definite loop
772
773 friends= ["kashi", "Dipendra", "Suresh", "Hemu"]
774 for friend in friends:
775     print(friend)
776 -----
777 # Looking inside the list
778
779 friends= ["kashi", "Dipendra", "Suresh", "Hemu"]
780
781 print(friends[1]) # how to read it: it is friends of 1
782 -----
783 # List are Mutable
784 # if you remember variables are not mutable but list are mutable.
785
786 string= "SAGAR"
787 string.lower() # this code does not actually change the Sagar to lower so not mutable
788 string
789
790 string= string.lower() # now this code does change, because here change is being saved
to the original. this automatically happens in list so list is called mutable
791 string
792
793 # for list
794
795 item_list= ["pen", "car", "mobilephone", "laptop"]
796
797 item_list[1]= "toyota car" # this actually change the original item_list so list is
called mutable.
798 # At this point list is truly change
799 -----
800
801 # Range() function in python
802 # Range() function in python a sequence of number from 0 to value minus 1.
803
804 print(range(4))
805

```

```

806 for i in range(3):
807     print(i)
808
809
810 flist=["Ram", "laxman", "bharat"]
811 print(range(len(flist)))
812 -----
813
814 # Normal old code what we have used before
815
816 friends = [ "Kashi", "Dipendra", "Surja", "Bijay"]
817 for friend in friends:
818     print("Happy new year", friend)
819
820 # Same code and same output using range() function
821
822 friends = [ "Kashi", "Dipendra", "Surja", "Bijay"]
823 for i in list(range(len(friends))):          # first find out len(friends) --4
824     and then use range(3) --0,1,2,3  and then make it a list [0, 1, 2,3]
825     print("Happy new year", friends[i])
826 -----
827
828 # Operations on lists
829 # Adding two lists
830 # Most of the operations in the list are same as the string
831 x = [ "hi", 1, 32, 5]
832 y = [ "Hello", 15, 20 ]
833
834 new_list= x + y      # the order of the entity in the list will be same as the order in
835 which list are added.
836 print(new_list)
837 -----
838
839 # Slicing of the list
840 # Remember as in string also, the last number in the slacing repersent upto but not
841 included
842 num_list = [ 1,5,7,9,6,36,84,66,25,94,22,16,546]
843
844 num_list[0:6]      # the number inside the slacing are talking about the position
845
846 num_list[:3]       # first three
847
848 num_list[5:]       # fifth positin entity till last
849
850 num_list[:]        # whole list
851 -----
852
853 # Just like there are a lot of library methods in string we have many library methods
854 for list too (eg append(), count(), len() etc)
855
856 demo_list = [ 1,5,7, "bharat",9,6,36, "sagar", "suraj", "hemant"]
857 dir(demo_list)      # these are the libraries you can use in the list
858
859 # we will try some of these libraries by building our own list from scratch
860 -----
861
862 # Building a list from scratch
863 # list maintain the order i.e the order you append the item, same order it will appear
864 in the list
865
866 check_list = list() # list is like empty pocket
867 check_list.append("tomatos")          # oder of append will decide the order of entity
868 in the list.
869
870 check_list.append("potatos")          # how many times you append sthg that will be in
871 the list that very time
872
873 check_list.append("chicken")          # since list are mutable, so every time we do
874 append our orginal list is being changed

```

```

860
861 print(check_list)
862
863 # quiz- can we append this this later to middle of the list? ASK THEM yes by insert()
864 -----
865
866 # IN to find items in the list
867
868 demo_list = [ 1,5,7, "bharat",9,6,36, "sagar", "suraj", "hemant"]
869
870 5 in demo_list
871
872 "sagar" in demo_list
873
874 15 in demo_list
875 -----
876
877 # Sorting in the list
878
879 friends = [ "Kashi", "Dipendra", "Surja", "Bijay"]
880 friends.sort()
881 print(friends)
882 -----
883
884 # Some built in function we can use in the list
885 # if you have the numeric list built-in function are far better to use then using loop
886
887 num_list = [ 1,5,7,9,6,36,84,66,25,94,22,16,546]
888
889 print(max(num_list))
890
891 print(min(num_list))
892
893 print(sum(num_list))
894
895 print(len(num_list))
896 -----
897
898 # Homework-10
899 # Write a program that pop up for the number to enter untill you enter done. and prints
900 # out the average of the entered numbers without using list and with using list.
901
902 total_sum = 0
903 total_count = 0
904 while True:
905     num = input("Enter a number:")
906     if num == "done":
907         break
908
909     num = float(num)
910     total_sum = total_sum + num
911     total_count = total_count + 1
912
913 print("The average is:", total_sum/total_count)
914
915 # Now trying the same code with list
916 num_list = list()
917
918 while True:
919     num = input("Enter a number:")
920     if num == "done":
921         break
922
923     num = float(num)
924     num_list.append(num)
925
926 Average= sum(num_list)/len(num_list)

```



```

922 print("The average is:", Average)
923 -----
924 # split() function in strings and list
925 # eg:- break a line(or string), split it and look for the each words in the line
926 # split() function can split the string on certain basis and return back us the list
927 # by default split() function split by the blank space
928
929 song = "give me some sunshine give me some rain"
930 song.split()          # this output is a list, where breaking cretria on the string
was blank space
931 -----
932 # more then one blank space will also be counted as single blank space while splitting
933
934 line = "A              lot of blank space we have"
935 line.split()
936 -----
937 # Other splitting cretria
938
939 demo_line = "A;what's;your;Name"
940 demo_line.split()      # no blank space means no splitting will happens,
single entity inside list
941
942 demo_line = "A;what's;your;Name"
943 demo_line.split(';')   # the splitting cretria is now ';', so splitting will
happens where where there is ';'
944 -----
945 # Homework-11
946 # Write a python program which ask for file to read, then then read the mbox-short.txt
file, get all the lines that starts from From cluse and check all those email
947 # and print out the date are are reviced on
948
949 file = input("Enter a file name:")
950 fhandle = open(file)
951
952 for line in fhandle:
953     line = line.strip()
954     if not line.startswith('From '):
955         continue
956
957     word_list= line.split()
958     word_list[2]
959
960 -----
961
962 # Homework-11 [Alternative good approach]
963 # Write a python program which ask for file to read, then then read the mbox-short.txt
file, get all the lines that starts from From cluse and check all those email
964 # and print out the date are are reviced on
965
966 file = input("Enter a file name:")
967 fhandle = open(file)
968
969 for line in fhandle:
970     line = line.rstrip()
971     if not line.startswith('From'):
972         continue
973
974     #print(line)
975     words=line.split()
976     if len(words) <= 2:
977         continue
978     #print(words)

```

