Classes (Chapter 9)

Object-oriented programming is one of the most effective approaches to writing software. In object-oriented programming you write *classes* that represent real-world things and situations, and you create *objects* based on these classes.

In [12]:
```
class Dog:
    """A simple attempt to model a dog."""

    def __init__(self, name, age):
        """Initialize name and age attributes."""
        self.name = name
        self.age = age

    def sit(self):
        """Simulate a dog sitting in response to a command."""
        print(f"{self.name} is now sitting.")

    def roll_over(self):
        """Simulate rolling over in response to a command."""
        print(f"{self.name} rolled over!")
```

Create two instances of the `Dog` class. These are called *instances* or *objects*.

In [13]:
```
my_dog = Dog('Willie', 6)
your_dog = Dog('Lucy', 3)
```

Accessing attributes and calling methods.

In [14]:
```
print(f"My dog's name is {my_dog.name}.")
print(f"My dog is {my_dog.age} years old.")
my_dog.sit()
```

```
print(f"\nYour dog's name is {your_dog.name}.")
print(f"Your dog is {your_dog.age} years old.")
your_dog.sit()
```

```
My dog's name is Willie.
My dog is 6 years old.
Willie is now sitting.

Your dog's name is Lucy.
Your dog is 3 years old.
Lucy is now sitting.
```

Working with Classes and Instances

In [13]:

```
"""A class that can be used to represent a car."""

class Car():
    """A simple attempt to represent a car."""

    def __init__(self, manufacturer, model, year):
        """Initialize attributes to describe a car."""
        self.manufacturer = manufacturer
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        """Return a neatly formatted descriptive name."""
        long_name = str(self.year) + ' ' + self.manufacturer + ' ' + self.model
        return long_name.title()

    def read_odometer(self):
        """Print a statement showing the car's mileage."""
        print("This car has " + str(self.odometer_reading) + " miles on it.")

    def update_odometer(self, mileage):
        """
        Set the odometer reading to the given value.
        Reject the change if it attempts to roll the odometer back.
        """
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
```

```
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        """Add the given amount to the odometer reading."""
        self.odometer_reading += miles
```

In [17]:
```
# create a new car, odometer receives a default value
my_new_car = Car('audi', 'a4', 2019)
print(my_new_car.get_descriptive_name())
my_new_car.read_odometer()
print(type(my_new_car))
```

```
2019 Audi A4
This car has 0 miles on it.
<class '__main__.Car'>
```

In [18]:
```
# modifying attribute values
my_new_car.odometer_reading = 23
my_new_car.read_odometer()
```

```
This car has 23 miles on it.
```

In [5]:
```
# modifying attribute directly with a method
my_new_car.update_odometer(56)
my_new_car.read_odometer()
# this allows custom logic to validate the attribute value
my_new_car.update_odometer(10)
```

```
This car has 56 miles on it.
You can't roll back an odometer!
```

In [19]:
```
# modifying attribute indirectly with a method
my_new_car.increment_odometer(20)
my_new_car.read_odometer()
```

```
This car has 43 miles on it.
```

Inheritance

`ElectricCar` inherts all methods and attributes from `Car`

In [20]:
```
class ElectricCar(Car):
    """Models aspects of a car, specific to electric vehicles."""
```

```
    def __init__(self, manufacturer, model, year):
        """Initialize attributes of the parent class."""
        super().__init__(manufacturer, model, year)
```

In [21]:
```
my_tesla = ElectricCar('tesla','model s',2019)
print(my_tesla.get_descriptive_name()) # inherited from Car
```

```
2019 Tesla Model S
```

New `ElectricCar` class that has a `battery_size` attribute and associated method `describe_battery()`

In [17]:
```
class ElectricCar(Car):
    """Models aspects of a car, specific to electric vehicles."""

    def __init__(self, manufacturer, model, year):
        """
        Initialize attributes of the parent class.
        Then initialize attributes specific to an electric car.
        """
        super().__init__(manufacturer, model, year)
        self.battery_size = 75

    def describe_battery(self):
        """Print a statement describing the battery size."""
        print(f"This car has a {self.battery_size}-kWh battery.")
```

In [19]:
```
my_tesla = ElectricCar('tesla','model s',2019)


my_tesla.get_descriptive_name()

#print(my_tesla.get_descriptive_name()) # inherited from Car
#my_tesla.describe_battery() # specific to Electric Cars

#my_toyota = Car('toyota','corolla',2007)
#my_toyota.describe_battery() # does not work, not an ElectricCar!
```

