Academic Integrity Statement

As a matter of Departmental policy, **we are required to give you a 0** unless you **type your name** after the following statement:

> *I certify on my honor that I have neither given nor received any help, or used any non-permitted resources, while completing this evaluation.*

DAVID NGUYEN

Permitted Resources

This exam is open-book, open-notes, open-internet, open-everything. The only thing you are not allowed to do is ask another human being for help, with the exception of me. Examples:

1. **Permitted**: Checking course notes and videos; browsing Google or StackOverflow; consulting previous Campuswire posts.
2. **Not permitted**: Requesting help on Chegg; publicly posting on Campuswire (messages to me are ok); privately asking classmates for help; posting on StackOverflow.

If you encounter any situation in which you're not 100% sure what's permitted, **just ask**!

Partial Credit

Let us give you partial credit! If you're stuck on a problem and just can't get your code to run:

First, **breathe**. Then, do any or all of the following:

1. Write down everything relevant that you know about the problem, as comments where your code would go.
2. If you have non-functioning code that demonstrates some correct ideas, indicate that and keep it (commented out).
3. Write down pseudocode (written instructions) outlining your solution approach.

In brief, even if you can't quite get your code to work, you can still **show us what you know.**

Part A (30 points)

Write a Car class. This car should have the following attributes and properties:

- A class variable called `wheels` with value `4` (since all cars have four wheels).
- An instance variable `make` (string), e.g. `"Toyota"`, `"Subaru"`, `"Chevy"`).
- An instance variable `model` (string), e.g. `"Corolla"`, `"Outback"`, `"Tahoe"`).
- An instance variable `year` (integer) the year the car was manufactured e.g. 2010).
- An instance variable `miles_per_gallon` (float) the number of miles the car can drive per gallon of gas: e.g. 18.0)
- An instance variable `current_gas_level` (float), the number of gallons currently in the tank of the car.

Prompt

1. Your function should have an `__init__` method which checks that `make` and `model` are strings, that `year` is an integer which is greater than 1900, that `current_gas_level` and `miles_per_gallon` are floats or ints, and that `current_gas_level` ≥ 0. You should also check that `miles_per_gallon` >0. Make sure to raise *informative* errors if any of these checks fail. You should appropriately use both type errors and value errors in your solution.
2. Write a `drive()` method. This method should accept an argument `number_of_miles` giving the desired number of miles to drive. The `current_gas_level` should be reduced by an appropriate amount. If the `current_gas_level` is not sufficient to drive the specified number of miles, then raise an informative error of an appropriate type, and do not change `current_gas_level`.
3. Demonstrate **one example** of successfully constructing an object of class `Car` and **three different examples** of raising an error from checks in the `__init__` method.
4. Demonstrate **one example** of sucessfully calling the `drive` method, printing the `current_gas_level` before and after. Demonstrate **one example** in which there is not enough gas and the corresponding error is raised.

Specs

Comments and docstrings, please!

1. Remember that the class itself should have a docstring describing its overall purpose, instance variables, and methods.
2. Each method (with the possible exception of `__init__` should have a docstring describing its purpose, inputs, and outputs/other effects.

In [161]:
```
# define your class here
class car:
    '''
    car class that has characteristics to represent a car in real life
    class variable is the number of wheels on the vehicle
    instance varibles represent the make, model, year, mpg, and gas tank
    drive method to show emulate gas consumption over certain number of miles
    '''

    wheels = 4 #class variable wheels

    def __init__(self, make, model , year, miles_per_gallon, current_gas_level):
        #error checks for each variable
        if type(make) is not str:
            raise TypeError("'make' should be string!")
        if type(model) is not str:
            raise TypeError("'model' should be a string!")
        if type(year) is not int:
            raise TypeError("'year' should be an int!")
        if year <= 1900:
            raise ValueError("'year' should be greater than 1900!")
        if type(current_gas_level) not in [float, int]:
            raise TypeError("'current_gas_level' should be a float or int!")
        if current_gas_level < 0:
            raise ValueError("'current_gas_level' should be at least 0")
        if type(miles_per_gallon) not in [float, int]:
            raise TypeError("'miles_per_gallon' should be a float or int!")
        if miles_per_gallon <= 0:
            raise ValueError("'miles_per_gallon' cannot be 0 or less!")

        #if we pass all the error checks, put the values into our instance variables
        self.make = make
        self.model = model
        self.year = year
        self.mpg = miles_per_gallon
        self.cgl = current_gas_level


    def drive(self, number_of_miles):
        '''
        takes input number of miles to drive and then adjusts the gas level
            to correct amount after drive
        if input, number_of_miles, is too larges, method raises an error
        '''
        #conversion to check the amount of gas needed for the input
        gal_needed = float(number_of_miles)/float(self.mpg)

        #check if number_of_miles is too much or valid
        if self.cgl < gal_needed:
            #if input is too much then, we raise a ValueError
            raise ValueError("'number_of_miles' is too large, not enough gas")
        else:
            #if input is value, we decrement the current gas level
            self.cgl -= gal_needed
```

Show the required examples in the code blocks below. Feel free to make new code blocks as needed.