```

979     print(words[2])
980     -----
981     # Homework-11 [alternative approach]
982     # Write a python program which ask for file to read, then then read the mbox-short.txt
983     # file, get all the lines that starts from From cluse and check all those email
984     # and print out the date are are reviced on
985     file = input("Enter a file name:")
986     fhandle = open(file)
987
988     for line in fhandle:
989         line = line.rstrip()
990         words=line.split()
991         #print(words)
992         if len(words) <= 2 or words[0] != "From":
993             continue
994         #print(words)
995         print(words[2])
996     -----
997     # Homework-12
998     # Redo the old code (Homework-7) for getting the university name from the line using
999     # split() function
1000     line= "From Augustin.Marcus@siue.edu Jan 4 2024 06:32PM"
1001
1002     words = line.split()
1003     sub_words = words[1].split('.')
1004     sub_words[1].split('@')[1]
1005
1006     -----
1007     # Dictionaries in Python
1008     # Dictionaries in python are denoted by {} inside which there is a key-value pair
1009     # Secodn type of data structure
1010     # difference between list and dictionary is: List is a linear collection of values that
1011     # stay in order while dictionary is a beg of value each with its own label
1012     # In list index is position but in dictionary index is key (of key-value pair)
1013     # so Dictionary is a key value pair.
1014     # let's look the template code of dictionary
1015
1016     purse = dict()
1017
1018     purse['Key'] = 'value-1'
1019
1020     print(purse)
1021     -----
1022     # quiz- Make a dictionary that has subject as the key and their subject code as a value
1023
1024     Statistics = dict()
1025
1026     Statistics['Machine learning'] = 'Stat 561'
1027     Statistics['Data Science'] = 'Stat 560'
1028     Statistics['SQL an Oracal'] = 'CMIS 563'
1029
1030     # ordering of the items in the dictionary does not have any pattern but key value pair
1031     # always comes together.
1032     print(Statistics)
1033
1034     # accessing the dictionary
1035
1036     Statistics['Machine learning']
1037     -----
1038     -----

```

```

1037 # Another easy way to create a dictionary
1038
1039 demDict = {'Stat': 561, 'OR': 585, 'QR': 101, 'CMIS': 563}
1040
1041 print(demDict)
1042
1043 print(type(demDict))
1044
1045 -----
1046 # directory is not mutable same as string, i.e original dictionary will not change if you
1047 # apply any operation on it
1048 # to make the change in the original dictionary you need to save the change to the
1049 # original dictionary
1050
1051 Statistics = dict()
1052
1053 Statistics['Machine learning'] = 'Stat 561'
1054 Statistics['Data Science'] = 'Stat 560'
1055 Statistics['SQL an Oracal'] = 'CMIS 563'
1056
1057 Statistics['Machine learning'] + ' and Stat 562' # this will not change our orgianl
1058 dictionary
1059
1060 print('Without Saving:-',Statistics)
1061
1062 # but this bellow code will change the dictionary as we are saving the change made in
1063 # the dictionary too
1064
1065 Statistics['Machine learning'] = Statistics['Machine learning'] + ' and Stat 562'
1066
1067 print('With Saving:-',Statistics)
1068
1069 -----
1070 # quiz- IF you are given a list of names, how do you know which one is the highest
1071 # repeted? ---by drawing histogram or by counting each time we saw same name.
1072 # we can do that using the python dictionary, by updating the value of the name (key)
1073 # each time we encounter.
1074
1075 friends = ["Kashi", "Dipendra", "Surja", "Bijay", "Kashi", "Dipendra","Dipendra", "Surja"
1076 , "Surja", "Bijay", "Kashi", "Dipendra"]
1077
1078 fdictionary = dict()
1079
1080 fdictionary["Kashi"] = 0
1081 fdictionary["Dipendra"] = 0
1082 fdictionary["Surja"] = 0
1083 fdictionary["Bijay"] = 0
1084
1085 for key in friends:
1086     #print(key)
1087     if key == "Kashi":
1088         fdictionary["Kashi"] = fdictionary["Kashi"] + 1
1089     elif key == "Dipendra":
1090         fdictionary["Dipendra"] = fdictionary["Dipendra"] + 1
1091     elif key == "Surja":
1092         fdictionary["Surja"] = fdictionary["Surja"] + 1
1093     else:
1094         fdictionary["Bijay"] = fdictionary["Bijay"] + 1
1095
1096 fdictionary
1097
1098 -----
1099 # looking if something is inside the dictionary or not
1100 # the search is done on the basis of key not value
1101
1102 demo_dict= dict()

```

```

1094 demo_dict["STAT"] = 561
1095 demo_dict["CMIS"] = 563
1096 demo_dict["OR"] = 585
1097
1098 demo_dict
1099
1100 print(type(demo_dict))
1101
1102 'CMIS' in demo_dict
1103
1104 'Stat' in demo_dict
1105
1106 'QR' in demo_dict
1107
1108 # this is very important because, with help of this we can do sthg in the dictionary, if
1109 # key is there or do something else
1110 -----
1111 -----
1112 # using the above key in the dictionary or not concept, we will count the names of the
1113 # friends that how many times they appears
1114
1115 friends = ["Kashi", "Dipendra", "Surja", "Bijay", "Kashi", "Dipendra", "Dipendra", "Surja",
1116            ",Surja", "Bijay", "Kashi", "Dipendra"]
1117
1118 fdictionary = dict()
1119
1120 for name in friends:
1121     if name not in fdictionary:
1122         fdictionary[name] = 1
1123     else :
1124         fdictionary[name] = fdictionary[name] + 1
1125
1126 print(fdictionary)
1127 -----
1128 -----
1129 # if there is no key in the dictionary, put it in. if there is key in the dictionary do
1130 # sthg to the existing value that are already there.
1131 # This is something we will do a lot in the dictionary, we have to do this so many
1132 # times such that they have developed function for this
1133 # the get() Methods for dictionary
1134 # the pattern of checking to see if a key is already in the dictionary and assuming a
1135 # default value if the key is not there is so common that there is a method
1136 # called get() that does this for us.
1137
1138 # eg. x = counts.get(name, 0) :-- here counts is the dictionary on which we have used
1139 # the get function which look for the key as a name and 0 is the default value
1140 # that mean if key does not exist, it will have value 0 for it
1141 -----
1142 -----
1143 # Using this get() function concept to solve the above problem of counting the name of
1144 # the friends how many times they appears
1145
1146 friends = ["Kashi", "Dipendra", "Surja", "Bijay", "Kashi", "Dipendra", "Dipendra", "Surja",
1147            ",Surja", "Bijay", "Kashi", "Dipendra"]
1148
1149 counts = dict()
1150 for name in friends:
1151     counts[name] = counts.get(name, 0) + 1
1152 print(counts)
1153 -----
1154 -----
1155 # Homework-13
1156 # write a python code to use all above concepts to find all the words from the mbox.txt
1157 # file and also give their count that how many times they appear?
1158
1159 file = input("Enter a file")

```

```

1147 fhandle = open(file)
1148 counts = dict()
1149
1150 for line in fhandle:
1151     line = line.strip()
1152     words = line.split()
1153     #print(words)
1154     for word in words:
1155         counts[word] = counts.get(word, 0) + 1
1156
1157 print(counts)
1158 -----
1159
1160 # for loops in dictionary
1161 counts = { 'Sagar' : 100, 'Kashi' : 55, 'Dipendra' : 32, 'Ram' : 105 }
1162
1163 for key in counts:                                     # for making my
1164     life easier I use the word key, it is not necessary
1165     print('the key value pairs are: ', key , '--', counts[key])
1166 -----
1167
1168 # getting list of key's from the dictionary
1169 counts = { 'Sagar' : 100, 'Kashi' : 55, 'Dipendra' : 32, 'Ram' : 105 }
1170
1171 print(list(counts))
1172
1173 print(counts.keys())
1174
1175 print(counts.values())
1176
1177 print(counts.items())    # tuple: we will talk about this a lot in next chapter,
1178                          # the output of this will be the key-value pair as (key,value)
1179                          # inside list
1180 -----
1181
1182 # items() bonus trick
1183 # Bonus: Two iteration variables in python, no other programming language can handle this
1184 counts = { 'Sagar' : 100, 'Kashi' : 55, 'Dipendra' : 32, 'Ram' : 105 }
1185
1186 for key, value in counts.items():    # note here key is the first iteration variable
1187     and value is the second iteration variable
1188     print(key, value)
1189 -----
1190
1191 # Homework-14
1192 # write a python code to use all above concepts to find all the words from the mbox.txt
1193 # file and also give their count that how many times they appear?
1194 # modify this homework-13 to give only the word which is highest frequent and also give
1195 # its number.
1196
1197 file = input("Enter a file name:")
1198 fhandle = open(file)
1199
1200 counts = dict()
1201 for line in fhandle:
1202     line = line.strip()
1203     words = line.split()
1204     #print(words)
1205
1206     for word in words:
1207         counts[word] = counts.get(word, 0) + 1

```

```

1205 #print(counts)
1206 bigword = None
1207 bigcount = None
1208
1209 for key, value in counts.items():
1210     #print(key, value)
1211     if bigcount is None or value > bigcount:
1212         bigword = key
1213         bigcount = value
1214 print(bigword, bigcount)
1215 -----
1216
1217 # Cross-checking the above code with string (single) line
1218 fhandle = "hello hello what is is name is your name what is is"
1219
1220 counts = dict()
1221 for line in fhandle:
1222     line = line.strip()
1223     words = line.split()
1224     #print(words)
1225
1226     for word in words:
1227         counts[word] = counts.get(word, 0) + 1
1228
1229 #print(counts)
1230 bigword = None
1231 bigcount = None
1232
1233 for key, value in counts.items():
1234     #print(key, value)
1235     if bigcount is None or value > bigcount:
1236         bigword = key
1237         bigcount = value
1238 print(bigword, bigcount)
1239 -----
1240
1241 # showing how program is actually working with a easy example
1242
1243 file = input('Enter a file Name: ')
1244 fhandle = open(file)
1245
1246 count = dict()
1247 for line in fhandle:
1248     line = line.strip()
1249     print(line)
1250     words = line.split()
1251     print(words)
1252     for word in words:
1253         if word in count:
1254             print(word)
1255             print('**Existing word**')
1256             print(count[word])
1257             count[word] = count[word] + 1
1258         else:
1259             print(word)
1260             print('**new word**')
1261             count[word] = 1
1262             print(count[word])
1263 print(count)
1264 -----
1265
1266 # tuple in python
1267 # tuples are like list, but here we use () parenthesis instead of [] squarebracket, note
1268 # formation wise only
1269 # index in tuple is postition and position starts from zero, all same as list
1270 # the only difference is they are not mutable

```

```

1268
1269 demo_tuple = ('Ram', 'Krishna', 'bishnu', 'Hanuman', 'Mahadev', 'kali')
1270
1271 demo_tuple[1]
1272 -----
1273
1273 # quiz- what does tuple are not mutable means
1274
1275 x_list = [ 5, 6, 32, 9, 0]
1276
1277 x_list[1] = 69
1278 print(x_list) # look list is changed
1279
1280 y_tuple = ( 5, 6, 32, 9, 0)
1281
1282 y_tuple[1] = 69 # traceback error: item assignment does not support
1283 -----
1284
1284 # there are lot of things that you can not do in tuple like
1285 # 1) you can not short the tuple
1286 # 2) you can not append into the tuple
1287 # 3) you can not reverse the tuple
1288
1289 # for list you can do
1290 x_list = [ 5, 6, 32, 9, 0]
1291 x_list[1] = 69
1292 x_list.append(5000)
1293 x_list.sort(reverse=True)
1294 x_list.reverse() # if you want to do descending
1295 print(x_list) # look list is changed
1296
1297
1298 # for tuple you can not do
1299 y_tuple = ( 5, 6, 32, 9, 0)
1300 y_tuple[1] = 69 # traceback error: item assignment does not support
1301 y_tuple.append('sagar')
1302 y_tuple.sort()
1303 y_tuple.reverse()
1304 -----
1305
1305 # so the question is what we can do with tuple
1306
1307 t = tuple()
1308 dir(t)
1309 -----
1310
1310 # comparing waht we can do with list vs what we can do with tuple
1311
1312 l = list()
1313 print(dir(l))
1314
1315 t = tuple()
1316 print(dir(t))
1317 -----
1318
1318 # tuple for assignment or assignment in tuple
1319
1320 (x, y) = ( 1, 100)
1321 print(x)
1322 print(y)
1323
1324 (first_name, last_name) = ( 'sagar' , 'kalauni')
1325 print(first_name)
1326 print(last_name)
1327 -----
1328
1328 # tuples from dictionary

```

