FEU Institute of Technology

(INTEGRATIVE PROGRAMMING AND TECHNOLOGIES)

EXERCISE

4

(Manipulation Collection Elements)

Guillermo, Justine Rome	Dr. Roman DeAngel
February 5, 2020	February 5, 2020

Lists

Lists are one of the great datastructures in Python. We are going to learn a little bit about lists now. Basic knowledge of lists is requrired to be able to solve some problems that we want to solve in this chapter.

Here is a list of numbers.

```
>>> x = [1, 2, 3]
```

And here is a list of strings.

```
>>> x = ["hello", "world"]
```

List can be heterogeneous. Here is a list containings integers, strings and another list.

```
>>> x = [1, 2, "hello", "world", ["another", "list"]]
```

The built-in function `len` works for lists as well.

```
>>> x = [1, 2, 3]
>>> len(x)
3
```

The `[]` operator is used to access individual elements of a list.

```
>>> x = [1, 2, 3]
>>> x[1]
2
>>> x[1] = 4
>>> x[1]
4
```

The first element is indexed with `0`, second with `1` and so on.

We'll learn more about lists in the next chapter.

Modules

Modules are libraries in Python. Python ships with many standard library modules.

A module can be imported using the `import` statement.

Lets look at `time` module for example:

```
>>> import time
>>> time.asctime()
'Tue Sep 11 21:42:06 2012'
```

The `asctime` function from the `time` module returns the current time of the system as a string.

The `sys` module provides access to the list of arguments passed to the program, among the other things.

The `sys.argv` variable contains the list of arguments passed to the program. As a convention, the first element of that list is the name of the program.

Lets look at the following program `echo.py` that prints the first argument passed to it.

```
import sys
print sys.argv[1]
```

Lets try running it.

```
$ python echo.py hello
hello
$ python echo.py hello world
hello
```

There are many more interesting modules in the standard library. We'll learn more about them in the coming chapters.

Problem 19: Write a program `add.py` that takes 2 numbers as command line arguments and prints its sum.

```
$ python add.py 3 5
8
$ python add.py 2 9
11
```

Lists

We've already seen quick introduction to lists in the previous chapter.

```
>>> [1, 2, 3, 4]
[1, 2, 3, 4]
>>> ["hello", "world"]
["hello", "world"]
>>> [0, 1.5, "hello"]
[0, 1.5, "hello"]
>>> [0, 1.5, "hello"]
[0, 1.5, "hello"]
```

A List can contain another list as member.

```
>>> a = [1, 2]
>>> b = [1.5, 2, a]
>>> b
[1.5, 2, [1, 2]]
```

The built-in function `range` can be used to create a list of integers.

```
>>> range(4)
[0, 1, 2, 3]
>>> range(3, 6)
[3, 4, 5]
>>> range(2, 10, 3)
[2, 5, 8]
```

The built-in function `len` can be used to find the length of a list.

```
>>> a = [1, 2, 3, 4]
>>> len(a)
4
```

The + and * operators work even on lists.

```
>>> a = [1, 2, 3]
>>> b = [4, 5]
>>> a + b
[1, 2, 3, 4, 5]
>>> b * 3
[4, 5, 4, 5, 4, 5]
```

List can be indexed to get individual entries. Value of index can go from 0 to (length of list - 1).

```
>>> x = [1, 2]
>>> x[0]
1
>>> x[1]
2
```

When a wrong index is used, python gives an error.

```
>>> x = [1, 2, 3, 4]
>>> x[6]
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: list index out of range
```

Negative indices can be used to index the list from right.

```
>>> x = [1, 2, 3, 4]
>>> x[-1]
4
>>> x [-2]
3
```

We can use list slicing to get part of a list.

```
>>> x = [1, 2, 3, 4]
>>> x[0:2]
[1, 2]
>>> x[1:4]
[2, 3, 4]
```

Even negative indices can be used in slicing. For example, the following examples strips the last element from the list.

```
>>> x[0:-1]
[1, 2, 3]
```

Slice indices have useful defaults; an omitted first index defaults to zero, an omitted second index defaults to the size of the list being sliced.

```
>>> x = [1, 2, 3, 4]
>>> a[:2]
[1, 2]
>>> a[2:]
[3, 4]
>>> a[:]
[1, 2, 3, 4]
```

An optional third index can be used to specify the increment, which defaults to 1.

```
>>> x = range(10)
>>> x
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> x[0:6:2]
[0, 2, 4]
```

We can reverse a list, just by providing -1 for increment.

```
>>> x[::-1]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

List members can be modified by assignment.

```
>>> x = [1, 2, 3, 4]
>>> x[1] = 5
>>> x
[1, 5, 3, 4]
```

Presence of a key in a list can be tested using `in` operator.

```
>>> x = [1, 2, 3, 4]
>>> 2 in x
True
>>> 10 in x
False
```

Values can be appended to a list by calling `append` method on list. A method is just like a function, but it is associated with an object and can access that object when it is called. We will learn more about methods when we study classes.

```
>>> a = [1, 2]
>>> a.append(3)
>>> a
[1, 2, 3]
```

Problem 20: What will be the output of the following program?