In [145]:
```
# demo 1: expected wrong make type
myCar = car(1, 'Accord', 2012, 23, 10)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-145-229f721335be> in <module>
      1 # demo 1: expected wrong make type
----> 2 myCar = car(1, 'Accord', 2012, 23, 10)

<ipython-input-144-9e828015498a> in __init__(self, make, model, year, miles_per_gallon, current_gas_level)
     10         #error checks
     11         if type(make) is not str:
---> 12             raise TypeError("'make' should be string!")
     13         if type(model) is not str:
     14             raise TypeError("'model' should be a string!")

TypeError: 'make' should be string!
```

In [146]:
```
# demo 2 : expected wrong year value
myCar = car('Honda', 'Accord', 1800, 23, 10)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-146-0beef6eece32> in <module>
      1 # demo 2 : expected wrong year value
----> 2 myCar = car('Honda', 'Accord', 1800, 23, 10)

<ipython-input-144-9e828015498a> in __init__(self, make, model, year, miles_per_gallon, current_gas_level)
     16             raise TypeError("'year' should be an int!")
     17         if year <= 1900:
---> 18             raise ValueError("'year' should be greater than 1900!")
     19         if type(current_gas_level) not in [float, int]:
     20             raise TypeError("'current_gas_level' should be a float or int!")

ValueError: 'year' should be greater than 1900!
```

In [147]:
```
# demo 3: expected wrong miles per gallon value
myCar = car('Honda', 'Accord', 2012, 0, 10 )
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-147-f6d54921949e> in <module>
      1 # demo 3: expected wrong miles per gallon value
----> 2 myCar = car('Honda', 'Accord', 2012, 0, 10 )

<ipython-input-144-9e828015498a> in __init__(self, make, model, year, miles_per_gallon, current_gas_level)
     24             raise TypeError("'miles_per_gallon' should be a float or int!")
     25         if miles_per_gallon <= 0:
---> 26             raise ValueError("'miles_per_gallon' cannot be 0 or less!")
     27
     28         #if we pass all the error checks, put the values into our instance variables

ValueError: 'miles_per_gallon' cannot be 0 or less!
```

In [150]:
```
# demo 4: expected that everything is right type and value
myCar = car('Honda', 'Accord', 2012, 20, 10.0)
```

In [151]:
```
# demonstrate successful driving here
print('Current gas level: ' + str(myCar.cgl))
print('Car drives 100 miles')
myCar.drive(100)
print('Current gas level: ' + str(myCar.cgl))
```

```
Current gas level: 10.0
Car drives 100 miles
Current gas level: 5.0
```

In [152]:
```
# demonstrate unsuccessful driving here
# (not enough gas)
print('Current gas level: ' + str(myCar.cgl))
print('Car wants to drive 1000 miles')
myCar.drive(1000)
print('Current gas level: ' + str(myCar.cgl))
```

```
Current gas level: 5.0
Car wants to drive 1000 miles

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-152-e17aa44ff080> in <module>
      3 print('Current gas level: ' + str(myCar.cgl))
      4 print('Car wants to drive 1000 miles')
----> 5 myCar.drive(1000)
      6 print('Current gas level: ' + str(myCar.cgl))
      7

<ipython-input-144-9e828015498a> in drive(self, number_of_miles)
     37         gal_needed = float(number_of_miles)/float(self.mpg)
     38         if self.cgl < gal_needed:
---> 39             raise ValueError("'number_of_miles' is too large, not enough gas")
     40         else:
     41             self.cgl -= gal_needed

ValueError: 'number_of_miles' is too large, not enough gas
```

Part B (10 points)

Write a `Motorcycle` class that possesses the same class variables, instance variables, and `drive()` method as the `Car` class.

Demonstrate a working example of an object of class `Motorcycle`. Then, print the `wheels` instance variable of the object to verify that the object has the correct number of wheels (which should be different from a `Car`).

Note: motorcycles have two wheels.

Specs

- **Your solution should be under five lines long.**
- Comments and docstrings are not required for this part.

In [168]:
```
# class definition here
class motorcycle(car):
    wheels = 2 #set class variable for motorcycle


# motorcycle class will have all the functions and instance variables that car
# class contains through inheritance
```

In [104]:
```
# construct object of class
myMotorcycle = motorcycle("Yamaha", "YDX", 2018, 75, 2)
```

In [164]:
```
# show number of wheels
myMotorcycle.wheels
```

Out[164]: 2

Part C (15 points)

Write a function called `age` whose input is an object of class `Car` and whose output is the age of that object, computed according to the formula

$$\text{age} = 2021 - \text{year manufactured}$$

Your function should check that the input is an instance of class `Car`. This implies that it should also work for objects of class `Motorcycle`, and other classes that inherit from `Car`.

Demonstrate that your function works and raises an appropriate error when the input is a `Car` or `Motorcycle`, but not when the input is an `int` or a `string`.

Specs

- Comments and docstrings are not required in this part.

In [128]:
```
# define function here
def age(myObject):
    '''
    will return age of object if obeject is in car class
    '''
    #check if input object is in car class using isinstance()
    if not isinstance(myObject, car):
        #if not in car class, that is a value error
        raise ValueError('input must be an object of the class car')
    else:
        #return age using formula provided
        return (2021 - myObject.year)
```

In [129]:
```
# show that it works on Cars
age(myCar)
```

Out[129]: 9

In [130]:
```
# show that it works on Motorcycles
age(myMotorcycle)
```

Out[130]: 3

In [131]:
```
# show an error on other inputs
age(11)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-131-5c43f065fe01> in <module>
      1 # show an error on other inputs
----> 2 age(11)

<ipython-input-128-1dd40ed64a92> in age(myObject)
      2 def age(myObject):
      3     if not isinstance(myObject, car):
----> 4         raise ValueError('input must be an object of the class car')
      5     else:
      6         return (2021 - myObject.year)

ValueError: input must be an object of the class car
```

In [132]:
```
age('car')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-132-cf15cd1a510a> in <module>
----> 1 age('car')

<ipython-input-128-1dd40ed64a92> in age(myObject)
      2 def age(myObject):
      3     if not isinstance(myObject, car):
----> 4         raise ValueError('input must be an object of the class car')
      5     else:
      6         return (2021 - myObject.year)

ValueError: input must be an object of the class car
```

In [ ]: