Q1 of 4outlined\_flag

Arrange the following time complexities in increasing order of growth rate.  
O(logn), O(n^2), O(n!), O(n), O(2^n)

Choose the appropriate answer:

O(n),O(logn),O(n^2),O(n!),O(2^n)

O(n),O(logn),O(2^n),O(2^n),O(n!)

O(logn),O(n),O(n^2),O(n!),O(2^n)

**Answer - A)**O(n),O(logn),O(n^2),O(n!),O(2^n)

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

Q2 of 4outlined\_flag

There are two sort algorithms, Algo1 with time complexity of O(n*2*) and the Algo2 with O(nlogn).  
Which one do you think is the better one?

Choose the appropriate answer:

Algo 1

Algo 2

**Answer -** Algo 2

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

Q3 of 4outlined\_flag

Fenny has come up with 4 different algorithms with different step-counts as shown below to solve a problem.  
Which of the following have the same Big-O complexity?

Step count for Algorithm1 : n^2 + n   
Step count for Algorithm2 : n^2 + 5\*n^2 + logn   
Step count for Algorithm3 : n^2 + 2^n   
Step count for Algorithm4 : nlogn + 2^n + n!

Algorithm1 and Algorithm4

Algorithm3 and Algorithm4

Algorithm1 and Algorithm2

Algorithm1 and Algorithm3

Algorithm2 and Algorithm3

**Answer -** Algorithm1 and Algorithm2

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

Q4 of 4outlined\_flag

Find the time complexity based on worst case analysis (Big-O) of the following code snippet.  
Assume that there are "c" statements in the while block taking 1 unit of time each for execution.

counter=0  
while(counter < n):  
  .....  
  .....  
  counter+=2

O(n)

O(n^2)

O(log n)

O(1)

**Answer -** O(n)

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

Q1 of 2outlined\_flag

The following values are to be stored in a hash table (arriving in the order shown) using the hash function, h(k)= k%5.    
81, 20, 34, 42, 21, 45  
Assume that the hash values are stored in ascending order.  
Identify whether collision will occur while mapping the values using the hash function.

Options:

a. Collision will occur at position at 0,1

b. Collision will occur at position 0

c. Collision will occur at position 1

d. Collision will occur at position 3

e. No collision

 a

b

c

d

e

**Answer -** a)Collision will occur at position at 0,1

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

Q2 of 2outlined\_flag

The following values are to be stored in a hash table: 24, 35, 45, 90, 43.  
Which of the below hash-functions is the best?

h(k)= k%5

h(k)= k%4

h(k)= k%3

h(k)= k%2

**Answer -** h(k)= k%4

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

Q1 of 3outlined\_flag

Using the binary search strategy of having numbers in sorted order, if you have to find 25 from a list containing numbers from 1 to 50 arranged in ascending order,how many guesses do you have to make?

0

1

25

50

**Answer -** B)1

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

Q2 of 3outlined\_flag

Using the binary search strategy of having numbers in sorted order, if you have to find 50 from a list containing numbers from 1 to 50 arranged in ascending order, how many guesses do you have to make?

6

8

4

2

**Answer -** B)8

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

Q3 of 3outlined\_flag

Do you think the number of guesses to be made is equal to the position of the number to be guessed?

Yes

No

**Answer -** No

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

**Dictionary** -   
Data can be store in the form of key and value.

d={  
key1:value1,  
key2:value2,  
key3:value3  
}

 d[key1]  
d.get(key1)

 for k,v in d.items():  
   print(k,v)

for key in d:  
     print(key)  
     print(d[key])

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

A dictionaries can be used to store an unordered collection of key-value pairs. The key should be unique and can be of any immutable data type. Like lists, dictionaries are mutable. Let’s now understand how a dictionary is implemented in Python.

