Problem Statement

Write a python function, **encrypt\_sentence()** which accepts a message and encrypts it based on rules given below and returns the encrypted message.  
  
Words at odd position -> Reverse It  
Words at even position -> Rearrange the characters so that all consonants appear before the vowels and their order should not change  
  
**Note:**

1. Assume that the sentence would begin with a word and there will be only a single space between the words.
2. Perform case sensitive string operations wherever necessary.

|  |  |
| --- | --- |
| **Sample Input** | **Expected Output** |
| the sun rises  in the east | eht snu sesir  ni eht stea |

Code -

def Test(message):

words = message.split()

encrypted\_message = []

for i, word in enumerate(words, 1):

if i % 2 != 0:

encrypted\_message.append(word[::-1])

else:

consonants = ''.join([c for c in word if c.lower() not in 'aeiou'])

vowels = ''.join([c for c in word if c.lower() in 'aeiou'])

encrypted\_message.append(consonants + vowels)

return ' '.join(encrypted\_message)

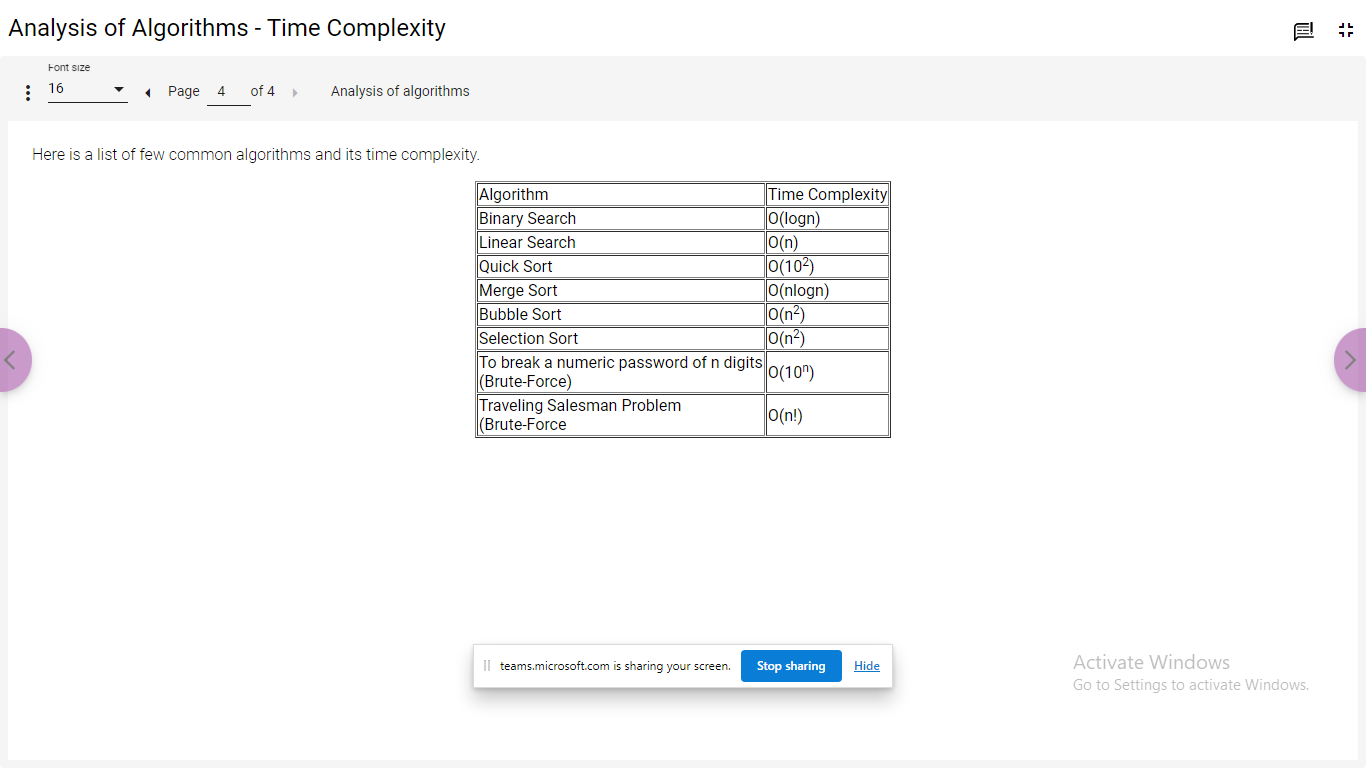
# Test the function

print(Test("the sun rises"))

print(Test("in the east"))

//===========================================================

Complexity -



//=============================================================  
Problem Statement

Let’s see how to create a list in Python and perform some operations on it.