```
x = [0, 1, [2]]
x[2][0] = 3
print x
x[2].append(4)
print x
x[2] = 2
print x
```

The for Statement

Python provides `for` statement to iterate over a list. A `for` statement executes the specified block of code for every element in a list.

```
for x in [1, 2, 3, 4]:
    print x

for i  in range(10):
   print i, i*i, i*i*i
```

The built-in function `zip` takes two lists and returns list of pairs.

```
>>> zip(["a", "b", "c"], [1, 2, 3])
[('a', 1), ('b', 2), ('c', 3)]
```

It is handy when we want to iterate over two lists together.

```
names = ["a", "b", "c"]
values = [1, 2, 3]
for name, value in zip(names, values):
    print name, value
```

Problem 21: Python has a built-in function `sum` to find sum of all elements of a list. Provide an implementation for `sum`.

```
>>> sum([1, 2, 3])
>>> 6
```

Problem 22: What happens when the above `sum` function is called with a list of strings? Can you make your `sum` function work for a list of strings as well.

```
>>> sum(["hello", "world"])
"helloworld"
>>> sum(["aa", "bb", "cc"])
"aabbcc"
```

Problem 23: Implement a function `product`, to compute product of a list of numbers.

```
>>> product([1, 2, 3])
6
```

Problem 24: Write a function `factorial` to compute factorial of a number. Can you use the `product` function defined in the previous example to compute factorial?

```
>>> factorial(4)
24
```

Problem 25: Write a function `reverse` to reverse a list. Can you do this without using list slicing?

```
>>> reverse([1, 2, 3, 4])
[4, 3, 2, 1]
>>> reverse(reverse([1, 2, 3, 4]))
[1, 2, 3, 4]
```

Problem 26: Python has built-in functions `min` and `max` to compute minimum and maximum of a given list. Provide an implementation for these functions. What happens when you call your `min` and `max` functions with a list of strings?

Problem 27: Cumulative sum of a list `[a, b, c, ...]` is defined as `[a, a+b, a+b+c, ...]`. Write a function `cumulative_sum` to compute cumulative sum of a list. Does your implementation work for a list of strings?

```
>>> cumulative_sum([1, 2, 3, 4])
[1, 3, 6, 10]
>>> cumulative_sum([4, 3, 2, 1])
[4, 7, 9, 10]
```

Problem 28: Write a function `cumulative_product` to compute cumulative product of a list of numbers.

```
>>> cumulative_product([1, 2, 3, 4])
[1, 2, 6, 24]
>>> cumulative_product([4, 3, 2, 1])
[4, 12, 24, 24]
```

Problem 29: Write a function *unique* to find all the unique elements of a list.

```
>>> unique([1, 2, 1, 3, 2, 5])
[1, 2, 3, 5]
```

Problem 30: Write a function *dups* to find all duplicates in the list.

```
>>> dups([1, 2, 1, 3, 2, 5])
[1, 2]
```

Problem 31: Write a function *group(list, size)* that take a list and splits into smaller lists of given size.

```
>>> group([1, 2, 3, 4, 5, 6, 7, 8, 9], 3)
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> group([1, 2, 3, 4, 5, 6, 7, 8, 9], 4)
[[1, 2, 3, 4], [5, 6, 7, 8], [9]]
```

Sorting Lists

The `sort` method sorts a list in place.

```
>>> a = [2, 10, 4, 3, 7]
>>> a.sort()
>>> a
[2, 3, 4, 7 10]
```

The built-in function `sorted` returns a new sorted list without modifying the source list.

```
>>> a = [4, 3, 5, 9, 2]
>>> sorted(a)
[2, 3, 4, 5, 9]
>>> a
[4, 3, 5, 9, 2]
```

The behavior of `sort` method and `sorted` function is exactly same except that sorted returns a new list instead of modifying the given list.

The `sort` method works even when the list has different types of objects and even lists.

```
>>> a = ["hello", 1, "world", 45, 2]
>>> a.sort()
>>> a
[1, 2, 45, 'hello', 'world']
>>> a = [[2, 3], [1, 6]]
>>> a.sort()
>>> a
[[1, 6], [2, 3]]
```

We can optionally specify a function as sort key.

```
>>> a = [[2, 3], [4, 6], [6, 1]]
>>> a.sort(key=lambda x: x[1])
>>> a
[[6, 1], [2, 3],  [4 6]]
```

This sorts all the elements of the list based on the value of second element of each entry.

Problem 32: Write a function `lensort` to sort a list of strings based on length.

```
>>> lensort(['python', 'perl', 'java', 'c', 'haskell', 'ruby'])
['c', 'perl', 'java', 'ruby', 'python', 'haskell']
```

Problem 33: Improve the *unique* function written in previous problems to take an optional *key* function as argument and use the return value of the key function to check for uniqueness.

```
>>> unique(["python", "java", "Python", "Java"], key=lambda s: s.lower())
["python", "java"]
```

Tuples

Tuple is a sequence type just like `list`, but it is immutable. A tuple consists of a number of values separated by commas.