Out[19]:
```
'2019 Tesla Model S'
```

Overriding methods in a child class

In [13]:
```
class ElectricCar(Car):
    """Models aspects of a car, specific to electric vehicles."""
```

```
    def __init__(self, manufacturer, model, year):
        """
        Initialize attributes of the parent class.
        Then initialize attributes specific to an electric car.
        """
        super().__init__(manufacturer, model, year)
        self.battery_size = 75

    def describe_battery(self):
        """Print a statement describing the battery size."""
        print(f"This car has a {self.battery_size}-kWh battery.")

    def get_descriptive_name(self): # overrides parent method!
        """Return a neatly formatted descriptive name."""
        long_name = str(self.year) + ' ' + self.manufacturer + ' ' + self.model + ' [Electric!]'
        return long_name.title()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[13], line 1
----> 1 class ElectricCar(Car):
      2     """Models aspects of a car, specific to electric vehicles."""
      4     def __init__(self, manufacturer, model, year):

NameError: name 'Car' is not defined
```

In [27]:
```
my_tesla = ElectricCar('tesla','model s',2019)
print(my_tesla.get_descriptive_name()) # specific to ElectricCar

print(my_toyota.get_descriptive_name()) # generic Car method
```

```
2019 Tesla Model S [Electric!]
2007 Toyota Corolla
```

Modeling Real World Objects

Deciding what classes to make and whether to use inheritance can be complicated. There is often no single right answer, it is up to the programmer to figure out what works best.

In [28]:
```
class Battery():
    """A simple attempt to model a battery for an electric car."""

    def __init__(self, battery_size=60):
```

```
        """Initialize the batteery's attributes."""
        self.battery_size = battery_size

    def describe_battery(self):
        """Print a statement describing the battery size."""
        print("This car has a " + str(self.battery_size) + "-kWh battery.")

    def get_range(self):
        """Print a statement about the range this battery provides."""
        if self.battery_size == 60:
            range = 140
        elif self.battery_size == 85:
            range = 185

        message = "This car can go approximately " + str(range)
        message += " miles on a full charge."
        print(message)
```

A new `ElectricCar` class that uses `Battery` to track aspects of the car's battery

In [29]:
```
class ElectricCar(Car):
    """Models aspects of a car, specific to electric vehicles."""

    def __init__(self, manufacturer, model, year):
        """
        Initialize attributes of the parent class.
        Then initialize attributes specific to an electric car.
        """
        super().__init__(manufacturer, model, year)
        self.battery = Battery()
```

In [30]:
```
my_tesla = ElectricCar('tesla','model s',2019)
my_tesla.battery.describe_battery() # my_tesla has a battery which has the method describe_battery()

my_tesla.battery.get_range()
```

```
This car has a 60-kWh battery.
This car can go approximately 140 miles on a full charge.
```

Importing Classes

In [31]:
```
import car
import electric_car
```

```
# use the Car class defined in car.py
my_beetle = car.Car('volkswagen', 'beetle', 2015)
print(my_beetle.get_descriptive_name())

# use the ElectricCar class defined in electric_car.py
my_tesla = electric_car.ElectricCar('tesla', 'roadster', 2015)
print(my_tesla.get_descriptive_name())
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[31], line 1
----> 1 import car
      2 import electric_car
      4 # use the Car class defined in car.py

ModuleNotFoundError: No module named 'car'
```

The Python Standard Library

In [40]:
```
# example usage of a module from the standard library
# random provides tools for generating random data

from random import randint # <-- we didn't write this file, it comes with Python
print(f"You rolled a {randint(1,6)}")

from random import choice
communities = ['subs','pilot','cyber','swo','marines','army']
# feeling lucky?
print(f"Your service selection is: {choice(communities)}")
```

```
You rolled a 4
Your service selection is: subs
```

Homework Problems

9-1. Restaurant: Make a class called `Restaurant`. The `__init__()` method for `Restaurant` should store two attributes: a `restaurant_name` and a `cuisine_type`. Make a method called `describe_restaurant()` that prints these two pieces of information, and a method called `open_restaurant()` that prints a message indicating that the restaurant is open.

Make an instance called `restaurant` from your class. Print the two attributes individually, and then call both methods.