**Creating a list:**

|  |  |  |
| --- | --- | --- |
| Creating an empty list | sample\_list=[] |  |
| Creating a list with known size and known elements | sample\_list1=["Mark",5,"Jack",9, "Chan",5] sample\_list2=["Mark","Jack", "Chan"] | List can store both homogeneous and heterogeneous elements |
| Creating a list with known size and unknown elements | sample\_list=[None]\*5 | None denotes an unknown value in Python |
| Length of the list | len(sample\_list) | Displays the number of elements in the list |

**Random access of elements:**

|  |  |  |
| --- | --- | --- |
| Random read | print(sample\_list[2]) |  |
| Random write | sample\_list[2]=“James” | List is mutable i.e., the above statement will rewrite the existing value at index position 2 with “James”. |

**Other operations:**

|  |  |  |
| --- | --- | --- |
| Adding an element to the end of the list | sample\_list.append("James") | List need not have a fixed size, it can grow dynamically |
| Concatenating two lists | new\_list=["Henry","Tim"]  sample\_list+=new\_list sample\_list=sample\_list+new\_list | sample\_list+=new\_list, concatenates new\_list to sample\_list  sample\_list=sample\_list+new\_list, creates a new list named sample\_list containing the concatenated elements from the original sample\_list and new\_list |

Code in Python 3

 #Random write

sample\_list[2]="James"

#Random read

print("Element at 2nd index position after random write:",sample\_list[2])

#Adding an element to list

sample\_list.append("James")

print("After adding element to list:",sample\_list)

#Combining two lists

new\_list=["Henry","Tim"]

sample\_list+=new\_list

#Adds Henry and Tim to the existing sample\_list

print("After combining two lists - 1st way:",sample\_list)

#Another way to combine two lists

sample\_list=sample\_list+new\_list

#Adds Henry and Tim to the new sample\_list

print("After combining two lists - 2nd way:",sample\_list)

#Accessing an element beyond the total number of elements in the list

print(sample\_list[11])

#Will give you IndexError  
  
  
//------------------------------------------------------------------------------

ticket\_list=[2345,5667,7889,5455]

print(ticket\_list)

print(ticket\_list[1])

ticket\_list[1]=9090

print(ticket\_list)

sample\_list=[]

print(sample\_list)

print(len(sample\_list))

sample\_list=[10]\*5

print(sample\_list)

list1=[1,2,3]

list2=[4,5,6]

list3=list1+list2

print(list3)  
//------------------------------------------------------------------------  
Q1 of 5outlined\_flag

Given the list list1 = ["e","d","u","c","a","t","i","o","n"]  
What expression will result in the list ["c", "a", "t"] ?

list1[2:5]

list1[3:6]

list1[5:2]

list1[3:5]

**Answer - list1[3:6]**

//---------------------------------------------------------------

Q2 of 5outlined\_flag

What is the output of the following code snippet?

num\_list = [100.5,30.465,-1.22,20.15]

num\_list.insert(1, -100.5)

num\_list.pop(0)

num\_list.sort()

print(num\_list[0])

100.5

-100.5

30.465

**Answer -** -100.5

//------------------------------------------------------------

Q3 of 5outlined\_flag

Which among the following statements may result in an error?

Assume that the statements are executed in the order in which it is written.a. list1=[5,10,15,20,25]  
b. print(len(list1))  
c. print(list1[4])  
d. print(list1[5])  
e. print(list1[4:5])  
f. list1[2]=12  
g. print(list1)  
h. list1=list1+[8,9]

b

d

E

h

**Answer - option D (**print(list1[5])**)**

//--------------------------------------------------------

Q4 of 5outlined\_flag

What is the output of the following code snippet?

