**TEXT HANDLING**

Python Program to Replace all Occurrences of ‘a’ with $ in a String

This is a Python Program to replace all occurrences of ‘a’ with ‘$’ in a string.

**Problem Description**

The program takes a string and replaces all occurrences of ‘a’ with ‘$’.

**Problem Solution**

1. Take a string and store it in a variable.  
2. Using the replace function, replace all occurrences of ‘a’ and ‘A’ with ‘$’ and store it back in the variable.  
3. Print the modified string.  
4. Exit.

**Program Explanation**

**Runtime Test Cases**

Case 1:

Enter string:Apple

Modified string:

$pple

Case 2:

Enter string:Asia

Modified string:

$si$

Python Program to Remove the nth Index Character from a Non-Empty String

This is a Python Program to remove the nth index character from a non-empty string.

**Problem Description**

The program takes a string and removes the nth index character from the non-empty string.

**Problem Solution**

1. Take a string from the user and store it in a variable.  
2. Take the index of the character to remove.  
3. Pass the string and the index as arguments to a function named remove.  
4. In the function, the string should then be split into two halves before the index character and after the index character.  
5. These two halves should then be merged together.  
6. Print the modified string.  
7. Exit.

**Runtime Test Cases**

Case 1:

Enter the sring:Hello

Enter the index of the character to remove:3

Modified string:

Helo

Case 2:

Enter the sring:Checking

Enter the index of the character to remove:4

Modified string:

Checing

Python Program to Form a New String where the First Character and the Last Character have been Exchanged

This is a Python Program to form a string where the first character and the last character have been exchanged.

**Problem Description**

The program takes a string and swaps the first character and the last character of the string.

**Problem Solution**

1. Take a string from the user and store it in a variable.  
2. Pass the string as an argument to a function.  
3. In the function, split the string.  
4. Then add the last character to the middle part of the string which is in turn added to the first character.  
5. Print the modified string.  
6. Exit.

**Runtime Test Cases**

Case 1:

Enter string:abcd

Modified string:

dbca

Case 2:

Enter string:hello world

Modified string:

dello worlh

Python Program to Count the Number of Vowels in a String

This is a Python Program to count the number of vowels in a string.

**Problem Description**

The program takes a string and counts the number of vowels in a string.

**Problem Solution**

1. Take a string from the user and store it in a variable.  
2. Initialize a count variable to 0.  
3. Use a for loop to traverse through the characters in the string.  
4. Use an if statement to check if the character is a vowel or not and increment the count variable if it is a vowel.  
5. Print the total number of vowels in the string.  
6. Exit.

**Runtime Test Cases**

Case 1:

Enter string:Hello world

Number of vowels are:

3

Case 2:

Enter string:WELCOME

Number of vowels are:

3

Python Program to Take in a String and Replace Every Blank Space with Hyphen

This is a Python Program to take a string and replace every blank space with a hyphen.

**Problem Description**

The program takes a string and replaces every blank space with a hyphen.

**Problem Solution**

1. Take a string and store it in a variable.  
2. Using the replace function, replace all occurrences of ‘ ‘ with ‘-‘ and store it back in the variable.  
3. Print the modified string.  
4. Exit.

**Runtime Test Cases**

Case 1:

Enter string:hello world

Modified string:

hello-world

Case 2:

Enter string:apple orange banana

Modified string:

apple-orange-banana

Python Program to Calculate the Length of a String Without Using a Library Function

This is a Python Program to calculate the length of a string without using library functions.

**Problem Description**

The program takes a string and calculates the length of the string without using library functions.

**Problem Solution**

1. Take a string from the user and store it in a variable.  
2. Initialize a count variable to 0.  
3. Use a for loop to traverse through the characters in the string and increment the count variable each time.  
4. Print the total count of the variable.  
5. Exit.

5. The total count of characters in the string which is the length of the string is printed.

**Runtime Test Cases**

Case 1:

Enter string:Hello

Length of the string is:

5

Case 2:

Enter string:Bangalore

Length of the string is:

9

Python Program to Calculate the Number of Words and the Number of Characters Present in a String

This is a Python Program to calculate the number of words and characters present in a string.