```

1329 # use items() function to find list of the tuples from the dictionary
1330
1331 demoDic = { 'stat' : 560, 'CMIS': 563, 'OR': 585 }
1332
1333 print(type(demoDic))
1334 demoDic.items()
1335 -----
1336
1337 # Comprasion in tuple
1338
1339 (1, 5, 7) < (2, -5 -10) # it will only look for the most significant digit in the
1340 tuple, first one is most significant, similar as comparing numbers
1341
1342 (1, 5, 7) < (1, -5 -10) # if the first one is same it will look for the second and so
1343 on
1344
1345 ('ram' , 'sita', 'hanuman') < ( 'hanuman', 'sita', 'ram')
1346
1347 ('ram' , 'sita', 'hanuman') < ( 'ram', 'hanuman', 'sita')
1348 -----
1349
1350 # shorting the list of tuples using the sorted() function
1351 # we can obtain the list of tuple by using the items() function in the dictionary
1352 # sorting is done on the basis of key
1353
1354 demo_dict = { 'Stat' : 560, 'CMIS': 563, 'OR': 585 }
1355
1356 tuple_list = demo_dict.items()
1357
1358 # sorting the list o tuples.
1359
1360 sorted(tuple_list, reverse = True)
1361 -----
1362
1363 # using sorted() and loop to print the key value pair in certain order
1364 # sorting is done on the basis of key
1365
1366 demo_dict = { 'Stat' : 560, 'CMIS': 563, 'OR': 585 }
1367
1368 tuple_list = demo_dict.items()
1369
1370 sorted(tuple_list)
1371
1372 for key, value in sorted(tuple_list):
1373     print (key, value)
1374
1375 # same code in descending order
1376
1377 demo_dict = { 'Stat' : 560, 'CMIS': 563, 'OR': 585 }
1378
1379 tuple_list = demo_dict.items()
1380
1381 sorted(tuple_list, reverse = True)
1382
1383 for key, value in sorted(tuple_list, reverse = True):
1384     print (key, value)
1385
1386 # always remember shorting by default is done on the basis of key
1387 -----
1388
1389 # quiz- how to do the sorting on the basis of value of the tuple.
1390 # idea is same that by default sorting is done on the basis of key of the tuple, so why
1391 not change the order of our tuple to make key as value and value as key
1392 # and then do sorting. those changed order tuples should be put in the new empty list
1393
1394 demo_dict = { 'a' : 560, 'c': 563, 'd': 585, 'z' : 99 }
1395

```



```

1389 temp_list = list()
1390 tuple_list = demo_dict.items()
1391
1392 for key, value in tuple_list:
1393     temp_list.append((value, key))
1394 #print(temp_list)
1395     for key, value in sorted(temp_list):
1396         print(key, value)
1397 -----
1398 # quiz- how to do the sorting on the basis of value of the tuple.
1399 # idea is same that by default sorting is done on the basis of key of the tuple, so why
1400 # not change the order of our tuple to make key as value and value as key
1401 # and and put them on one empty list then do sorting.
1402 demo_dict = { 'a' : 560, 'c': 563, 'd': 585, 'z' : 99 }
1403
1404 temp_list = list()
1405 tuple_list = demo_dict.items()
1406
1407 for key, value in tuple_list:
1408     temp_list.append((value, key))
1409 #print(temp_list)
1410 for key, value in sorted(temp_list):
1411     print(key, value)
1412 -----
1413 # Homework-15
1414 # Using all above concepts, find out the top 10 most used words in mbox-shor.txt file
1415 # with standard coding format.
1416
1417 file = input("Enter a file: ")
1418 fhandle = open(file)
1419
1420 count = dict()
1421 for line in fhandle:
1422     line = line.strip()
1423     words = line.split()
1424
1425     for word in words:
1426         count[word] = count.get(word, 0) + 1
1427 #print(count)
1428 temp_list = list()
1429 tuple_list = count.items()
1430
1431 for key, value in tuple_list:
1432     temp_list.append((value, key))
1433 #print(temp_list)
1434
1435 for key, value in sorted(temp_list, reverse = True)[:10]: # just to limit 10 items from
1436     the list
1437     print(key, value)
1438 -----
1439 # quiz- Can you do the same above coding of finding the sorted list of highly frequented
1440 # data from this dictionary: demo_dict = {'a': 10, 'b':2, 'c':99}
1441 # doing it with even shorter version, not necessary to understand right now, just
1442 # showing you when you are comfortable with programming how it can be too short also
1443 # List comprasion
1444
1445 demo_dict = {'a': 10, 'b':2, 'c':99, 'd':11, 'z':81}
1446
1447 print( sorted([(v, k) for k, v in demo_dict.items()], reverse= True)[0:3])
1448 -----
1449 ~~~~~Chater-3 Accessing web
1450 data through python~~~~~

```

```

1446 # Regular Expression in python
1447 # Regular expression is something you can skip also
1448 # Regular expression quick guide
1449 ^ match the begining of the line
1450 $ match the end of the line
1451 . match any character
1452 \s match whitespace
1453 \S match any non-whitespace character
1454 * repetes a character zero or more times
1455 *? repetes a character zero or more times (non greedy)
1456 + repetes a character one or more times
1457 +? repetes a character one or more times (non greedy)
1458 ...
1459 etc
1460 -----
1461 -----
1462 # Regular expression is not built in python, so we kind of import it in the working
1463 # re.search() ---> same as the find function in the string [Used for finding the match]
1464 # re.findall() ---> same as the combination of find and slicing in the string [Used for
1465 # extracting]
1466
1467 # using find()
1468 file = input("Enter a file name: ")
1469 fhandle = open(file)
1470
1471 for line in fhandle:
1472     line = line.strip()
1473     if line.find('From:') >=0:
1474         print(line)
1475
1476 # using regular expression search
1477 import re
1478
1479 file = input("Enter a file name: ")
1480 fhandle = open(file)
1481
1482 for line in fhandle:
1483     line = line.strip()
1484     if re.search('From:', line):
1485         print(line)
1486
1487 -----
1488 -----
1489 # if we want our first character to be certain to startwith, we can using the regular
1490 # expression
1491 # as we used to use startswith() function for string, here we will use ^ before the
1492 # thing we should search for indicating that this should be the first character.
1493
1494 # using startswith()
1495 file = input("Enter a file name: ")
1496 fhandle = open(file)
1497
1498 for line in fhandle:
1499     line = line.strip()
1500     if line.startswith('From:'):
1501         print(line)
1502
1503 # using regular expression search
1504 import re
1505
1506 file = input("Enter a file name: ")
1507 fhandle = open(file)
1508
1509 for line in fhandle:
1510     line = line.strip()
1511     if re.search('^From:', line):
1512         print(line)

```

```

1507 -----
1508 # wild card characters
1509 # ^X.*: ---> looking for lines, (^) X at the begning, (.) match any character, (*) any
    number of character. [with no condition at the end]
1510
1511 # using regular expression search
1512 import re
1513
1514 file = input("Enter a file name: ")
1515 fhandle = open(file)
1516
1517 for line in fhandle:
1518     line = line.strip()
1519     if re.search('^X.*:', line):
1520         print(line)
1521 -----
1522
1523 # Being more preicse
1524 # ^X-\S+: ---> looking for lines, (^) X at the begning, (.) match any character, (*) any
    number of character,
1525
1526 # using regular expression search
1527 import re
1528
1529 file = input("Enter a file name: ")
1530 fhandle = open(file)
1531
1532 for line in fhandle:
1533     line = line.strip()
1534     if re.search('^X-\S+:', line):
1535         print(line)
1536 -----
1537
1538 # Now extracting the data with the help of Regular expression
1539 # anything inside squre bracket is one character to look for but it can inside have
    anything like range, list etc
1540 # so [0-9]+, this means look for any one digit from 0-9, by saying + it means it can
    have more character too. so look for one or more digit
1541
1542 import re
1543 x = 'Hell my name is Sagar. My age is 26. I am right now in 4 th semester of my masters.
    I have courses stat 579, stat 561 this semester'
1544
1545 y = re.findall('[0-9]+', x)
1546 print(y)
1547 -----
1548
1549 # looking for how many vowels are in the given string (line)
1550 import re
1551 x = 'Hell my name is Sagar. My age is 26. I am right now in 4 th semester of my masters.
    I have courses stat 579, stat 561 this semester '
1552
1553 y = re.findall('[aeiou]', x)
1554 print(y)
1555
1556 count = dict()
1557 for letter in y:
1558     count[letter] = count.get(letter, 0) +1
1559 print(count)
1560 -----
1561
1562 # Homework-16
1563 # Use this code to find the number in the each line of mbox-short.txt
1564 import re

```