[0, 0, 0, 0, 0]

[10, 20, 30, 40, 50]

[0, 0, 10, 20, 30]

[0, 10, 20, 30, 40]

**Answer -** [0, 10, 20, 30, 40]

//---------------------------------------------------------

Q5 of 5outlined\_flag

How many comparisons will take place before flag becomes 1?

num\_list = [1,33,31,5,26,7,8,92,10]

num = 7

flag = 0

for item in num\_list:

   if(item == num):

       flag = 1

   else:

       continue

if(flag == 1):

   print(num, "found in the list")

else:

   print(num, "NOT found in the list")

6

5

1

9

**Answer - 6**

//-------------------------------------------------------------------------

Problem Statement

Write a Python program to generate the next 15 leap years starting from a given year. Populate the leap years into a list and display the list.

Code -

def leap\_years(start\_year):

leap\_years = []

year = start\_year

while len(leap\_years) < 15:

if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):

leap\_years.append(year)

year += 1

return leap\_years

# Test the function

start\_year = int(input("Enter starting year: "))

leap\_years\_list = leap\_years(start\_year)

print("Next 15 leap years starting from", start\_year, ":", leap\_years\_list)

//-----------------------------------------------------------------------

ARS Gems Store sells different varieties of gems to its customers.  
  
Write a Python program to calculate the bill amount to be paid by a customer based on the list of gems and quantity purchased. Any purchase with a total bill amount above Rs.30000 is entitled for 5% discount. If any gem required by the customer is not available in the store, then consider total bill amount to be -1.  
  
Assume that quantity required by the customer for any gem will always be greater than 0.

#lex\_auth\_012693795044450304151

def calculate\_bill\_amount(gems\_list, price\_list, reqd\_gems,reqd\_quantity):  
    bill\_amount=0  
    #Write your logic here  
    return bill\_amount

#List of gems available in the store  
gems\_list=["Emerald","Ivory","Jasper","Ruby","Garnet"]

#Price of gems available in the store. gems\_list and price\_list have one-to-one correspondence  
price\_list=[1760,2119,1599,3920,3999]

#List of gems required by the customer  
reqd\_gems=["Ivory","Emerald","Garnet"]

#Quantity of gems required by the customer. reqd\_gems and reqd\_quantity have one-to-one correspondence  
reqd\_quantity=[3,10,12] bill\_amount=calculate\_bill\_amount(gems\_list, price\_list, reqd\_gems, reqd\_quantity)  
print(bill\_amount)

**Code -**

def calculate\_bill\_amount(gems\_list, price\_list, reqd\_gems, reqd\_quantity):

total\_bill = 0

for gem, quantity in zip(reqd\_gems, reqd\_quantity):

if gem in gems\_list:

index = gems\_list.index(gem)

total\_bill += price\_list[index] \* quantity

else:

return -1

if total\_bill > 30000:

total\_bill -= total\_bill \* 0.05

return total\_bill

# List of gems available in the store

gems\_list = ["Emerald", "Ivory", "Jasper", "Ruby", "Garnet"]

# Price of gems available in the store. gems\_list and price\_list have one-to-one correspondence

price\_list = [1760, 2119, 1599, 3920, 3999]

# List of gems required by the customer

reqd\_gems = ["Ivory", "Emerald", "Garnet"]

# Quantity of gems required by the customer. reqd\_gems and reqd\_quantity have one-to-one correspondence

reqd\_quantity = [3, 10, 12]

bill\_amount = calculate\_bill\_amount(gems\_list, price\_list, reqd\_gems, reqd\_quantity)

print("Total bill amount:", bill\_amount)

//----------------------------------------------------------------------------

def create\_largest\_number(number\_list):  
    pass  
    #remove pass and write your logic here

number\_list=[23,45,67]  
largest\_number=create\_largest\_number(number\_list)  
print(largest\_number)

**Code -**

def create\_largest\_number(number\_list):

# Convert numbers to strings and sort them based on custom comparison

number\_list = sorted(map(str, number\_list), key=lambda x: x\*3, reverse=True)

# Concatenate the sorted numbers and return

return int(''.join(number\_list))

number\_list = [23, 45, 67]

largest\_number = create\_largest\_number(number\_list)

print(largest\_number)

//----------------------------------------------------------

**Set -**

A={1,2,3,4,5}

B={1,2,3,6,7}

print(A&B)

print(A-B)

print(B-A)

print(A|B)