|  |  |  |
| --- | --- | --- |
| Creating a dictionary | crew\_details= { "Pilot":"Kumar", "Co-pilot":"Raghav", "Head-Strewardess":"Malini", "Stewardess":"Mala" } | First element in every pair is the key and the second element is the value. |
| Accessing the value using key | crew\_details["Pilot"] | This will return the corresponding value for the specified key |
| Iterating through the dictionary | for key,value in crew\_details.items():      print(key,":",value) | items function gives both key and value, which can be used in a for loop. |

Dictionary in Python also have many inbuilt functions.

| **Function** | **Output** | **Explanation** |
| --- | --- | --- |
| crew\_details.get("Pilot") | Kumar | Returns the value for given key. If the given key is not found, returns None |
| crew\_details.update({"Flight Attendant":"Jane", "Co-pilot":"Henry"}) | No output, dictionary will be updated | Updates the dictionary with the given key-value pairs. If a key-value pair is already existing, it will be overwritten, otherwise it will be added to the dictionary |

//-------------------------------------------------------------------------------------  
#Creating a dictionary  
crew\_details={  
            "Pilot":"Kumar",  
            "Co-Pilot":"Raghav",  
            "Head-Strewardess":"Malini",  
            "Stewardess":"Mala"  
}  
print(crew\_details["Pilot"])

 print("\nIterating the dictionary using items function")  
for key,value in crew\_details.items():  
    print(key,":",value)

#Usually while working with dictionary, you will be interested in specific values.   
#Let’s find the value of all pilots from crew\_details.  
print("\nIterating the dictionary using keyword 'in'")  
for key in crew\_details:  
    if(key=="Pilot" or key=="Co-Pilot"):  
        print(crew\_details[key])  
#Note: Dictionary being unordered, the order of the values being displayed may vary during each execution of the above for loop.

 #Dictionaries are mutable  
crew\_details["Pilot"]="James" # Here the value for key "Pilot" is being updated to "James"  
print("\nAfter modifying the value of Pilot:", crew\_details["Pilot"])

 print("------------------------------------------------------------------")  
print("Before update:")  
# Usage of get method()  
print("Co-pilot:",crew\_details.get("Co-Pilot"))

 #Usage of update method()  
crew\_details.update({"Flight Attendant":"Jane", "Co-pilot":"Henry"})

print("\nAfter update:")  
print("Co-pilot:",crew\_details.get("Co-pilot"))  
print("Flight Attendant:",crew\_details["Flight Attendant"])

**Code -**

# Creating a dictionary

crew\_details = {

"Pilot": "Kumar",

"Co-Pilot": "Raghav",

"Head-Stewardess": "Malini",

"Stewardess": "Mala"

}

# Accessing a specific value

print("Pilot:", crew\_details["Pilot"])

print("\nIterating the dictionary using items function")

for key, value in crew\_details.items():

print(f"{key}: {value}")

print("\nIterating the dictionary using keyword 'in'")

for key in crew\_details:

if "Pilot" in key or "Co-Pilot" in key:

print(crew\_details[key])

crew\_details["Pilot"] = "James" # Here the value for key "Pilot" is being updated to "James"

print("\nAfter modifying the value of Pilot:", crew\_details["Pilot"])

print("------------------------------------------------------------------")

print("Before update:")

print("Co-pilot:", crew\_details.get("Co-Pilot"))

crew\_details.update({"Flight Attendant": "Jane", "Co-Pilot": "Henry"})

print("\nAfter update:")

print(f"Co-pilot: {crew\_details.get('Co-Pilot')}")

print(f"Flight Attendant: {crew\_details['Flight Attendant']}")

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

Q1 of 2outlined\_flag

What is the output of the following code snippet?

sample\_dict = {'a':1,'b':2}  
sample\_dict.update({'b':5, 'c':10 })  
print(sample\_dict.get('b'), sample\_dict.get('c'))

5,10

2 None

2 10

5 None

**Answer -** 5,10

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

Q2 of 2outlined\_flag

Assume that a dictionary has data in the form of { key1:value1, key2:value2…}

my\_library =  
{  
    103 : "Alice in Wonderland",  
    104 : "The Turning Point",  
    113 : "Wings on Fire",  
    134 : "Harry Potter"  
}

What do you think my\_library[104] would point to ?