```
>>> a = (1, 2, 3)
>>> a[0]
1
```

The enclosing braces are optional.

```
>>> a = 1, 2, 3
>>> a[0]
1
```

The built-in function `len` and slicing works on tuples too.

```
>>> len(a)
3
>>> a[1:]
2, 3
```

Since parenthesis are also used for grouping, tuples with a single value are represented with an additional comma.

```
>>> a = (1)
>> a
1
>>> b = (1,)
>>> b
(1,)
>>> b[0]
1
```

Sets

Sets are unordered collection of unique elements.

```
>>> x = set([3, 1, 2, 1])
set([1, 2, 3])
```

Python 2.7 introduced a new way of writing sets.

```
>>> x = {3, 1, 2, 1}
set([1, 2, 3])
```

New elements can be added to a set using the `add` method.

```
>>> x = set([1, 2, 3])
>>> x.add(4)
>>> x
set([1, 2, 3, 4])
```

Just like lists, the existance of an element can be checked using the `in` operator. However, this operation is faster in sets compared to lists.

```
>>> x = set([1, 2, 3])
>>> 1 in x
True
>>> 5 in x
False
```

Problem 34: Reimplement the *unique* function implemented in the earlier examples using sets.

List Comprehensions

List Comprehensions provide a concise way of creating lists. Many times a complex task can be modelled in a single line.

Here are some simple examples for transforming a list.

```
>>> a = range(10)
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> [x for x in a]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> [x*x for x in a]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> [x+1 for x in a]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

It is also possible to filter a list using `if` inside a list comprehension.

```
>>> a = range(10)
>>> [x for x in a if x % 2 == 0]
[0, 2, 4, 6, 8]
>>> [x*x for x in a if x%2 == 0]
[0, 4, 8, 36, 64]
```

It is possible to iterate over multiple lists using the built-in function `zip`.

```
>>> a = [1, 2, 3, 4]
>>> b = [2, 3, 5, 7]
>>> zip(a, b)
[(1, 2), (2, 3), (3, 5), (4, 7)]
>>> [x+y for x, y in zip(a, b)]
[3, 5, 8, 11]
```

we can use multiple `for` clauses in single list comprehension.

```
>>> [(x, y) for x in range(5) for y in range(5) if (x+y)%2 == 0]
[(0, 0), (0, 2), (0, 4), (1, 1), (1, 3), (2, 0), (2, 2), (2, 4), (3, 1), (3, 3), (4,

>>> [(x, y) for x in range(5) for y in range(5) if (x+y)%2 == 0 and x != y]
[(0, 2), (0, 4), (1, 3), (2, 0), (2, 4), (3, 1), (4, 0), (4, 2)]

>>> [(x, y) for x in range(5) for y in range(x) if (x+y)%2 == 0]
[(2, 0), (3, 1), (4, 0), (4, 2)]
```

The following example finds all Pythagorean triplets using numbers below 25. `(x, y, z)` is a called pythagorean triplet if `x*x + y*y == z*z`.

```
>>> n = 25
>>> [(x, y, z) for x in range(1, n) for y in range(x, n) for z in range(y, n) if x*x
[(3, 4, 5), (5, 12, 13), (6, 8, 10), (8, 15, 17), (9, 12, 15), (12, 16, 20)]
```

Problem 43: Provide an implementation for `zip` function using list comprehensions.

```
>>> zip([1, 2, 3], ["a", "b", "c"])
[(1, "a"), (2, "b"), (3, "c")]
```

Problem 44: Python provides a built-in function `map` that applies a function to each element of a list. Provide an implementation for `map` using list comprehensions.

```
>>> def square(x): return x * x
...
>>> map(square, range(5))
[0, 1, 4, 9, 16]
```

Problem 45: Python provides a built-in function `filter(f, a)` that returns items of the list `a` for which `f(item)` returns true. Provide an implementation for `filter` using list comprehensions.

```
>>> def even(x): return x %2 == 0
...
>>> filter(even, range(10))
[0, 2, 4, 6, 8]
```

Problem 46: Write a function `triplets` that takes a number `n` as argument and returns a list of triplets such that sum of first two elements of the triplet equals the third element using numbers below n. Please note that `(a, b, c)` and `(b, a, c)` represent same triplet.

```
>>> triplets(5)
[(1, 1, 2), (1, 2, 3), (1, 3, 4), (2, 2, 4)]
```

Problem 47: Write a function `enumerate` that takes a list and returns a list of tuples containing `(index,item)` for each item in the list.

```
>>> enumerate(["a", "b", "c"])
[(0, "a"), (1, "b"), (2, "c")]
>>> for index, value in enumerate(["a", "b", "c"]):
...     print index, value
0 a
1 b
2 c
```

Problem 48: Write a function `array` to create an 2-dimensional array. The function should take both dimensions as arguments. Value of each element can be initialized to None:

```
>>> a = array(2, 3)
>>> a
[[None, None, None], [None, None, None]]
>>> a[0][0] = 5
[[5, None, None], [None, None, None]]
```