**Problem Description**

The program takes a string and calculates the number of words and characters present in the string.

**Problem Solution**

1. Take a string from the user and store it in a variable.  
2. Initialize the character count variable to 0 and the word count variable to 1.  
3. Use a for loop to traverse through the characters in the string and increment the character count variable each time.  
4. Increment the word count variable only if a space is encountered.  
5. Print the total count of the characters and words in the string.  
6. Exit.

**Runtime Test Cases**

Case 1:

Enter string:Hello world

Number of words in the string:

2

Number of characters in the string:

11

Case 2:

Enter string:I love python

Number of words in the string:

3

Number of characters in the string:

13

Python Program to Calculate the Number of Digits and Letters in a String

This is a Python Program to calculate the number of digits and letters in a string.

**Problem Description**

The program takes a string and calculates the number of digits and letters in a string.

**Problem Solution**

1. Take a string from the user and store it in a variable.  
2. Initialize the two count variables to 0.  
3. Use a for loop to traverse through the characters in the string and increment the first count variable each time a digit is encountered and increment the second count variable each time a character is encountered.  
4. Print the total count of both the variables.  
5. Exit.

**Runtime Test Cases**

Case 1:

Enter string:Hello123

The number of digits is:

3

The number of characters is:

8

Case 2:

Enter string:Abc12

The number of digits is:

2

The number of characters is:

5

Python Program to Check if a Substring is Present in a Given String

This is a Python Program to check if a substring is present in a given string.

**Problem Description**

The program takes a string and checks if a substring is present in the given string.

**Problem Solution**

1. Take a string and a substring from the user and store it in separate variables.  
2. Check if the substring is present in the string using find() in-built function.  
3. Print the final result.  
4. Exit.

**Runtime Test Cases**

Case 1:

Enter string:Hello world

Enter word:world

Substring in string!

Case 2:

Enter string:Hello world

Enter word:apple

Substring not found in string!

**FILES**

Python Program to Read the Contents of a File

This is a Python Program to read the contents of a file.

**Problem Description**

The program takes the file name from the user and reads the contents of that file.

**Problem Solution**

1. Take the file name from the user.  
2. Use readline() function for the first line first.  
3. Use a while loop to print the first line and then read the remaining lines and print it till the end of file.  
4. Exit.

**Runtime Test Cases**

Case 1:

Contents of file:

Hello world

Output:

Enter the name of the file with .txt extension: data1.txt

Case 2:

Contents of file:

This programming language is

Python.

Output:

Enter the name of the file with .txt extension: data2.txt

This programming language is

Python.

Python Program to Count the Number of Words in a Text File

This is a Python Program to count the number of words in a text file.

**Problem Description**

The program takes the file name from the user and counts number of words in that file.

**Problem Solution**

1. Take the file name from the user.  
2. Read each line from the file and split the line to form a list of words.  
3. Find the length of items in the list and print it.  
4. Exit.

**Runtime Test Cases**

Case 1:

Contents of file:

Hello world

Output:

Enter file name: data1.txt

Number of words:

2

Case 2:

Contents of file:

This programming language is

Python

Output:

Enter file name: data2.txt

Number of words:

5

Python Program to Count the Number of Lines in a Text File

This is a Python Program to count the number of lines in a text file.

**Problem Description**

The program takes the file name from the user and counts number of lines in that file.

**Problem Solution**

1. Take the file name from the user.  
2. Read each line from the file and increment the count variable  
3. Print the line count.  
4. Exit.

**Runtime Test Cases**

Case 1:

Contents of file:

Hello world

Output:

Enter file name: data1.txt

Number of lines:

1

Case 2:

Contents of file:

This programming language is

Python

Output:

Enter file name: data2.txt

Number of lines:

2

Python Program to Read a String from the User and Append it into a File

This is a Python Program to read a string from the user and appends it into a file

**Problem Description**

The program takes a string from the user and appends the string into an existing file.

**Problem Solution**

1. Take the file name from the user.  
2. Open the file in append mode.  
2. Take in a string from the user, append it to the existing file and close the file.  
3. Open the file again in read mode and display the contents of the file.  
4. Exit.

**Runtime Test Cases**

Case 1:

Contents of file:

Hello world

Output:

Enter file name: test.txt

Enter string to append:

test

Contents of appended file:

Hello world

test

Case 2:

Contents of file:

This programming language is

Python

Output:

Enter file name: test1.txt

Enter string to append:

test

Contents of appended file:

This programming language is

Python

test

Python Program to Copy the Contents of One File into Another

This is a Python Program to copy the contents of one file into another.

**Problem Description**

The program copies the contents of one file and writes it into another.

**Problem Solution**

1. Open one file called test.txt in read mode.  
2. Open another file out.txt in write mode.  
3. Read each line from the input file and write it into the output file.  
4. Exit.

.

**Runtime Test Cases**

Case 1:

Contents of file(test.txt):

Hello world

Output(out.text):

Hello world

Case 2:

Contents of file(test.txt):

Sanfoundry

Output(out.text):

Sanfoundry

#### Searching and sorting

## BUBBLE SORT

The **bubble sort** makes multiple passes through a list. It compares adjacent items and exchanges those that are out of order. Each pass through the list places the next largest value in its proper place. In essence, each item “bubbles” up to the location where it belongs.

[Figure 1](http://interactivepython.org/courselib/static/pythonds/SortSearch/TheBubbleSort.html#fig-bubblepass) shows the first pass of a bubble sort. The shaded items are being compared to see if they are out of order. If there are *n* items in the list, then there are n−1n−1 pairs of items that need to be compared on the first pass. It is important to note that once the largest value in the list is part of a pair, it will continually be moved along until the pass is complete.



At the start of the second pass, the largest value is now in place. There are n−1n−1 items left to sort, meaning that there will be n−2n−2 pairs. Since each pass places the next largest value in place, the total number of passes necessary will be n−1n−1. After completing the n−1n−1 passes, the smallest item must be in the correct position with no further processing required. [ActiveCode 1](http://interactivepython.org/courselib/static/pythonds/SortSearch/TheBubbleSort.html" \l "lst-bubble) shows the complete bubbleSortfunction. It takes the list as a parameter, and modifies it by exchanging items as necessary.

The exchange operation, sometimes called a “swap,” is slightly different in Python than in most other programming languages. Typically, swapping two elements in a list requires a temporary storage location (an additional memory location). A code fragment such as

temp = alist[i]

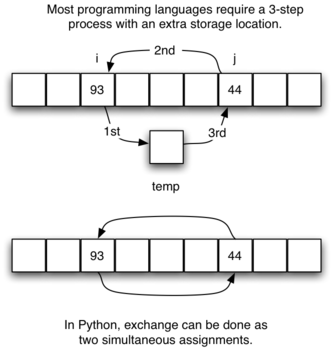
alist[i] = alist[j]

alist[j] = temp

will exchange the ith and jth items in the list. Without the temporary storage, one of the values would be overwritten.

In Python, it is possible to perform simultaneous assignment. The statement a,b=b,a will result in two assignment statements being done at the same time (see [Figure 2](http://interactivepython.org/courselib/static/pythonds/SortSearch/TheBubbleSort.html#fig-pythonswap)). Using simultaneous assignment, the exchange operation can be done in one statement.

Lines 5-7 in [ActiveCode 1](http://interactivepython.org/courselib/static/pythonds/SortSearch/TheBubbleSort.html" \l "lst-bubble) perform the exchange of the ii and (i+1)th(i+1)th items using the three–step procedure described earlier. Note that we could also have used the simultaneous assignment to swap the items.



## Selection Sort Algorithm

The Selection Sort algorithm is based on successive selection of minima or maxima values. Assume that we have a [list](http://code.tutsplus.com/tutorials/a-smooth-refresher-to-pythons-lists--cms-25286) which we want to sort in ascending order (from smaller to larger values). The smallest element will be at the beginning of the list, and the largest element will be at the end of the list.

Let's say that the original list looks as follows:

| 7 | 5 | 3.5 | 4 | 3.1 |

The first thing we do is find the minimum value in the list, which is in our case 3.1.

After finding the minimum value, swap that minimum value with the first element in the list. That is, swap 3.1 with 7. The list will now look as follows:

| 3.1 | 5 | 3.5 | 4 | 7 |

Now that we are certain that the first element is in the correct position in the list, we repeat the above step (finding the minimum value) starting from the second element in the list. We can find that the minimum value in the list (starting from the second element) is 3.5. We thus now swap 3.5 with 5. The list now becomes as follows:

| 3.1 | 3.5 | 5 | 4 | 7 |

At this point, we are certain that the first element and the second element are in their correct positions.

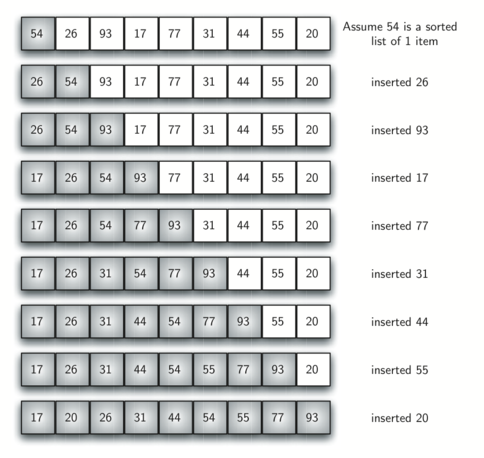
Now, we check the minimum value in the remainder of the list, that is starting from the third element 5. The minimum value in the remainder of the list is 4, and we now swap it with 5. The list thus becomes as follows:

| 3.1 | 3.5 | 4 | 5 | 7 |

So we are now certain that the first three elements are in the correct positions, and the process continues that way.

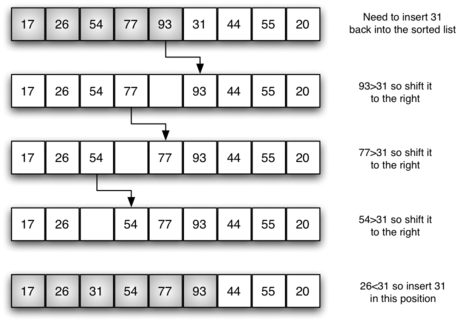
**The Insertion Sort**

The **insertion sort**, although still O(n2)O(n2), works in a slightly different way. It always maintains a sorted sublist in the lower positions of the list. Each new item is then “inserted” back into the previous sublist such that the sorted sublist is one item larger. [Figure 4](http://interactivepython.org/courselib/static/pythonds/SortSearch/TheInsertionSort.html#fig-insertionsort) shows the insertion sorting process. The shaded items represent the ordered sublists as the algorithm makes each pass.



We begin by assuming that a list with one item (position 00) is already sorted. On each pass, one for each item 1 through n−1n−1, the current item is checked against those in the already sorted sublist. As we look back into the already sorted sublist, we shift those items that are greater to the right. When we reach a smaller item or the end of the sublist, the current item can be inserted.

[Figure 5](http://interactivepython.org/courselib/static/pythonds/SortSearch/TheInsertionSort.html#fig-insertionpass) shows the fifth pass in detail. At this point in the algorithm, a sorted sublist of five items consisting of 17, 26, 54, 77, and 93 exists. We want to insert 31 back into the already sorted items. The first comparison against 93 causes 93 to be shifted to the right. 77 and 54 are also shifted. When the item 26 is encountered, the shifting process stops and 31 is placed in the open position. Now we have a sorted sublist of six items.



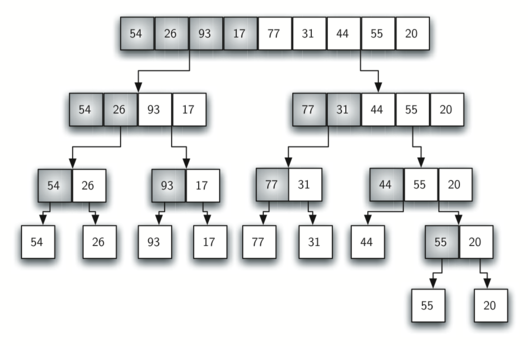
The implementation of insertionSort ([ActiveCode 1](http://interactivepython.org/courselib/static/pythonds/SortSearch/TheInsertionSort.html" \l "lst-insertion)) shows that there are again n−1n−1 passes to sort *n*items. The iteration starts at position 1 and moves through position n−1n−1, as these are the items that need to be inserted back into the sorted sublists. Line 8 performs the shift operation that moves a value up one position in the list, making room behind it for the insertion. Remember that this is not a complete exchange as was performed in the previous algorithms.