104

This is a dictionary and not a list, you cant do that!

The Turning Point

There are only 4 elements here, what on earth are you calling index 104 for

**Answer -** The Turning Point

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

Care hospital wants to know the medical speciality visited by the maximum number of patients. Assume that the patient id of the patient along with the medical speciality visited by the patient is stored in a list. The details of the medical specialities are stored in a dictionary as follows:  
{  
"P":"Pediatrics",  
"O":"Orthopedics",  
"E":"ENT  
}   
  
Write a function to find the medical speciality visited by the maximum number of patients and return the name of the speciality.  
  
**Note:**

1. Assume that there is always only one medical speciality which is visited by maximum number of patients.
2. Perform case sensitive string comparison wherever necessary.

|  |  |
| --- | --- |
| **Sample Input** | **Expected Output** |
| [101,P,102,O,302,P,305,P] | Pediatrics |
| [101,O,102,O,302,P,305,E,401,O,656,O] | Orthopedics |
| [101,O,102,E,302,P,305,P,401,E,656,O,987,E] | ENT |

**Code** -

def max\_speciality(visits):

specialities = {

"P": "Pediatrics",

"O": "Orthopedics",

"E": "ENT"

}

count = {}

# Counting visits for each speciality

for visit in visits:

if visit in count:

count[visit] += 1

else:

count[visit] = 1

# Finding the speciality with maximum visits

max\_visits = max(count.values())

max\_speciality = [key for key, value in count.items() if value == max\_visits][0]

return specialities[max\_speciality]

# Test cases

test\_cases = [

[101, 'P', 102, 'O', 302, 'P', 305, 'P'],

[101, 'O', 102, 'O', 302, 'P', 305, 'E', 401, 'O', 656, 'O'],

[101, 'O', 102, 'E', 302, 'P', 305, 'P', 401, 'E', 656, 'O', 987, 'E']

]

for visits in test\_cases:

print(max\_speciality(visits))

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

Write a python function, **find\_correct()** which accepts a dictionary and returns a list as per the rules mentioned below.  
The input dictionary will contain correct spelling of a word as key and the spelling provided by a contestant as the value.  
  
The function should identify the degree of correctness as mentioned below:  
CORRECT, if it is an exact match  
  
ALMOST CORRECT, if no more than 2 letters are wrong  
WRONG, if more than 2 letters are wrong or if length (correct spelling versus spelling given by contestant) mismatches.  
  
and return a list containing the number of CORRECT answers, number of ALMOST CORRECT answers and number of WRONG answers.   
Assume that the words contain only uppercase letters and the maximum word length is 10.

|  |  |
| --- | --- |
| **Sample Input** | **Expected Output** |
| {"THEIR": "THEIR", "BUSINESS": "BISINESS","WINDOWS":"WINDMILL","WERE":"WEAR","SAMPLE":"SAMPLE"} | [2, 2, 1] |

**Code -**

def find\_correct(words):

correct = 0

almost\_correct = 0

wrong = 0

for key, value in words.items():

if key == value:

correct += 1

elif len(key) == len(value):

count\_diff = sum(1 for a, b in zip(key, value) if a != b)

if count\_diff <= 2:

almost\_correct += 1

else:

wrong += 1

else:

wrong += 1

return [correct, almost\_correct, wrong]

# Test case

words = {"THEIR": "THEIR", "BUSINESS": "BISINESS", "WINDOWS": "WINDMILL", "WERE": "WEAR", "SAMPLE": "SAMPLE"}

print(find\_correct(words)) # Output: [2, 2, 1]

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

Write a python program that accepts a text and displays a string which contains the word with the largest frequency in the text and the frequency itself separated by a space.  
  