```

1563 file = input("Enter a file name: ")
1564 fhandle = open(file)
1565
1566 for line in fhandle:
1567     line = line.strip()
1568     y= re.findall('[0-9]+', line)
1569     print(y)
1570 -----
1571
1572 # greedy matching in the expression
1573 import re
1574 x = 'From: using the:'
1575
1576 y=re.search('^F.+:', x)
1577 print(y)
1578
1579 # how to remove the gredyness of the expression
1580 # greedy matching in the expression is removed by adding ?
1581 import re
1582 x = 'From: using the:'
1583
1584 y=re.search('^F.+?:', x) # (.)any character, (+) one or more time, (?) but don't be
greedy
1585 print(y)
1586 -----
1587
1588 # Homework-17
1589 # find all the email address provided in the mbox.txt file using expression. hint: email
has non-blank character followed by @ followed by non-blank character.
1590
1591 file = input('Enter a file: ')
1592 fhandle = open(file)
1593
1594 for line in fhandle:
1595     line = line.strip()
1596     y = re.findall('\S+@\S+', line)
1597     if y == []:
1598         continue
1599     print(y)
1600 -----
1601
1602 # for more exact
1603 # find all the email address provided in the mbox.txt file using expression. hint: email
has non-blank character followed by @ followed by non-blank character.
1604
1605 import re
1606 file = input('Enter a file: ')
1607 fhandle = open(file)
1608
1609 for line in fhandle:
1610     line = line.strip()
1611     y = re.findall('^From (\S+@\S+)', line) # what i am saying inside extration, go look
word starting with From but I dont want that, after that there should be
1612     if y == []: # space and the thing to be extracted will
be inside the parenthesis which is non-blank character followed by some
1613         continue # some character followed by @ sign, again
followed by non-blank character and other characters
1614     print(y)
1615 -----
1616
1617 # Recall Coding problem
1618 # Recalling the university or company name from the email address
1619 # Method-1 (Already discussed)
1620
1621 line= "From Augustin.Marcus@siue.edu Jan 4 2024 06:32PM"
1622

```

```

1620 Spoint = line.find('@')
1621
1622 Epoint = line.find('.', Spoint)
1623
1624 line[Spoint + 1 : Epoint]
1625
1626 # Method-2 (already discussed)
1627
1628 line= "From Augustin.Marcus@siue.edu Jan 4 2024 06:32PM"
1629
1630 new_line = line.split()[1]
1631
1632 new_line_1 = new_line.split('.')[1]
1633
1634 print(new_line_1.split('@')[1])
1635
1636 # Method-3
1637
1638 import re
1639 line= "From Augustin.Marcus@siue.edu Jan 4 2024 06:32PM"
1640
1641 y=re.search('@([^.]+)', line).group(1)
1642 print(y)
1643 -----
1644
1645 # Homework-18
1646 # Extracting data from the mbox-short.txt file
1647 # write a python code to extract the float number in each line after the word
1648 # X-DSPAM-Confidence. finally give the list of all the float numbers
1649
1650 file = input('Enter a file: ')
1651 fhandle = open(file)
1652
1653 for line in fhandle:
1654     line = line.strip()
1655     word = re.findall('^X-DSPAM-Confidence [0-9].*', line)
1656     print(word)
1657 -----
1658
1659 # Homework-18
1660 # Extracting data from the mbox-short.txt file
1661 # write a python code to extract the float number in each line after the word
1662 # X-DSPAM-Confidence. finally give the list of all the float numbers
1663
1664 import re
1665 file = input('Enter a file: ')
1666 fhandle = open(file)
1667
1668 num_list = list()
1669 for line in fhandle:
1670     line = line.strip()
1671     num = re.findall('^X-DSPAM-Confidence: ([0-9.]+)', line) # look for the line having
1672     # X-DSPAM-Confidence, start extracting after the space after X-DSPAM-Confidence
1673     if num == []: # look for any one
1674         # character from 0-9 and ., (+) with one or more character
1675         continue
1676     else:
1677         num_list.append(num[0])
1678 print(num_list)
1679
1680 # if you want the maximum and minimum of this
1681 print(max(num_list))
1682 print(min(num_list))
1683 -----
1684
1685 # same question little different approach
1686 import re

```

```

1680 file = input('Enter a file: ')
1681 fhandle = open(file)
1682
1683 num_list = list()
1684 for line in fhandle:
1685     line = line.strip()
1686     num = re.findall('^X-DSPAM-Confidence: ([0-9.]*)', line)
1687     if len(num) != 1:
1688         continue
1689     else:
1690         num_list.append(num[0])
1691 print(num_list)
1692
1693 # if you want the maximum and minimum of this
1694 print(max(num_list))
1695 print(min(num_list))
1696 -----
1697 # What if you have to extract some symbol like ($) which have a special meaning in the
1698 # expression. use \ (back-slash) in front of it
1699
1700 line = 'Hey my name is sagar and I work as a teaching assistant in siue and they pay me
1701 $1060 as stipend monthly'
1702
1703 import re
1704
1705 my_salary = re.findall('\$+', line)
1706 print(my_salary)
1707 -----
1708 # Quick review of the expression and their use in python
1709
1710 ^    Matches the beginning of a line
1711 $    Matches the end of the line
1712 .    Matches any character
1713 \s   Matches whitespace
1714 \S   Matches any non-whitespace character
1715 *    Repeats a character zero or more times
1716 *?   Repeats a character zero or more times (non-greedy)
1717 +    Repeats a character one or more times
1718 +?   Repeats a character one or more times (non-greedy)
1719
1720 [aeiou]    Matches a single character in the listed set
1721 [^XYZ]     Matches a single character not in the listed set
1722 [a-z0-9]   The set of characters can include a range
1723 (         Indicates where string extraction is to start
1724 )         Indicates where string extraction is to end
1725 -----
1726 # Now onward let's talk to the internet with the help of python
1727
1728 # TCP connections/ Sockets
1729 # In a computer networking, an internet socket or network socket is an endpoint of
1730 # bidirectional inter-process communication flow across an internet protocol-based
1731 # computer network, such as the internet.
1732 # process <--- internet ---> process
1733 # web server ko lagi port ho 80
1734 # the protocol we are going to talk about is Http
1735 # url = protocol + host + document. eg http://www.dr.chunk.com/page1-htm.
1736 -----
1737
1738 # General Rule of how it works
1739 # Create a socket between your application (python code) and web server
1740 # once socket is created, connect it to the web server by giving the host and port.
1741 # after that we need to have our first communication, and first communication is always
1742 # done from our side [as a rule of http or port 80]

```