Problem 49: Write a python function `parse_csv` to parse csv (comma separated values) files.

```
>>> print open('a.csv').read()
a,b,c
1,2,3
2,3,4
3,4,5
>>> parse_csv('a.csv')
[['a', 'b', 'c'], ['1', '2', '3'], ['2', '3', '4'], ['3', '4', '5']]
```

Problem 50: Generalize the above implementation of csv parser to support any delimiter and comments.

```
>>> print open('a.txt').read()
# elements are separated by ! and comment indicator is #
a!b!c
1!2!3
2!3!4
3!4!5
>>> parse('a.txt', '!', '#')
[['a', 'b', 'c'], ['1', '2', '3'], ['2', '3', '4'], ['3', '4', '5']]
```

Problem 51: Write a function `mutate` to compute all words generated by a single mutation on a given word. A mutation is defined as inserting a character, deleting a character, replacing a character, or swapping 2 consecutive characters in a string. For simplicity consider only letters from `a` to `z`.

```
>>> words = mutate('hello')
>>> 'helo' in words
True
>>> 'cello' in words
True
>>> 'helol' in words
True
```

Problem 52: Write a function `nearly_equal` to test whether two strings are nearly equal. Two strings `a` and `b` are nearly equal when `a` can be generated by a single mutation on `b`.

```
>>> nearly_equal('python', 'perl')
False
>>> nearly_equal('perl', 'pearl')
True
>>> nearly_equal('python', 'jython')
True
>>> nearly_equal('man', 'woman')
False
```

Dictionaries

Dictionaries are like lists, but they can be indexed with non integer keys also. Unlike lists, dictionaries are not ordered.

```
>>> a = {'x': 1, 'y': 2, 'z': 3}
>>> a['x']
1
>>> a['z']
3
>>> b = {}
>>> b['x'] = 2
>>> b[2] = 'foo'
>>> b[(1, 2)] = 3
>>> b
{(1, 2): 3, 'x': 2, 2: 'foo'}
```

The `del` keyword can be used to delete an item from a dictionary.

```
>>> a = {'x': 1, 'y': 2, 'z': 3}
>>> del a['x']
>>> a
{'y': 2, 'z': 3}
```

The `keys` method returns all keys in a dictionary, the `values` method returns all values in a dictionary and `items` method returns all key-value pairs in a dictionary.

```
>>> a.keys()
['x', 'y', 'z']
>>> a.values()
[1, 2, 3]
>>> a.items()
[('x', 1), ('y', 2), ('z', 3)]
```

The `for` statement can be used to iterate over a dictionary.

```
>>> for key in a: print key
...
x
```

```
y
z
>>> for key, value in a.items(): print key, value
...
x 1
y 2
z 3
```

Presence of a key in a dictionary can be tested using `in` operator or `has_key` method.

```
>>> 'x' in a
True
>>> 'p' in a
False
>>> a.has_key('x')
True
>>> a.has_key('p')
False
```

Other useful methods on dictionaries are `get` and `setdefault`.

```
>>> d = {'x': 1, 'y': 2, 'z': 3}
>>> d.get('x', 5)
1
>>> d.get('p', 5)
5
>>> d.setdefault('x', 0)
1
>>> d
{'x': 1, 'y': 2, 'z': 3}
>>> d.setdefault('p', 0)
0
>>> d
{'y': 2, 'x': 1, 'z': 3, 'p': 0}
```

Dictionaries can be used in string formatting to specify named parameters.

```
>>> 'hello %(name)s' % {'name': 'python'}
'hello python'
>>> 'Chapter %(index)d: %(name)s' % {'index': 2, 'name': 'Data Structures'}
'Chapter 2: Data Structures'
```

Example: Word Frequency

Suppose we want to find number of occurrences of each word in a file. Dictionary can be used to store the number of occurrences for each word.

Lets first write a function to count frequency of words, given a list of words.

```
def word_frequency(words):
    """Returns frequency of each word given a list of words.

        >>> word_frequency(['a', 'b', 'a'])
        {'a': 2, 'b': 1}
    """
    frequency = {}
    for w in words:
        frequency[w] = frequency.get(w, 0) + 1
    return frequency
```

Getting words from a file is very trivial.

```
def read_words(filename):
    return open(filename).read().split()
```

We can combine these two functions to find frequency of all words in a file.

```
def main(filename):
    frequency = word_frequency(read_words(filename))
    for word, count in frequency.items():
        print word, count

if __name__ == "__main__":
    import sys
    main(sys.argv[1])
```

Problem 53: Improve the above program to print the words in the descending order of the number of occurrences.

Problem 54: Write a program to count frequency of characters in a given file. Can you use character frequency to tell whether the given file is a Python program file, C program file or a text file?