**Rules:**

1. The word should have the largest frequency.
2. In case multiple words have the same frequency, then choose the word that has the maximum length.

**Assumptions:**

1. The text has no special characters other than space.
2. The text would begin with a word and there will be only a single space between the words.

Perform case insensitive string comparisons wherever necessary.

|  |  |
| --- | --- |
| **Sample Input** | **Expected Output** |
| "Work like you do not need money love like you have never been hurt and dance like no one is watching" | like 3 |
| "Courage is not the absence of fear but rather the judgement that something else is more important than fear" | fear 2 |

**Code -**

def max\_frequency\_word(text):

# Convert text to lowercase and split into words

words = text.lower().split()

# Count frequency of each word

frequency = {}

for word in words:

frequency[word] = frequency.get(word, 0) + 1

# Find the word with the maximum frequency

max\_freq = max(frequency.values())

max\_freq\_words = [word for word, freq in frequency.items() if freq == max\_freq]

# If multiple words have the same frequency, choose the one with maximum length

max\_freq\_words.sort(key=lambda x: len(x), reverse=True)

return f"{max\_freq\_words[0]} {max\_freq}"

# Test cases

text1 = "Work like you do not need money love like you have never been hurt and dance like no one is watching"

text2 = "Courage is not the absence of fear but rather the judgement that something else is more important than fear"

print(max\_frequency\_word(text1))

print(max\_frequency\_word(text2))

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

sentence="Work like you do not need money love like you have never been hurt and dance like no one is watching"

wordList=sentence.split(" ")

print(wordList)

wordList= set(wordList)

print(wordList)

d={}

for word in wordList:

    #print(word,' -> ',sentence.count(word))

    d[word]=sentence.count(word)

 print(d)

**Code -**

sentence = "Work like you do not need money love like you have never been hurt and dance like no one is watching"

wordList = sentence.split()

d = {}

for word in wordList:

d[word] = d.get(word, 0) + 1

max\_freq = max(d.values())

max\_freq\_words = [word for word, freq in d.items() if freq == max\_freq]

max\_freq\_words.sort(key=lambda x: len(x), reverse=True)

print(f"{max\_freq\_words[0]} {max\_freq}")

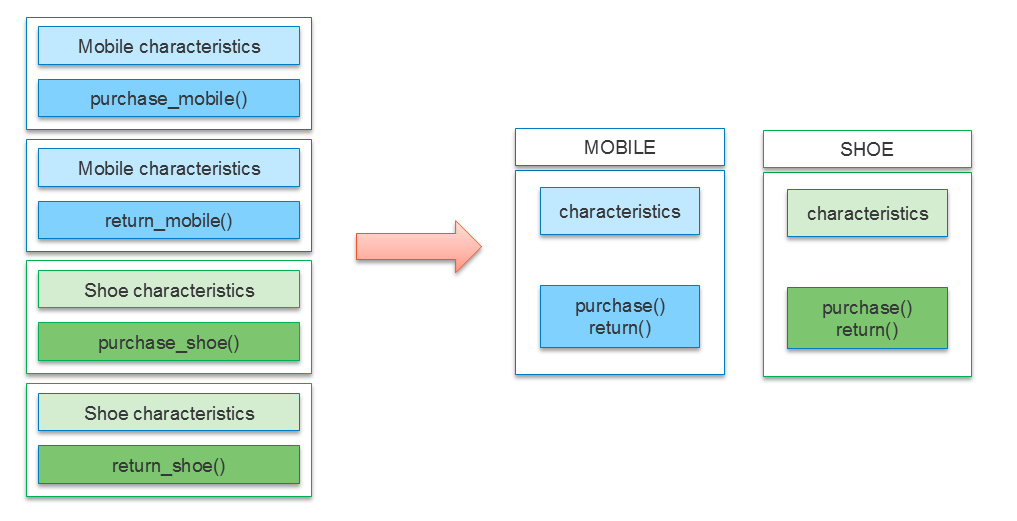
//------------------------------------------------------------------------------------------

**OOP -**Object oriented programming approach allows us to club together the data and behavior so that it becomes easier to code real world scenarios.