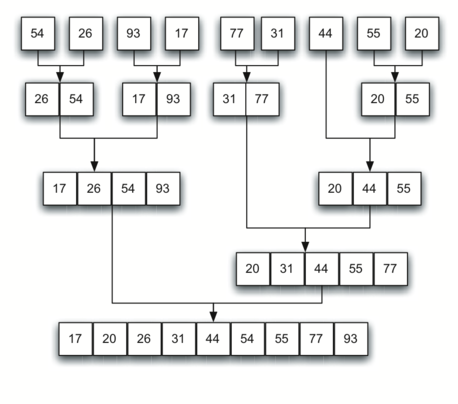
The maximum number of comparisons for an insertion sort is the sum of the first n−1n−1 integers. Again, this is O(n2)O(n2). However, in the best case, only one comparison needs to be done on each pass. This would be the case for an already sorted list.

One note about shifting versus exchanging is also important. In general, a shift operation requires approximately a third of the processing work of an exchange since only one assignment is performed. In benchmark studies, insertion sort will show very good performance.

The Merge Sort

We now turn our attention to using a divide and conquer strategy as a way to improve the performance of sorting algorithms. The first algorithm we will study is the **merge sort**. Merge sort is a recursive algorithm that continually splits a list in half. If the list is empty or has one item, it is sorted by definition (the base case). If the list has more than one item, we split the list and recursively invoke a merge sort on both halves. Once the two halves are sorted, the fundamental operation, called a **merge**, is performed. Merging is the process of taking two smaller sorted lists and combining them together into a single, sorted, new list. [Figure 10](http://interactivepython.org/courselib/static/pythonds/SortSearch/TheMergeSort.html#fig-mergesorta) shows our familiar example list as it is being split by mergeSort. [Figure 11](http://interactivepython.org/courselib/static/pythonds/SortSearch/TheMergeSort.html#fig-mergesortb) shows the simple lists, now sorted, as they are merged back together.





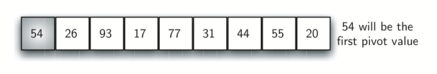
The mergeSort function shown in [ActiveCode 1](http://interactivepython.org/courselib/static/pythonds/SortSearch/TheMergeSort.html" \l "lst-merge) begins by asking the base case question. If the length of the list is less than or equal to one, then we already have a sorted list and no more processing is necessary. If, on the other hand, the length is greater than one, then we use the Python slice operation to extract the left and right halves. It is important to note that the list may not have an even number of items. That does not matter, as the lengths will differ by at most one.

**The Quick Sort**

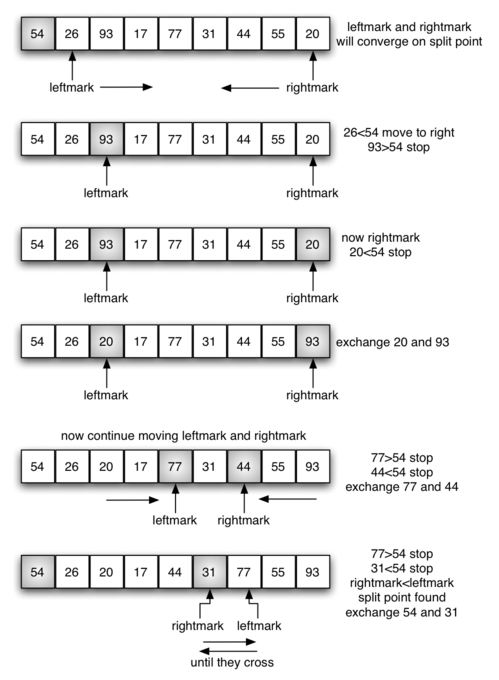
The **quick sort** uses divide and conquer to gain the same advantages as the merge sort, while not using additional storage. As a trade-off, however, it is possible that the list may not be divided in half. When this happens, we will see that performance is diminished.