Problem 55: Write a program to find anagrams in a given list of words. Two words are called anagrams if one word can be formed by rearranging letters of another. For example 'eat', 'ate' and 'tea' are anagrams.

```
>>> anagrams(['eat', 'ate', 'done', 'tea', 'soup', 'node'])
[['eat', 'ate', 'tea], ['done', 'node'], ['soup']]
```

Problem 56: Write a function `valuesort` to sort values of a dictionary based on the key.

```
>>> valuesort({'x': 1, 'y': 2, 'a': 3})
[3, 1, 2]
```

Problem 57: Write a function `invertdict` to interchange keys and values in a dictionary. For simplicity, assume that all values are unique.

```
>>> invertdict({'x': 1, 'y': 2, 'z': 3})
{1: 'x', 2: 'y', 3: 'z'}
```

Understanding Python Execution Environment

Python stores the variables we use as a dictionary. The `globals()` function returns all the globals variables in the current environment.

```
>>> globals()
{'__builtins__': <module '__builtin__' (built-in)>, '__name__': '__main__', '__doc__'
>>> x = 1
>>> globals()
{'__builtins__': <module '__builtin__' (built-in)>, '__name__': '__main__', '__doc__'
>>> x = 2
>>> globals()
{'__builtins__': <module '__builtin__' (built-in)>, '__name__': '__main__', '__doc__'
>>> globals()['x'] = 3
>>> x
3
```

Just like `globals` python also provides a function `locals` which gives all the local variables in a function.

```
>>> def f(a, b): print locals()
...
>>> f(1, 2)
{'a': 1, 'b': 2}
```

```
>>> def f(name):
...     return "Hello %(name)s!" % locals()
...
>>> f("Guido")
Hello Guido!
```

Further Reading:

- The article A Plan for Spam by Paul Graham describes a method of detecting spam using probability of occurrence of a word in spam.

Problem 19:

```
# PROBLEM 19
import sys
print(int(sys.argv[1]) + int(sys.argv[2]))
```

```
C:\Users\ajajc\PycharmProjects\January18\module4>python add.py 3 5
8

C:\Users\ajajc\PycharmProjects\January18\module4>python add.py 2 9
11
```

Problem 20:

The output will be:

Output:

[0, 1, [3]]
[0, 1, [3, 4]]
[0, 1, 2]

Problem 21:

```
print("Sum of", [1,2,3,4,5,6,7,8,9,10])
print(sum([1,2,3,4,5,6,7,8,9,10]), end='')
```

```
Sum of [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
55
Process finished with exit code 0
```

Problem 22:

```
# PROBLEM 22
def sum(name_list):
    total_string = ""
    for x in name_list:
        total_string = total_string + x
    return total_string
print(["hello", "world"])
print(sum(["hello", "world"]), end='')
```

```
['hello', 'world']
helloworld
Process finished with exit code 0
```

Problem 23:

```
# PROBLEM 23
def product(name_list):
    total_product = 1
    for x in name_list:
        total_product = total_product * x
    return total_product
print([1,2,3])
print("Prodcut =",product([1,2,3]), end='')
```

```
[1, 2, 3]
Prodcut = 6
Process finished with exit code 0
```

Problem 24:

```
# PROBLEM 24
def factorial(num):
    fact = 1
    counter = 1
    while counter <= num:
        fact = fact * counter;
        counter += 1
    return fact
print("Factorial of 7 = ",end='')
print(factorial(7), end='')
```

```
Factorial of 7 = 5040
Process finished with exit code 0
```

Problem 25:

```
# PROBLEM 25
def reverse(char_list):
    x = len(char_list)
    x -= 1
    reverse_content = []
    while x >= 0:
        reverse_content.append(char_list[x])
        x -= 1
    return reverse_content
print("Content =", [1,2,3,4])
print("Reverse = ",reverse([1,2,3,4]))
print("Reverse then reverse =",reverse(reverse([1,2,3,4])), end='')
```

```
Content = [1, 2, 3, 4]
Reverse =  [4, 3, 2, 1]
Reverse then reverse = [1, 2, 3, 4]
Process finished with exit code 0
```

Problem 26:

```
# PROBLEM 26
print("This is the List in Integer [2,1,7,4,5,6,3]")
print(f"Minimum = {min([2,1,7,4,5,6,3])}")
print(f"Maximum = {max([2,1,7,4,5,6,3])}")

print("This is the List in Strings ['ea','ae','uq','og','ie']")
print(f"Minimum = {min(['ea','ae','uq','og','ie'])}")
print(f"Maximum = {max(['ea','ae','uq','og','ie'])}", end='')
```

```
This is the List in Integer [2,1,7,4,5,6,3]
Minimum = 1
Maximum = 7
This is the List in Strings ['ea','ae','uq','og','ie']
Minimum = ae
Maximum = uq
Process finished with exit code 0
```

Problem 27:

```
# PROBLEM 27
def cumulative_sum(char_list):
    new_char_list = []
    x = len(char_list)
    x -= 1
    ctr = 0
    while ctr <= x:
        if ctr == 0:
            new_char_list.append(char_list[ctr])
        else:
            inner_ctr = 0
            total = 0
            while inner_ctr <= ctr:
                total = total + char_list[inner_ctr]
                inner_ctr += 1
            new_char_list.append(total)
        ctr += 1
    return new_char_list
print([1,2,3,4], "= ", end='')
print(cumulative_sum([1,2,3,4]))
print([4,3,2,1], "= ", end='')
print(cumulative_sum([4,3,2,1]), end='')
```

```
[1, 2, 3, 4] = [1, 3, 6, 10]
[4, 3, 2, 1] = [4, 7, 9, 10]
Process finished with exit code 0
```