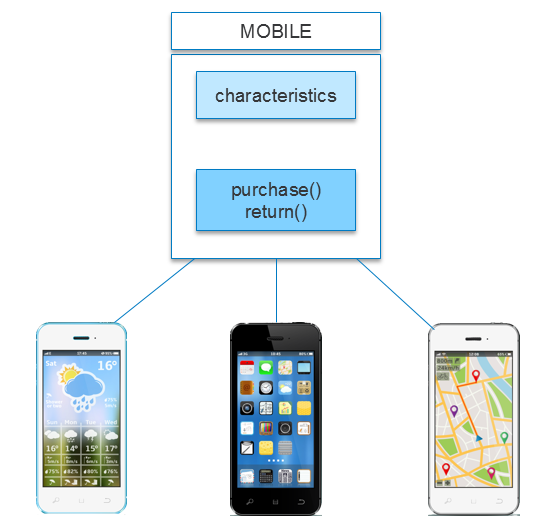
Consider the following example, where Mobile and Shoe clubs characteristics as data & purchase(), return() as it's behavior.

**Object -**

**Class -**



Once we have a template of the data and the related behavior we can use that template to create many copies.



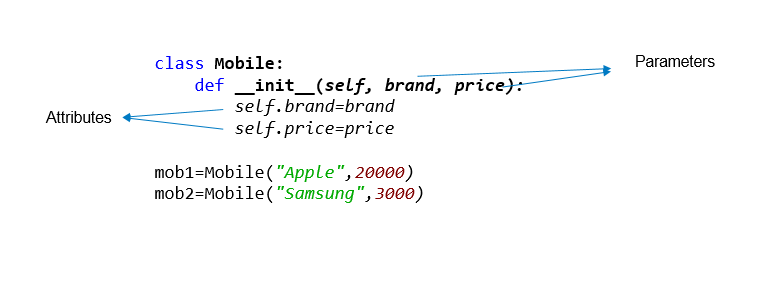
This style of programming where we create a template and create copies from that template is called object oriented programming. This style allows us to code for scenarios closely linked with real life.

The template we create is called a Class and the copies we create out of it is called an object.

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

**How do we create attributes in a class?**

    Attributes can be added to a class through a special function called \_\_init\_\_(). We will discuss more about the syntax later. But for now, this is how the mobile class will look like with attributes in it.



In the code, brand and price are the attributes. All objects of this class will now have these attributes automatically. Here mob1 is assigned “Apple” and 20000 as values for the attributes brand and price respectively.

**Note:**The parameter names and attribute names need not match

# Constructor & Self - Introduction    When we create an object, the special \_\_init\_\_() method inside the class of that object is invoked automatically. This special function is called as a **constructor**.

class Mobile:

def \_\_init\_\_(self):

    print("Inside constructor")

mob1=Mobile()

When we create an object, the special \_\_init\_\_() method inside the class of that object is invoked automatically. This special function is called as a **constructor**.

class Mobile:

def \_\_init\_\_(self):

 print("Inside constructor")

mob1=Mobile()

**self** is not a keyword. self refers to the current object being executed.

class Mobile:

 def \_\_init\_\_(self):

 print("Id of self in constructor", id(self))

mob1=Mobile()

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

Class Example -  
Problem Statement

WeCare insurance company wants to calculate premium of vehicles.  
Vehicles are of two types – "Two Wheeler" and "Four Wheeler". Each vehicle is identified by vehicle id, type, cost and premium amount.  
Premium amount is 2% of the vehicle cost for two wheelers and 6% of the vehicle cost for four wheelers. Calculate the premium amount and display the vehicle details.

Identify the class name and attributes to represent vehicles.