A quick sort first selects a value, which is called the **pivot value**. Although there are many different ways to choose the pivot value, we will simply use the first item in the list. The role of the pivot value is to assist with splitting the list. The actual position where the pivot value belongs in the final sorted list, commonly called the **split point**, will be used to divide the list for subsequent calls to the quick sort.

[Figure 12](http://interactivepython.org/courselib/static/pythonds/SortSearch/TheQuickSort.html#fig-splitvalue) shows that 54 will serve as our first pivot value. Since we have looked at this example a few times already, we know that 54 will eventually end up in the position currently holding 31. The **partition**process will happen next. It will find the split point and at the same time move other items to the appropriate side of the list, either less than or greater than the pivot value.

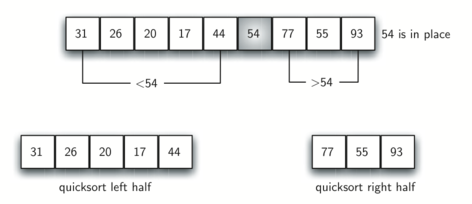


Partitioning begins by locating two position markers—let’s call them leftmark and rightmark—at the beginning and end of the remaining items in the list (positions 1 and 8 in [Figure 13](http://interactivepython.org/courselib/static/pythonds/SortSearch/TheQuickSort.html#fig-partitiona)). The goal of the partition process is to move items that are on the wrong side with respect to the pivot value while also converging on the split point. [Figure 13](http://interactivepython.org/courselib/static/pythonds/SortSearch/TheQuickSort.html#fig-partitiona) shows this process as we locate the position of 54.



We begin by incrementing leftmark until we locate a value that is greater than the pivot value. We then decrement rightmark until we find a value that is less than the pivot value. At this point we have discovered two items that are out of place with respect to the eventual split point. For our example, this occurs at 93 and 20. Now we can exchange these two items and then repeat the process again.

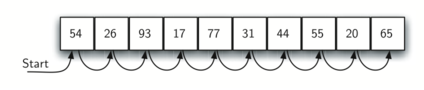
At the point where rightmark becomes less than leftmark, we stop. The position of rightmark is now the split point. The pivot value can be exchanged with the contents of the split point and the pivot value is now in place ([Figure 14](http://interactivepython.org/courselib/static/pythonds/SortSearch/TheQuickSort.html#fig-partitionb)). In addition, all the items to the left of the split point are less than the pivot value, and all the items to the right of the split point are greater than the pivot value. The list can now be divided at the split point and the quick sort can be invoked recursively on the two halves.



The Sequential Search

When data items are stored in a collection such as a list, we say that they have a linear or sequential relationship. Each data item is stored in a position relative to the others. In Python lists, these relative positions are the index values of the individual items. Since these index values are ordered, it is possible for us to visit them in sequence. This process gives rise to our first searching technique, the **sequential search**.

[Figure 1](http://interactivepython.org/courselib/static/pythonds/SortSearch/TheSequentialSearch.html#fig-seqsearch) shows how this search works. Starting at the first item in the list, we simply move from item to item, following the underlying sequential ordering until we either find what we are looking for or run out of items. If we run out of items, we have discovered that the item we were searching for was not present.

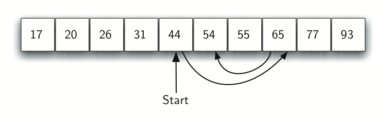


The Python implementation for this algorithm is shown in [CodeLens 1](http://interactivepython.org/courselib/static/pythonds/SortSearch/TheSequentialSearch.html" \l "lst-seqsearchpython). The function needs the list and the item we are looking for and returns a boolean value as to whether it is present. The boolean variable foundis initialized to False and is assigned the value True if we discover the item in the list.

The Binary Search

It is possible to take greater advantage of the ordered list if we are clever with our comparisons. In the sequential search, when we compare against the first item, there are at most n−1n−1 more items to look through if the first item is not what we are looking for. Instead of searching the list in sequence, a **binary search** will start by examining the middle item. If that item is the one we are searching for, we are done. If it is not the correct item, we can use the ordered nature of the list to eliminate half of the remaining items. If the item we are searching for is greater than the middle item, we know that the entire lower half of the list as well as the middle item can be eliminated from further consideration. The item, if it is in the list, must be in the upper half.