//----------------------------------------------------

A set is an unordered group of values with no duplicate entries. Set can be created by using the keyword set or by using curly braces {}. set function is used to eliminate duplicate values in a list.

|  |  |  |
| --- | --- | --- |
| Creating a set | flight\_set={500,520,600,345,520,634,600,500,200,200} | Removes the duplicates from the given group of values to create the set |
| Eliminating duplicates from a list | passengers\_list=["George", "Annie", "Jack", "Annie", "Henry", "Helen", "Maria", "George", "Jack", "Remo"] unique\_passengers=set(passengers\_list) | set function - removes the duplicates from the list and returns a set |
| Common elements between setA and setB | setA & setB | Creates a new set which has common elements from setA and setB |
| Elements that are only in setA | setA - setB | Creates a new set which has only unique elements of setA |
| Merges elements of setA and setB | setA | setB | Creates a new set which has all the elements of setA and setB |

 //-------------------------------------------------------------

#list of passengers  
passengers\_list=["George","Annie", "Jack","Annie","Henry", "Helen","Maria","George","Jack","Remo"]

#set function - removes the duplicates from the list and returns a set  
unique\_passengers=set(passengers\_list)  
print(unique\_passengers)

#creating a set  
flight\_set={500,520,600,345,520,634,600,500,200,200}  
print(flight\_set)

flights\_at\_src = ["AI230","BA944","EM395","AI704","BA944","AI704"]  
flights\_at\_dest = ["SI107","AI034","EM395","AI704","BA802","SI236"]  
print(flights\_at\_src)  
print(flights\_at\_dest)

#Creating list of unique flights at source and destination  
uniq\_src\_flights = set(flights\_at\_src)  
uniq\_dest\_flights = set(flights\_at\_dest)  
print(uniq\_src\_flights)  
print(uniq\_dest\_flights)

#setA-setB -> Gives the elements that are only in setA  
#List of flights only at source airport  
flights\_only\_at\_src = uniq\_src\_flights-uniq\_dest\_flights  
print(flights\_only\_at\_src)

#setA&setB -> Gives the common elements between setA and setB  
#List of flights common to source and destination airports  
common\_flights=uniq\_src\_flights&uniq\_dest\_flights  
print(common\_flights)

#setA|setB -> merges setA and setB after removing duplicates  
#List of all flights at source and destination airports  
all\_flights=uniq\_src\_flights|uniq\_dest\_flights  
print(all\_flights)

//---------------------------------------------------------------------------------

**Implicit and Explicit Conversion -**

Similarly, one has to be careful in explicit conversions as well. For example,

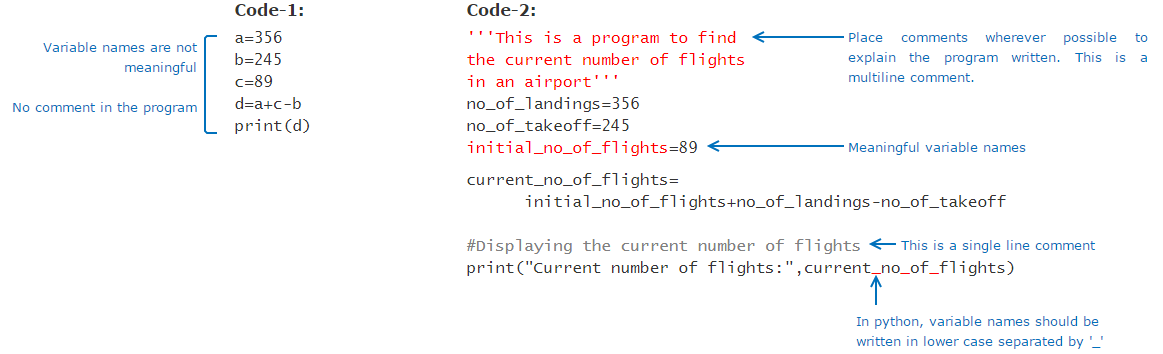
1. Converting a floating point value to integer would result in loss of decimal point values.
2. A larger data type if converted to smaller data type will result in loss of data as the number will be truncated.