* calculate\_premium()
* vehicle\_cost
* TwoWheeler
* vehicle\_type
* vehicle\_id
* Vehicle
* premium\_amount
* FourWheeler
* premium\_percentage
* calculate\_vehicle\_cost()
* \_\_init\_\_()
* display\_vehicle\_details()

Write a Python program to implement the class chosen with its attributes and methods.

**Note:**

1. Consider all instance variables to be private and methods to be public
2. Include getter and setter methods for all instance variables
3. Display appropriate error message, if the vehicle type is invalid
4. Perform case sensitive string comparison

Represent few objects of the class, initialize instance variables using setter methods, invoke appropriate methods and test your program.

**Code -**

class Vehicle:

def \_\_init\_\_(self, vehicle\_id, vehicle\_type, vehicle\_cost):

self.\_\_vehicle\_id = vehicle\_id

self.\_\_vehicle\_type = vehicle\_type

self.\_\_vehicle\_cost = vehicle\_cost

self.\_\_premium\_amount = 0

def get\_vehicle\_id(self):

return self.\_\_vehicle\_id

def set\_vehicle\_id(self, vehicle\_id):

self.\_\_vehicle\_id = vehicle\_id

def get\_vehicle\_type(self):

return self.\_\_vehicle\_type

def set\_vehicle\_type(self, vehicle\_type):

self.\_\_vehicle\_type = vehicle\_type

def get\_vehicle\_cost(self):

return self.\_\_vehicle\_cost

def set\_vehicle\_cost(self, vehicle\_cost):

self.\_\_vehicle\_cost = vehicle\_cost

def get\_premium\_amount(self):

return self.\_\_premium\_amount

def calculate\_premium(self):

if self.\_\_vehicle\_type == "Two Wheeler":

self.\_\_premium\_amount = self.\_\_vehicle\_cost \* 0.02

elif self.\_\_vehicle\_type == "Four Wheeler":

self.\_\_premium\_amount = self.\_\_vehicle\_cost \* 0.06

else:

print("Invalid vehicle type")

def display\_vehicle\_details(self):

print("Vehicle ID:", self.\_\_vehicle\_id)

print("Vehicle Type:", self.\_\_vehicle\_type)

print("Vehicle Cost:", self.\_\_vehicle\_cost)

print("Premium Amount:", self.\_\_premium\_amount)

class TwoWheeler(Vehicle):

def \_\_init\_\_(self, vehicle\_id, vehicle\_cost):

super().\_\_init\_\_(vehicle\_id, "Two Wheeler", vehicle\_cost)

class FourWheeler(Vehicle):

def \_\_init\_\_(self, vehicle\_id, vehicle\_cost):

super().\_\_init\_\_(vehicle\_id, "Four Wheeler", vehicle\_cost)

# Test the program

two\_wheeler = TwoWheeler("TW001", 50000)

two\_wheeler.calculate\_premium()

two\_wheeler.display\_vehicle\_details()

four\_wheeler = FourWheeler("FW001", 100000)

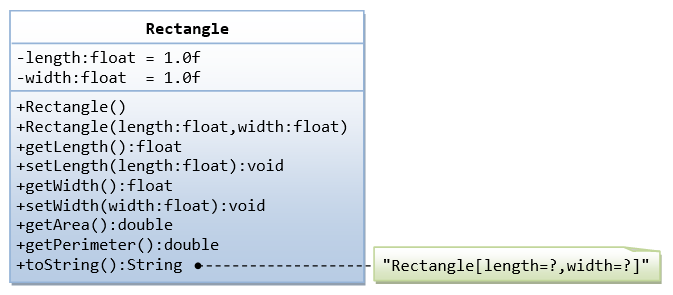
four\_wheeler.calculate\_premium()

four\_wheeler.display\_vehicle\_details()

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

**Problem-1**

A class called Rectangle, which models a rectangle with a length and a width (in float), is designed as shown in the following class diagram. Write the Rectangle class.