Problem 28:

```
# PROBLEM 27
def cumulative_product(char_list):
    new_char_list = []
    x = len(char_list)
    x -= 1
    ctr = 0
    while ctr <= x:
        if ctr == 0:
            new_char_list.append(char_list[ctr])
        else:
            inner_ctr = 0
            total = 1
            while inner_ctr <= ctr:
                total = total * char_list[inner_ctr]
                inner_ctr += 1
            new_char_list.append(total)
        ctr += 1
    return new_char_list
```

```
print([1,2,3,4], "= ", end='')
print(cumulative_product([1,2,3,4]))
print([4,3,2,1], "= ", end='')
print(cumulative_product([4,3,2,1]), end='')
```

```
[1, 2, 3, 4] = [1, 2, 6, 24]
[4, 3, 2, 1] = [4, 12, 24, 24]
Process finished with exit code 0
```

Problem 29:

```
# PROBLEM 29
def unique(name_list):
    unique_content = []
    for x in name_list:
        if x in unique_content:
            continue
        else:
            unique_content.append(x)
    return unique_content
print("Content =",[1,2,1,3,2,5])
print("Unique =",unique([1,2,1,3,2,5]), end='')
```

```
Content = [1, 2, 1, 3, 2, 5]
Unique = [1, 2, 3, 5]
Process finished with exit code 0
```

Problem 30:

```
# PROBLEM 30
def dups(name_list):
    duplicate_content = []
    ctr = 0
    x = len(name_list)
    x-=1
    while ctr <= x:
        inr_ctr = ctr + 1
        while inr_ctr <= x:
            if name_list[ctr] == name_list[inr_ctr]:
                duplicate_content.append(name_list[ctr])
            inr_ctr += 1
        ctr += 1
    return duplicate_content
```

```
print("Content =",[1,2,1,3,2,5])
print("Duplicate =",dups([1,2,1,3,2,5]), end='')
```

```
Content = [1, 2, 1, 3, 2, 5]
Duplicate = [1, 2]
Process finished with exit code 0
```

Problem 31:

```
# PROBLEM 31
def group(name_list, size):
    new_content = []
    inner_list = []
    ctr = 0
    x = len(name_list)
    get_modulo = x % size
    new = x-get_modulo
    x -= 1
    while ctr <= x:
        inner_list.append(name_list[ctr])
        if(len(inner_list) == size):
            new_content.append(inner_list)
            inner_list = []
        ctr += 1
    while new <= x:
        new_content.append(name_list[new])
        new += 1
    return new_content
print("Content =", [1,2,3,4,5,6,7,8,9])
print("3 =",group([1,2,3,4,5,6,7,8,9], 3))
print("4 =",group([1,2,3,4,5,6,7,8,9], 4), end='')
```

```
Content = [1, 2, 3, 4, 5, 6, 7, 8, 9]
3 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
4 = [[1, 2, 3, 4], [5, 6, 7, 8], 9]
Process finished with exit code 0
```

Problem 32:

```
# Problem 32
def lensort(words):
    num = []
    new_content = []
    ctr = 0
    for x in words:
        num.append(len(x))
    num.sort()
    while ctr < len(num):
        in_ctr = 0
        while in_ctr < len(words):
            if(len(words[in_ctr]) == num[ctr]):
                if words[in_ctr] not in new_content:
                    new_content.append(words[in_ctr])
            in_ctr += 1
        ctr += 1
    return new_content
print("Content =", ['python', 'perl', 'java', 'c', 'haskell', 'ruby'])
print("Sort by length =", lensort(['python', 'perl', 'java', 'c', 'haskell',
'ruby']),end='')
```

```
Content = ['python', 'perl', 'java', 'c', 'haskell', 'ruby']
Sort by length = ['c', 'perl', 'java', 'ruby', 'python', 'haskell']
Process finished with exit code 0
```

Problem 33:

```
# PROBLEM 33
def unique(name_list, tolowers):
    unique_content = []
    for x in name_list:
        if tolowers(x) in unique_content:
            continue
        else:
            unique_content.append(tolowers(x))
    return unique_content
print("Content = ['Python', 'java', 'python', 'Java', 'Sample', 'sampLE']")
print("Unique  =", unique(['Python', 'java', 'python', 'Java', 'Sample', 'sampLE'],
lambda s: s.lower()),  end='')
```

```
Content = ['Python', 'java', 'python', 'Java', 'Sample', 'sampLE']
Unique  = ['python', 'java', 'sample']
Process finished with exit code 0
```

Problem 34:

```
# PROBLEM 34
def unique(name_list):
    return name_list
print("Content = {1,2,1,3,2,5}")
print("Unique =", unique({1,2,1,3,2,5}), end='')
```

```
Content = {1,2,1,3,2,5}
Unique = {1, 2, 3, 5}
Process finished with exit code 0
```