```

1739 # so write a command and send the command through the socket.
1740 # after that web-server will look at the command and give back you the data on the basis
    of what command you have sent
1741 # revive the data from the web server through socket.
1742
1743 import socket
1744
1745 mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # think of socket as file
    handle that does not have data
1746 mysock.connect(('data.pr4e.org',80)) # connect is the function
    which take single tuple as the input
1747 cmd = 'GET http://data.pr4e.org/romeo.txt HTTP/1.0\r\n\r\n'.encode() # we are sending
    as utf-8 data
1748 mysock.send(cmd)
1749
1750 while True:
1751     data = mysock.recv(512)
1752     if (len(data) < 1):
1753         break
1754     print(data.decode()) # we will recive as utf-8 data, so need to
        decode
1755 mysock.close()
1756 -----
1757 # the thing that comes at the end of the url is called the get parameter
1758 # http://data.pr4e.org/romeo.txt/guess=1 , here guess=1 is called the get parameter
1759 # status code = 200 means you are good we found the web page for that
1760 # status code = 404 means error not found
1761 # status code = 302 means wrong web browser but will be directed to the correct one
1762 -----
1763 # Since computer does not understand the letters, so we need to have the standard
    conversion for computer and one of them was ASCII (American standard code for
1764 # information interchange)
1765 # way to look at the number crossponding to letters
1766
1767 print(ord('H'))
1768 print(ord('A'))
1769
1770 print(ord('\n')) #remember new line is a single character
1771 -----
1772 # in the old times, american computer can talk to american computer only and japnease
    computer can talk to japnease computer only as they have their own stardard
1773 # but it was the problem to communicate between the computer in japan to computer in
    america. So they have introduced the concept of unicode. hence all the input
1774 # you will give to the computer will be converted to some encode which is same all over
    the world and computer will give you output in the same encode version
1775 # which need to be decoded to understand.
1776 # basic concept is we are sending our command as bite by encoding it and computer is
    send back data in the form of bite and we are decoding it back to the string.
1777 -----
1778 # Retriving the data from the web.
1779 # since every time we have to do the same stuffs of creating socket, connecting to
    webserver, send request command as a bite, reciving a data as a bite. Why not
1780 # make our life easy by defining a function or libraries that do this for us. yes are
    are some libraries for doing all this and making coding all easy in python
1781 # the libraries will parse the url, figure out what server to talk with, what document
    to retrieve, waht http version are are inside the library. it will simply
1782 # open the url and give the fhandle as we used to get in the old cases from our own
    computer, means it is almost same as open()
1783
1784 import urllib.request, urllib.parse, urllib.error
1785
1786 fhandle = urllib.request.urlopen('http://data.pr4e.org/romeo.txt')
1787 for line in fhandle:

```

```

1788     print(line.decode().strip())          # this line you get is bite array, so may be
1789     need to decode to string
1789 -----
1790 # comprasion to get same work done with and with out using the library
1791
1792 # without library
1793 import socket
1794
1795 mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # think of socket as file
1796 mysock.connect(('data.pr4e.org',80)) # connect is the function
1797 which take single tuple as the input
1797 cmd = 'GET http://data.pr4e.org/romeo.txt HTTP/1.0\r\n\r\n'.encode() # we are sending
1798 as utf-8 data
1798 mysock.send(cmd)
1799
1800 while True:
1801     data = mysock.recv(512)
1802     if (len(data) < 1):
1803         break
1804     print(data.decode()) # we will recive as utf-8 data, so need to
1805     decode
1805 mysock.close()
1806
1807 # With library [quite easy and stright forward]
1808
1809 import urllib.request, urllib.parse, urllib.error
1810
1811 fhandle = urllib.request.urlopen('http://data.pr4e.org/romeo.txt')
1812 for line in fhandle:
1813     #line = line.strip()
1814     print(line.decode().strip()) # this line you get is bite array, so may
1815     be need to decode to string
1816
1817 # with this 4 line of code we are actually reading the web page, that's the power of
1818 python
1819 # Now with these ideas, it is no longer a web page for us, it is simply like a file in
1819 our computer whcih we can open and do what ever calculation we want to
1818 -----
1819 # Homework-19
1820 # access the web page http://data.pr4e.org/romeo.txt and look for the words on it and
1821 count how many word are repeted how many times? [Same as old questions]
1822
1822 import urllib.request, urllib.parse, urllib.error
1823
1824 fhandle = urllib.request.urlopen('http://data.pr4e.org/romeo.txt')
1825
1826 count = dict()
1827 for line in fhandle:
1828     line = line.decode().strip()
1829     #print(line)
1830     word_list = line.split()
1831     for word in word_list:
1832         count[word] = count.get(word, 0) + 1
1833 print(count)
1834 -----
1835 # Homework-20
1836 # access teh web page http://data.pr4e.org/romeo.txt and find out the most frequent word
1837 and also give us the frequency of the word in the web page.
1838
1838 import urllib.request, urllib.parse, urllib.error
1839
1840 fhandle = urllib.request.urlopen('http://data.pr4e.org/romeo.txt')
1841

```