In [20]:
```
class Restaurant():

    def __init__(self, restaurant_name, cuisine_type):
        self.name = restaurant_name
        self.food = cuisine_type

    def describe_restaurant(self, rating):
        print(self.name)
        print(self.food)
        print(f'It got {rating} out of 10.')

    def open_restaurant(self):
        print(f'{self.name} is open!')

# ------------------------------------------------------------------------

# Creating an instance of Restaurant.
first = Restaurant('chiptole','burrito')

first.describe_restaurant(4)
first.open_restaurant()


#first.describe_restaurant()
#first.open_restaurant()
#print(first.Restaurant())
```

```
chiptole
burrito
It got 4 out of 10.
chiptole is open!
```

9-2. Three Restaurants: Start with your class from above. Create three different instances of the class, and call `describe_restaurant()` for each instance.

In [23]:
```
class Restaurant():

    def __init__(self, restaurant_name, cuisine_type):
        self.name = restaurant_name
        self.food = cuisine_type

    def describe_restaurant(self, rating):
```

```
        print(self.name)
        print(self.food)
        print(f'It got {rating} out of 10.')

    def open_restaurant(self):
        print(f'{self.name} is open!')

one = Restaurant('chickfila','chicken')
two = Restaurant('fiveguys','burgers')
three = Restaurant('papajohns', 'pizza')

one.describe_restaurant('8')
two.describe_restaurant('8')
three.describe_restaurant('7')
```

```
chickfila
chicken
It got 8 out of 10.
fiveguys
burgers
It got 8 out of 10.
papajohns
pizza
It got 7 out of 10.
```

9-3. Users: Make a class called `User`. Create two attributes called `first_name` and `last_name`, and then create at least three other attributes that are typically stored in a user profile. Make a method called `describe_user()` that prints a summary of the user's information. Make another method called `greet_user()` that prints a personalized greeting to the user.

Create three instances representing different users, and call both methods for each user.

In [8]:
```
class User():
    def __init__(self,first_name,last_name,age,city,school):
        self.first_name = first_name
        self.last_name = last_name
        self.age = age
        self.city = city
        self.school = school
    def describe_user(self):
        print(f'summary: {self.first_name}, {self.last_name}, {self.age}, {self.city}, {self.school}')
    def greet_user(self):
        print(f'Hello {self.first_name} {self.last_name}!')

user_one = User('Erin','Kincade','20','ATL','USNA')
```

```
user_two = User('Josh','Kincade','21','ATL','KSU')
user_three = User('Avery','Williams','21','ATL','GT')

user_one.describe_user()
user_one.greet_user()

user_two.describe_user()
user_two.greet_user()

user_three.describe_user()
user_three.greet_user()
```

```
summary: Erin, Kincade, 20, ATL, USNA
Hello Erin Kincade!
summary: Josh, Kincade, 21, ATL, KSU
Hello Josh Kincade!
summary: Avery, Williams, 21, ATL, GT
Hello Avery Williams!
```

9-4. Number Served: Start with your program from 9-1. Add an attribute called `number_served` with a default value of 0. Create an instance called `restaurant` from this class. Print the number of customers the restaurant has served, and then change this value and print it again.

Add a method called `set_number_served()` that lets you set the number of customers that have been served. Call this method with a new number and print the value again.

Add a method called `increment_number_served()` that lets you increment the number of customers who've been served. Call this method with any number you like that could represent how many customers were served in, say, a day of business.

In [1]:
```
class Restaurant():

    def __init__(self, restaurant_name, cuisine_type):
        self.name = restaurant_name
        self.food = cuisine_type

    def describe_restaurant(self, rating):
        print(self.name)
        print(self.food)
        print(f'It got {rating} out of 10.')

    def open_restaurant(self):
        print(f'{self.name} is open!')
```

```
    def number_served(self, num = 0):
        self.num = num
        print(f'{self.num} people have been served')
    def set_number_served(self, newNum):
        self.newNum = newNum
        print(f'{self.newNum} people have been served!')
    def increment_number_served(self, customers):
        self.customers = customers
        total = self.newNum + self.customers
        print(f'{total} total people have been served!!')


restaurant = Restaurant('chiptole','burrito')
restaurant.number_served(5)
restaurant.set_number_served(6)
restaurant.increment_number_served(2)
```

```
5 people have been served
6 people have been served!
8 total people have been served!!
```

9-5. Login Attempts: Add an attribute called `login_attempts` to your `User` class from 9-3. Write a method called `increment_login_attempts()` that increments the value of `login_attempts` by 1. Write another method called `reset_login_attempts()` that resets the value of `login_attempts` to 0.

Make an instance of the `User` class and call `increment_login_attempts()` several times. Print the value of `login_attempts` to make sure it was incremented propertly, and then call `reset_login_attempts()`. Print `login_attempts` again to make sure it was reset to 0.