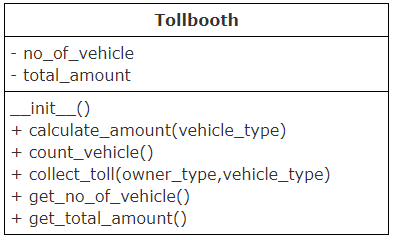


**Code -**

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

Problem Statement

A toll booth on the way to Bangalore wants to keep the track of the number of vehicles passed through it and total amount collected by them.  
Write a python program to implement the class diagram given below.



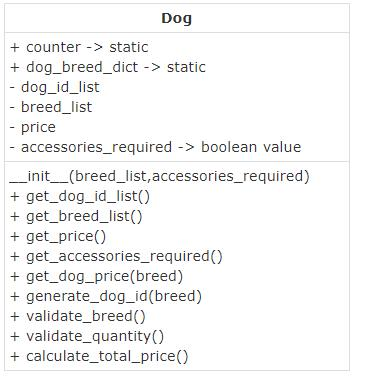
|  |
| --- |
| **Class description:** **Constructor:** Initialize both the instance variables, no\_of\_vehicle, total\_amount to 0   1. **count\_vehicle():** Increment total number of vehicle by 1 2. **calculate\_amount(vehicle\_type):** Accept vehicle type and identify toll amount for that vehicle based on details given in the table. Add it to the total\_amount instance variable. 3. **collect\_toll(owner\_type,vehicle\_type):**Accept owner type and vehicle type of the vehicle for which toll should be collected.   If the owner of the vehicle is a "VIP", then toll amount need not be collected but number of vehicles should be updated. For any other type of owner, calculate the toll amount and update the number of vehicles. (Hint: Invoke appropriate methods to complete the functionality) Perform case insensitive string comparison. Create an object of Tollbooth class, invoke collect\_toll() method for different vehicles and test your program. |

**Code -**

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

Problem Statement

Little Puppy Kennel sells dogs of different breeds. They want to automate their selling process.  
Write a python program to implement the class diagram given below.



**Class Description:**

|  |
| --- |
| 1. **dog\_breed\_dict:** Static dictionary which contains the breed of the dog as key and the number of dogs available as value. Initialize it with the sample data given |



1. nitialize static variable counter to 100
2. **breed\_list:**List of dog breeds required by the customer. Initialize it in the constructor
3. **dog\_id\_list:** List of dog ids. Initialize it to an empty list in the constructor
4. **price:** Total price to be paid by the customer. Initialize it to 0 in the constructor
5. **accessories\_required:** Boolean value – True indicated accessories are required and False indicated accessories are not required.                          Initialize it in the constructor
6. **validate\_breed():** Return true if all the breeds required by the customer are available. Else return false
7. **validate\_quantity():** Return true if one dog/breed is available for all the breeds requested by the customer. Else return false
8. **generate\_dog\_id(breed):** Accept the breed of the dog for which dog id should be generated.                                                                                Auto-generate dog id starting from 101 prefixed by the first character of the breed
9. **get\_dog\_price(breed):** Return the price of the dog whose breed is passed to the method
10. **calculate\_total\_price():**Calculate the total price of all the dogs required by the customer.
    1. Validate breed and quantity of all the dogs required by the customer
    2. If valid,
       1. For every breed in breed\_list,
          1. Update quantity in dog\_breed\_dict
          2. Auto-generate dog id and append it to attribute, dog\_id\_list
          3. Add price to attribute, price
       2. If accessories are required, add 350 to attribute, price
       3. If price is more than 1500, provide 5% discount on price
    3. If any breed is not available, return -1
    4. If quantity is not available for any breed, return -2
       1. If quantity is not available for any breed, return -2

|  |  |
| --- | --- |
| **Breed** | **Price** |
| Labrador Retriever | 800 |
| German Shepherd | 1230 |
| Beagle | 650 |

**Code -**

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