```

1842 count = dict()
1843 for line in fhandle:
1844     line = line.decode().strip()
1845     word_list = line.split()
1846     for word in word_list:
1847         count[word] = count.get(word, 0) + 1
1848 #print(count)
1849
1850 emp_list = list()
1851 for k,v in count.items():
1852     emp_list.append((v,k))
1853
1854 print(emp_list)
1855 print(max(emp_list)) # so the is one of the most repeted words in the above web page if
1856                      # frist seen and secodn tuple with same max value, so compared by
1857                      # looking the second cordinate
1858 -----
1859 # quiz- what do you think this web scarping can be used for. does this make sense to you
1860 # that right now we can look for one certain web page, get the link of
1861 # other page inside that page and follow the link and go again and again, This could be
1862 # one of your homework-13
1863
1864 import urllib.request, urllib.parse, urllib.error
1865
1866 fhandle = urllib.request.urlopen('http://www.dr-chunk.com/page1-htm')
1867
1868 for line in fhandle:
1869     line = line.decode().strip()
1870     print(line)
1871 -----
1872
1873 # What is web scarping?
1874 # when a program or script pretends to be a browser and retrives web pages, looks at
1875 # those web pages, extract information and then look at more web pages.
1876 # search engine scrape web pages, we call this 'spidering the web' or 'web crawling'
1877 # So in simple web scarping means look at the web pages for some links and look at those
1878 # links for some other links to get final answers
1879 -----
1880
1881 # if it is a text file to be retrieve from the web page, we did it and was nice but if it
1882 # was the html or xml file they are poorly managed so need special library
1883 # instaled to parse it and make it more informative called beautiful soup
1884
1885 # Beautiful Soup is a Python library that is used for web scraping purposes to pull the
1886 # data out of HTML and XML files. It provides Pythonic idioms for iterating,
1887 # searching, and modifying the parse tree, making it easy to scrape information from web
1888 # pages.
1889
1890 # you can think of this way also, we you retrieve html file usig python code and it has
1891 # links insde it with href:'url of link', one way to get this link is using
1892 # expression in pythons and that is kind of hard, beautiful soup can do this with all
1893 # one line of code and make our life easier.
1894
1895 # General syntax for data retrival using Beautiful soup
1896
1897 import urllib.request, urllib.parse, urllib.error
1898 from bs4 import BeautifulSoup
1899
1900 url = input ('Enter your url: ')
1901 html = urllib.request.urlopen(url).read() # because data we retrieve by html file is
1902      # messy, so we name it as html
1903 Soup = BeautifulSoup(html, 'html.parser') # by taking that nasty html data it will
1904      # parse it into nice tree like object, (we don't need to know what's inside)
1905

```

```

1893 # Retriving all of the anchor tags
1894 tags = Soup('a')
1895 for tag in tags:
1896     print(tag.get('href', None))
1897 -----
1898
1899 # installing and importing beautifulsoup in the jupiter notebook
1900 !pip install beautifulsoup4
1901 from bs4 import BeautifulSoup
1902 -----
1903
1904 import urllib.request, urllib.parse, urllib.error
1905 from bs4 import BeautifulSoup
1906 import ssl
1907
1908 # ignore ssl certificate error, [you don't need to know it exactly]
1909 ctx = ssl.create_default_context()
1910 ctx.check_hostname = False
1911 ctx.verify_mode = ssl.CERT_NONE
1912
1913 url = input('Enter your url:')
1914 html = urllib.request.urlopen(url, context = ctx).read() #it returns entrie docomnet
1915 # in the web page in a single big string with new line at the end of each line
1916 soup = BeautifulSoup(html, 'html.parser')
1917
1918 # retriving all the anchor tags, anchor tag means any things that starts with <a ... to >
1919 tags = soup('a') # it will give list of anchor tags
1920 for tag in tags:
1921     print(tag.get("href", None))
1922 -----
1923
1924 # till now we have studied how to communicate between two computer which are completely
1925 # distenct using socket and then to make our life easier we do have python
1926 # library to do all those line of code in single line.
1927 # now when we are sending our command or data from our computer using python or java
1928 # any, we are not actually sending python code we are sending the data which
1929 # is first serialize from our computer to particular format and then it is being to to
1930 # another computer which may be using java where the data is de-serialize in
1931 # that particular format.
1932
1933 # With the http request/response well understood and well supported, there was a natural
1934 # move towards exchanging data between programs using those protocols.
1935 # we needed to come up with an agreed way to repersent data going between applications
1936 # and across networks. There are two commonly used format: XML and JSON
1937 -----
1938
1939 # XML (extensible mark up language)
1940 # in rough word, two program agree on one syntex to share there data across the network
1941 # primary purpose is to help information systems share structured data
1942 # XML BASICS
1943
1944 <person>                                ---start tag
1945     <name>Sagar</name>
1946     <phone type = "intl">
1947         +1 (618) 917-9128                ----text content
1948     </phone>
1949     <email hide="yes" />                ----self closing tag
1950 </person>                                ----end tag
1951 -----
1952
1953 # XML Schema
1954 # So we have two cooperating applications, and they've got to send data to one another,
1955 # and they have a disagreement as to whether or not the data is right.
1956 # One side might blow up or the other side might blow up and it's like whose fault is
1957 # it? To stop this happen XML has its own rule called Schema of XML.
1958 -----

```

```

1947 # How to parse XML in python
1948 # let's create a general syntax for parsing XML and extracting information from it
1949
1950 # again to make things easier for us to do we do have a library to work with it
1951 import xml.etree.ElementTree as ET
1952 # normally we would be reading all of these data with urllib and read and whatever and
1953 # then we would parse it. But just to make these simple on one screen,
1954 # I've kept it simple. And so I have a string. also triple-quoted string in Python is a
1955 # potentially multi-line string.
1956 data = '''<person>
1957     <name>Sagar</name>
1958     <phone type="intl">
1959         +1 (618) 917-9128
1960     </phone>
1961     <email hide="yes" />
1962 </person>'''
1963
1964 # And what fromstring says is take this string and give us back basically a nice tree.
1965 # if you have syntax error in your data(or sting ) this could blow up
1966 tree = ET.fromstring(data)
1967 print('name:', tree.find('name').text) # with in that formed tree go find me the
1968 # tag named- name and give the text part of it
1969 print('attribute', tree.find('email').get('hide')) # with in that formed tree go find
1970 # me the tag named email, look for the attribute hide for it
1971
1972 -----
1973
1974 # in the above example we did extract information from only one tree, but in many cases
1975 # we may have to extract the information from the list of trees.
1976
1977 import xml.etree.ElementTree as ET # loading library, built in xml parser library
1978 inputs = '''<stuff> # creating dummy xml data, latter on we will
1979     parse from the website
1980         <users>
1981             <user x="2">
1982                 <id>001</id>
1983                 <name>sagar</name>
1984             </user>
1985             <user x="7">
1986                 <id>009</id>
1987                 <name>Kashi</name>
1988             </user>
1989         </users>
1990     </stuff>'''
1991
1992 stuff = ET.fromstring(inputs) # making a nice tree named stuff from the dummy xml
1993 # data
1994 lst = stuff.findall('users/user') # because we are accessing a multiple tags so making
1995 # a list to store it
1996 print('user count', len(lst)) # looking how many items are in the list
1997
1998 for item in lst:
1999     print('Name:', item.find('name').text) # accessing each item of the list to retrieve
2000     # particular information.
2001     print('ID:', item.find('id').text)
2002     print('Attribute', item.get('x'))
2003
2004 -----
2005
2006 # JSON [javaScrip object Notation]
2007 # So now we're going to talk about a new serialization format. We've talked about XML,
2008 # which is kind of complex. And there's a simple one called JSON. XML is way
2009 # powerful and we did not use it a lot. JSON is quite popular nowadays.
2010 # Sample code for JSON
2011
2012 import json # we have a inbuilt function called json to parse data.
2013
2014 data = '''{ "name" : "Sagar", # sample demo data we are creating,

```