Problem 43:

```
# PROBLEM 43
a = [1,2,3]
b = ['a', 'b', 'c']
c = zip(a,b)
print("a = [1,2,3]\nb = ['a', 'b', 'c']")
print(list(c), end='')
```

```
a = [1,2,3]
b = ['a', 'b', 'c']
[(1, 'a'), (2, 'b'), (3, 'c')]
Process finished with exit code 0
```

Problem 44:

```
# PROBLEM 44
def square(x):
    return x*x
result = map(square, range(5))
print("Squared of range(5)")
print(list(result), end='')
```

```
Squared of range(5)
[0, 1, 4, 9, 16]
Process finished with exit code 0
```

Problem 45:

```
# Problem 45
def even(x):
    return x%2 == 0
print("Even Numbers of range(10)")
print(list(filter(even, range(10))), end='')
```

```
Even Numbers of range(10)
[0, 2, 4, 6, 8]
Process finished with exit code 0
```

Problem 46:

```
def triplets(n):
    list_final = []
    for a in range(1, n):
        for b in range(a, n):
            for c in range(b, n):
                if a + b == c:
                    list_final.append((a, b, c))
    return list_final
print("triplets(5) =",triplets(5), end='')
```

```
triplets(5) = [(1, 1, 2), (1, 2, 3), (1, 3, 4), (2, 2, 4)]
Process finished with exit code 0
```

Problem 47:

```
#Problem 47
print("enumerate(['a','b','c']")
for index, value in enumerate(["a","b","c"]):
    print(index, value)
```

```
enumerate(['a','b','c']
0 a
1 b
2 c
```

Problem 48:

```
# Problem 48
def array(row,col):
    new_list = []
    ctr = 0
    while ctr < row:
        inner_ctr = 0
        inner_list = []
        while inner_ctr < col:
            inner_list.append("None")
            inner_ctr += 1
        new_list.append(inner_list)
        ctr += 1
    return new_list
print("a = array(2,3) = ", end='')
a = array(2,3)
print(a)
print("a[0][0] = 5 = ", end='')
a[0][0] = 5
print(a, end='')
```

```
a = array(2,3) = [['None', 'None', 'None'], ['None', 'None', 'None']]
a[0][0] = 5 = [[5, 'None', 'None'], ['None', 'None', 'None']]
Process finished with exit code 0
```

Problem 49:

```
# Problem 49
def parse_csv(name_file):
    print("Content: ")
    f = open(str(name_file), "r")
    new_list = []
    line = f.readline()
    while line != '':
        new_line = line.strip()
        print(new_line)
        new_list.append(new_line.split('\t'))
        line = f.readline()
    f.close()
    return new_list
print(parse_csv('a.csv'), end='')
```

```
Content:
a   b   c
1   2   3
2   3   4
3   4   5
[['a', 'b', 'c'], ['1', '2', '3'], ['2', '3', '4'], ['3', '4', '5']]
Process finished with exit code 0
```

Problem 50:

```python
# Problem 50
def parse(name_file, deli1, deli2):
    print("Content: ")
    f = open(str(name_file), "r")
    new_list = []
    line = f.readline()
    while line != '':
        new_line = line.strip()
        print(new_line)
        if str(deli1) in new_line:
            new_list.append(new_line.split(str(deli1)))
        elif str(deli2) in new_line:
            new_list.append(new_line.split(str(deli1)))
        else:
            new_list.append(new_line.split())
        line = f.readline()
    f.close()
    return new_list
print(parse('a.txt', '!', '#'), end='')
```

```
Content:
a!b!c
1!2!3
2!3!4
3!4!5
[['a', 'b', 'c'], ['1', '2', '3'], ['2', '3', '4'], ['3', '4', '5']]
Process finished with exit code 0
```

Problem 51:

```
def mutate(d):
    ret=[d]
    i=0
    l=len(d)
    alp=map(chr,range(97,123))
    while i<l:
        cop=d
        ret.append(cop[:i]+cop[i+1:])
        if i<l-2:
            ret.append(cop[:i]+cop[i+1]+cop[i]+cop[i+2:])
        elif i<l-1:
            ret.append(cop[:i]+cop[i+1]+cop[i])
        for x in alp:
            ret.append(cop[:i]+x+cop[i+1:])
        for x in alp:
            ret.append(d+x)
            ret.append(x+d)
            ret.append(cop[:i]+x+cop[i:])
        i=i+1
    return ret
print("'hefllo' in mutate('hello') = ", end='')
print('hefllo' in mutate('hello'))
print("'hllo' in mutate('hello') = ", end='')
print('hllo' in mutate('hello'), end='')
```

```
'hefllo' in mutate('hello') = False
'hllo' in mutate('hello') = True
Process finished with exit code 0
```