We can then repeat the process with the upper half. Start at the middle item and compare it against what we are looking for. Again, we either find it or split the list in half, therefore eliminating another large part of our possible search space. [Figure 3](http://interactivepython.org/courselib/static/pythonds/SortSearch/TheBinarySearch.html#fig-binsearch) shows how this algorithm can quickly find the value 54. The complete function is shown in [CodeLens 3](http://interactivepython.org/courselib/static/pythonds/SortSearch/TheBinarySearch.html" \l "lst-binarysearchpy).



#### Recursions

Python Program to Determine Whether a Given Number is Even or Odd Recursively

This is a Python Program to determine whether a given number is even or odd recursively.

**Problem Description**

The program takes a number and determines whether a given number is even or odd recursively.

**Problem Solution**

1. Take a number from the user and store it in a variable.  
2. Pass the number as an argument to a recursive function.  
3. Define the base condition as the number to be lesser than 2.  
4. Otherwise call the function recursively with the number minus 2.  
5. Then return the result and check if the number is even or odd.  
6. Print the final result.  
7. Exit.

**Program Explanation**

1. User must enter a number and store it in a variable.  
2. The number is passed as an argument to a recursive function.  
3. The base condition is that the number has to be lesser than 2.  
4. Otherwise the function is called recursively with the number minus 2.  
5. The result is returned and an if statement is used to check if the number is odd or even.  
6. The final result is printed.

**Runtime Test Cases**

Case 1:

Enter number:124

Number is even!

Case 2:

Enter number:567

Number is odd!

Python Program to Determine How Many Times a Given Letter Occurs in a String Recursively

This is a Python Program to determine how many times a given letter occurs in a string recursively.

**Problem Description**

The program takes a string and determines how many times a given letter occurs in a string recursively.

**Problem Solution**

1. Take a string and a character from the user and store it in different variables.  
2. Pass the string and the characters as arguments to a recursive function.  
3. Pass the base condition that the string isn’t empty.  
4. Check if the first character of the string is equal to the character taken from the user and if it is equal, increment the count.  
5. Progress the string either wise and print the number of times the letter occurs in the string.  
6. Exit.

**Runtime Test Cases**

Case 1:

Enter string:abcdab

Enter character to check:b

Count is:

2

Case 2:

Enter string:hello world

Enter character to check:l

Count is:

3

Python Program to Find the Fibonacci Series Using Recursion

This is a Python Program to find the fibonacci series using recursion.

**Problem Description**

The program takes the number of terms and determines the fibonacci series using recursion upto that term.

**Problem Solution**

1. Take the number of terms from the user and store it in a variable.  
2. Pass the number as an argument to a recursive function named fibonacci.  
3. Define the base condition as the number to be lesser than or equal to 1.  
4. Otherwise call the function recursively with the argument as the number minus 1 added to the function called recursively with the argument as the number minus 2.  
5. Use a for loop and print the returned value which is the fibonacci series.  
6. Exit.

**Runtime Test Cases**

Case 1:

Enter number of terms:5

Fibonacci sequence:

0 1 1 2 3

Case 2:

Enter number of terms:7

Fibonacci sequence:

0 1 1 2 3 5 8

Python Program to Find the Factorial of a Number Using Recursion

This is a Python Program to find the factorial of a number using recursion.

**Problem Description**

The program takes a number and determines the factorial of the number using recursion.

**Problem Solution**

1. Take a number from the user and store it in a variable.  
2. Pass the number as an argument to a recursive factorial function.  
3. Define the base condition as the number to be lesser than or equal to 1 and return 1 if it is.  
4. Otherwise call the function recursively with the number minus 1 multiplied by the number itself.  
5. Then return the result and print the factorial of the number.  
6. Exit.

**Runtime Test Cases**

Case 1:

Enter number:5

Factorial:

120

Case 2:

Enter number:9

Factorial:

362880