```

later on we will look from the web
2001         "phone" : {
2002             "type": "intl",
2003             "number" : "+1 (618) 917-9128"},
2004         "email" : { "hide": "yes" }
2005     }'''
2006
2007 info = json.loads(data)                # json.loads() is a function that arrange that
messy data into a nice tree of information
2008 print('Name:', info['name'])
2009 print('Email:', info['email']['hide'])
2010 -----
2011 # Making the list of dictionary tree and extracting the information from there
2012
2013 import json
2014
2015 data = '''[
2016     {
2017         "name": "Sagar",
2018         "Id" : "800752****",
2019         "email" : "skalaun@siue.edu"
2020     },
2021     {
2022         "name" : "kashi",
2023         "Id" : "800700****",
2024         "email" : "kashi@gmail.com"
2025     }
2026 ]'''
2027
2028 info = json.loads(data)
2029 print('User count', len(info))
2030
2031 for item in info:
2032     print("Name:", item['name'])
2033     print("email:", item['email'])
2034     print("id no.", item['Id'])
2035 -----
2036 # this is the general format to give parameter at the end of the url, where (+) means
space and (%2C) means comma.
2037 https://maps.googleapis.com/maps/api/geocode/json?address=Ann+arbor%2C+MI
2038 -----
2039 -----
2040 # Now using all the concepts we have learned till now and accessing the data from the
web, converting them to useful format and retriving data from there
2041
2042 import urllib.request, urllib.parse, urllib.error
2043 import json
2044
2045 # Note that Google is increasingly requiring keys
2046 # for this API
2047 serviceurl = 'http://maps.googleapis.com/maps/api/geocode/json?'    # The web address of
the API
2048
2049 while True:
2050     address = input('Enter location: ')    # Ask the user to input an address
2051     if len(address) < 1: break    # If the entered line is blank, break
2052
2053     url = serviceurl + urllib.parse.urlencode(    # Make a url by concatenating the API
url and the url form of the address
        {'address': address})
2054
2055     print('Retrieving', url)
2056     uh = urllib.request.urlopen(url)    # Get a handle for the url
2057     data = uh.read().decode()    #call the read method to pull the entire document

```

```

    & decode (from probably UTF-8)
2059 print('Retrieved', len(data), 'characters')
2060
2061 try:
2062     js = json.loads(data)    # Parse the data as string data
2063 except:
2064     js = None
2065
2066 if not js or 'status' not in js or js['status'] != 'OK':    # Check for failures -
if js is false, if status key is missing, or status is not equal to "OK"
    print('==== Failure To Retrieve ====')
    print(data)
    continue
2070
2071 print(json.dumps(js, indent=4))
2072
2073 lat = js["results"][0]["geometry"]["location"]["lat"]    # Walking down the tree of
keys to look for
2074 lng = js["results"][0]["geometry"]["location"]["lng"]
2075 print('lat', lat, 'lng', lng)
2076 location = js['results'][0]['formatted_address']
2077 print(location)
2078
2079 ~~~~~
2080 ~~~~~Using database with python~~~~~
2081 # Python string to bytes
2082 # When we talk to external resource like a network socket we sends bytes, so we need to
encode python 3 string into a given character encoding.
2083 # When we read data from the external resource, we must decode it based on the character
set so it is properly repersented in python 3 as a string.
2084
2085 While True:
2086     data = mysock.recv(512)
2087     if (len(data)<1):
2088         break
2089     mystring = data.decode()    # usually we have to specify from what to decode from
like utf-8, Ascii or other,
2090     print(mystring)
2091 ~~~~~
2092 # Object Oriented programming
2093 # Then the concept of dog is like a class. But when you see a dog and you grab the dog,
that's an object.
2094 # Some python objects
2095
2096 x = 'abc'
2097 type(x)
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115

```

2116  
2117  
2118  
2119  
2120  
2121  
2122  
2123  
2124  
2125  
2126  
2127  
2128  
2129  
2130  
2131  
2132  
2133  
2134  
2135  
2136  
2137  
2138  
2139  
2140  
2141  
2142  
2143  
2144  
2145  
2146  
2147  
2148  
2149  
2150  
2151  
2152  
2153  
2154  
2155  
2156  
2157  
2158  
2159  
2160  
2161  
2162  
2163  
2164  
2165  
2166  
2167  
2168  
2169  
2170  
2171  
2172  
2173  
2174  
2175  
2176  
2177  
2178  
2179  
2180  
2181  
2182

2183  
2184  
2185  
2186  
2187  
2188  
2189  
2190  
2191  
2192  
2193  
2194  
2195  
2196  
2197  
2198  
2199  
2200  
2201  
2202  
2203  
2204  
2205  
2206  
2207  
2208  
2209  
2210  
2211  
2212  
2213  
2214  
2215  
2216  
2217  
2218  
2219  
2220  
2221  
2222  
2223  
2224  
2225  
2226  
2227  
2228  
2229  
2230  
2231  
2232  
2233  
2234  
2235  
2236  
2237  
2238  
2239  
2240  
2241  
2242  
2243  
2244  
2245  
2246  
2247  
2248  
2249

2250  
2251  
2252  
2253  
2254  
2255  
2256  
2257  
2258  
2259  
2260  
2261  
2262  
2263  
2264  
2265  
2266  
2267  
2268  
2269  
2270  
2271  
2272  
2273  
2274  
2275  
2276  
2277  
2278  
2279  
2280  
2281  
2282  
2283  
2284  
2285  
2286  
2287  
2288  
2289  
2290  
2291  
2292  
2293  
2294  
2295  
2296  
2297  
2298  
2299  
2300  
2301  
2302  
2303  
2304  
2305  
2306  
2307  
2308  
2309  
2310  
2311  
2312  
2313  
2314  
2315  
2316



2317  
2318  
2319  
2320  
2321  
2322  
2323  
2324  
2325  
2326  
2327  
2328  
2329  
2330  
2331  
2332  
2333  
2334  
2335  
2336  
2337  
2338  
2339  
2340  
2341  
2342  
2343  
2344  
2345  
2346  
2347  
2348  
2349  
2350  
2351  
2352  
2353  
2354  
2355  
2356  
2357  
2358  
2359  
2360  
2361  
2362  
2363  
2364  
2365  
2366  
2367  
2368  
2369  
2370  
2371  
2372  
2373  
2374  
2375  
2376  
2377  
2378  
2379  
2380  
2381  
2382  
2383

2384  
2385  
2386  
2387  
2388  
2389  
2390  
2391  
2392  
2393  
2394  
2395  
2396  
2397  
2398  
2399  
2400  
2401  
2402  
2403  
2404  
2405  
2406  
2407  
2408  
2409  
2410  
2411  
2412  
2413  
2414  
2415  
2416  
2417  
2418  
2419  
2420  
2421  
2422  
2423  
2424  
2425  
2426  
2427  
2428  
2429  
2430  
2431  
2432  
2433  
2434  
2435  
2436  
2437  
2438  
2439  
2440  
2441