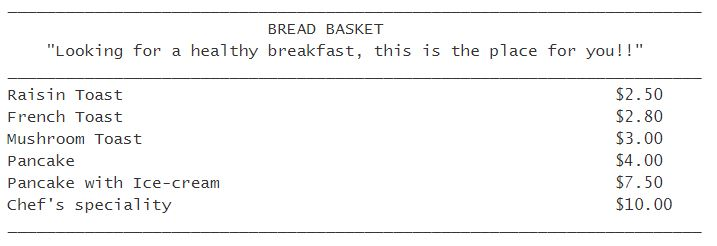
|  |  |
| --- | --- |
| **Conversion** | **Python** |
| Conversion to int | int()  Example: num=int(“10”) Value of num will be 10 |
| Conversion to string | str()  Example: num=str(10) Value of num will be “10” |

**Code** -

 Here are two programs implementing the same functionality.

|  |  |
| --- | --- |
| **Code 1:** | **Code 2:** |
| a=356b=245c=89d=a+c-bprint(d) | '''This is a program to find thecurrent number of flights in an airport'''landings\_count=356takeoffs\_count=245initial\_flights=89current\_flights = initial\_flights + landings\_count - takeoffs\_count#Displaying the current number of flightsprint("Current number of flights:",current\_flights) |





//------------------------------------------------------------------------------

|  |  |  |
| --- | --- | --- |
| **Operation** | **Example** | **Description** |
| Creating a tuple | lunch\_menu=("Welcome Drink","Veg Starter","Non-Veg Starter","Veg Main Course","Non-Veg Main Course","Dessert") | () are optional, a set of values separated by comma is also considered to be a tuple.  sample\_tuple="A","B","C" Although () are optional, it is a good practice to have them for readability of code.  If we need to create a tuple with a single element, then we need to include a comma as shown below: sample\_tuple=("A",) |
| Random Write | lunch\_menu[0]="" | This will result in an error as tuple is immutable. Hence random write is not possible in tuple. |

 //---------------------------------------------------------------------  
  
[3:21 PM] Ashutosh (Unverified)

Write a python function to add 'ing' at the end of a given string and return the new string.   
If the given string already ends with 'ing' then add 'ly'.  
If the length of the given string is less than 3, leave it unchanged.

|  |  |
| --- | --- |
| **Sample Input** | **Expected Output** |
| sleep | sleeping |
| amazing | amazingly |
| is | is |

**Code-**

def add\_string(s):

if len(s) < 3:

return s

elif s.endswith('ing'):

return s + 'ly'

else:

return s + 'ing'

# Test cases

print(add\_string("sleep")) # Output: "sleeping"

print(add\_string("amazing")) # Output: "amazingly"

print(add\_string("is")) # Output: "is"

//-----------------------------------------------------------------------

Creating a tuple  
lunch\_menu=("Welcome Drink","Veg Starter","Non-Veg Starter","Veg Main Course","Non-Veg Main Course","Dessert")  
  
#These are also valid  
sample\_tuple="A","B","C"  
sample\_tuple1=("D",)  
  
#Length of the tuple  
print("Number of elements in the tuple, lunch\_menu:",len(lunch\_menu))  
  
#Random read  
print("Element at 2nd index position in lunch\_menu:", lunch\_menu[2])  
  
print("Concatenating tuples:")  
#Concatenating two tuples  
sample\_tuple=sample\_tuple+sample\_tuple1 #This will create a new tuple by combining the elements of existing sample\_tuple and sample\_tuple1  
print(sample\_tuple)  
  
#Adding a single element to a tuple  
sample\_tuple=sample\_tuple+("E",)  # This will also create a new tuple, sample\_tuple  
print(sample\_tuple)

//------------------------------------------------------------------------

Write a python function which accepts a list of numbers and returns true, if 1, 2, 3 appears in sequence in the list.

Otherwise, it should return false.

|  |  |
| --- | --- |
| **Sample Input** | **Expected Output** |
| [1, 1, 2, 3, 1] | True |
| [1, 1, 2, 4, 3] | False |

**Code -**def sequence\_123(nums):

for i in range(len(nums) - 2):

if nums[i] == 1 and nums[i+1] == 2 and nums[i+2] == 3:

return True

return False

# Test the function

print(sequence\_123([1, 1, 2, 3, 1])) # Output: True

print(sequence\_123([1, 1, 2, 4, 3])) # Output: False

//------------------------------------------------------------