Problem 52:

```
# Problem 52
def nearly_equal(a,b):
    value_bool = True
    a = a.lower()
    b = b.lower()
    size_a = len(a)
    size_b = len(b)
    duplicate_content = []
    count = 0
    ctr = 0
    list_ab = []
    ab = a+b
    for x in ab:
        list_ab.append(x)
    while ctr < len(list_ab):
        inr_ctr = ctr + 1
```

```
        while inr_ctr < len(list_ab):
            if list_ab[ctr] == list_ab[inr_ctr]:
                duplicate_content.append(list_ab[ctr])
            inr_ctr += 1
        ctr += 1
    if size_a == size_b:
        if size_a - len(duplicate_content) <= 1:
            value_bool = True
        else:
         value_bool = False
    elif size_a > size_b:
        if size_a - len(duplicate_content) <= 1:
            value_bool = True
        else:
         value_bool = False
    else:
        if size_b - len(duplicate_content) <= 1:
            value_bool = True
        else:
         value_bool = False
    return value_bool
print("Nearly Equal (True or False)")
print("'python', 'perl' = ", nearly_equal('python', 'perl'))
print("'pearl', 'perl' = ", nearly_equal('pearl', 'perl'))
print("'python', 'jython' = ", nearly_equal('python', 'jython'))
print("'man', 'woman' = ", nearly_equal('man', 'woman'), end='')
```

```
Nearly Equal (True or False)
'python', 'perl' =  False
'pearl', 'perl' =  True
'python', 'jython' =  True
'man', 'woman' =  False
Process finished with exit code 0
```

Problem 53:

```
def word_frequency(words):
    frequency = {}
    for w in words:
        frequency[w] = frequency.get(w, 0) + 1
    return frequency

def read_words(filename):
    print("Content of sample.txt:")
    print(open(filename).read(), "\n")
    return open(filename).read().split()
```

```
def main(filename):
    frequency = word_frequency(read_words(filename))
    for word, count in sorted(frequency.items(), key=lambda w: w[1]):
        print('%s: %d' % (word, count))
import sys
main("sample.txt")
```

```
Content of sample.txt:
word is word
sample is word


sample: 1
is: 2
word: 3


Process finished with exit code 0
```

Problem 54:

```
def word_frequency(words):
    print("Frequency: ")
    frequency = {}
    for w in words:
        for ch in w:
            frequency[ch] = frequency.get(ch, 0) + 1
    return frequency

def read_words(filename):
    print("\nContent of the file:")
    print(open(filename).read(), "\n")
    return open(filename).read().split()

def main(filename):
    frequency = word_frequency(read_words(filename))
    for word, count in sorted(frequency.items(), key=lambda w: w[1]):
        print('%s: %d' % (word, count))
import sys
import os
files = os.listdir()
print("Files = ", files)
input_file = input("Enter a file: ")
check_input_file = input_file.split('.')
print("\nFile = ",end='')
if check_input_file[-1] == 'py':
    print("Python Program File")
```

```
elif check_input_file[-1] == 'c':
    print("C Program File")
elif check_input_file[-1] == 'txt':
    print("Text File")
else:
    print("Unknown File")
main(input_file)
```

```
Files =  ['a.c', 'a.csv', 'a.txt', 'add.py', 'assignment4.2.py',
Enter a file: a.c

File = C Program File

Content of the file:
seed floor

Frequency:
s: 1
d: 1
f: 1
l: 1
r: 1
e: 2
o: 2

Process finished with exit code 0
```

Problem 55:

```
# Problem 55
def anagrams(words_list):
    ctr = 0
    new_conten = []
    while ctr < len(words_list):
        inner_ctr = 0
        inner_list = []
        while inner_ctr < len(words_list):
            word1 = sorted(words_list[ctr])
            word2 = sorted(words_list[inner_ctr])
            if word1 == word2:
                if words_list[inner_ctr] not in inner_list:
                    inner_list.append(words_list[inner_ctr])
            inner_ctr += 1
        if inner_list not in new_conten:
            new_conten.append(inner_list)
        ctr += 1
    return new_conten
```

```
print("Content =", ['eat','ate','done','tea','soup','node'])
print("Anagrams = ",anagrams(['eat','ate','done','tea','soup','node']),end='')
```

```
Content = ['eat', 'ate', 'done', 'tea', 'soup', 'node']
Anagrams =  [['eat', 'ate', 'tea'], ['done', 'node'], ['soup']]
Process finished with exit code 0
```

Problem 56:

```
# Problem 56
def valuesort(dict_words):
    new_content = []
    for i in sorted(dict_words.keys()):
        new_content.append(dict_words.get(i))
    return new_content
print("Content = {'x':1, 'y':2,'a':3}")
print("Sorted key's values = ", valuesort({'x':1, 'y':2,'a':3}), end='')
```

```
Content = {'x':1, 'y':2,'a':3}
Sorted key's values =  [3, 1, 2]
Process finished with exit code 0
```

Problem 57:

```
# Problem 57
def invertdict(words_dict):
    new_dict = {}
    for x,y in words_dict.items():
        new_dict[y] = x
    return new_dict
print("Content = {'x':1, 'y':2, 'z':3}")
print("Inverted = ", invertdict({'x':1, 'y':2, 'z':3}), end='')
```

```
Content = {'x':1, 'y':2, 'z':3}
Inverted =  {1: 'x', 2: 'y', 3: 'z'}
Process finished with exit code 0
```