In [14]:
```
class User():
    def __init__(self,first_name,last_name,age,city,school):
        self.first_name = first_name
        self.last_name = last_name
        self.age = age
        self.city = city
        self.school = school
    def describe_user(self):
        print(f'summary: {self.first_name}, {self.last_name}, {self.age}, {self.city}, {self.school}')
    def greet_user(self):
        print(f'Hello {self.first_name} {self.last_name}!')
    def login_attempts(self, number_of_attempts):
        self.number_of_attempts = number_of_attempts
        print(f'number of login attempts is {self.number_of_attempts}')
```

```
    def increment_login_attempts(self):
        self.number_of_attempts = self.number_of_attempts + 1
        print(f'number of login attempts is {self.number_of_attempts}')
    def reset_login_attempts(self):
        self.number_of_attempts = 0
        print(f'number of login attempts reset to {self.number_of_attempts}')

user_one = User('Erin','Kincade','20','ATL','USNA')
user_one.login_attempts(1)
user_one.increment_login_attempts()
user_one.increment_login_attempts()
user_one.increment_login_attempts()
user_one.login_attempts(user_one.increment_login_attempts())
user_one.reset_login_attempts()
user_one.login_attempts(user_one.reset_login_attempts())
```

```
number of login attempts is 1
number of login attempts is 2
number of login attempts is 3
number of login attempts is 4
number of login attempts is 5
number of login attempts is None
number of login attempts reset to 0
number of login attempts reset to 0
number of login attempts is None
```

9-6. Ice Cream Stand: An ice cream stand is a specific kind of restaurant. Write a class caled `IceCreamStand` that inherits from the `Restaurant` class you wrote in 9-1 or 9-4. Either version of the clas will work; just pick the one you like better. Add an attribute called `flavors` that stores a list of ice cream flavors. Write a method that displays these flavors. Create an instance of `IceCreamStand`, and call this method.

In [7]:
```
class IceCreamStand(Restaurant):

    def __init__(self, restaurant_name, cuisine_type):
        super().__init__(restaurant_name, cuisine_type)

    def flavors(self, list_flavors):
        self.list_flavors = list_flavors
        print(self.list_flavors)

flav_list = IceCreamStand('dairy queen','ice cream')
flav_list.flavors(['chocolate','vanilla','strawberry'])
```

```
['chocolate', 'vanilla', 'strawberry']
```

9-7. Admin: An administrator is a special kind of user. Write a class called `Admin` that inherits from the `User` class you wrote in 9-3 or 9-5. Add an attribute, `privileges`, that stores a list of strings like "*can add post*", "*can delete post*", "*can ban user*", and so on. Write a method called `show_privileges()` that lists the administrator's set of privileges. Create an instance of `Admin`, and call your method.

In [11]:
```
class Admin(User):
    def  __init__(self,first_name,last_name,age,city,school):
        super().__init__(first_name,last_name,age,city,school)
    def show_privileges(self, priv):
        self.priv = priv
        if self.priv == 'admin':
            print('This user can add post')
            print('This user can delete post')
            print('This user can ban user')
        else:
            print('This user may only add post')

one = Admin('Erin','Kincade','20','ATL','USNA')
one.show_privileges('admin')

two = Admin('Josh','Kincade','21','ATL','KSU')
two.show_privileges('user')
```

```
This user can add post
This user can delete post
This user can ban user
This user may only add post
```

9-8. Privileges: Write a separate `Privileges` class. The class should have one attribute, `privileges`, that stores a list of strings as described in 9-7. Move the `show_privileges()` method to this class. Make a `Privileges` instance as an attribute in the `Admin` class. Create a new instance of `Admin` and use your method to show its privileges.

In [ ]:
```
class Privileges():
    def __init__(self,privileges):
        import show_privileges
```

9-9. Battery Upgrade: Use the final verion of the `electric_car.py` from this lecture. Add a method to the `Battery` class called `upgrade_battery()`. This method should check the battery size and set the capacity to 100 if it isn't already. Make an electric car with a default battery size, call `get_range()` once, and the call `get_range()` a second time after upgrading the battery. You should see an increase in the car's range.

In [15]:
```
class ElectricCar(Car):
    """Models aspects of a car, specific to electric vehicles."""

    def __init__(self, manufacturer, model, year):
        """
        Initialize attributes of the parent class.
        Then initialize attributes specific to an electric car.
        """
        super().__init__(manufacturer, model, year)
        self.battery_size = 75

    def describe_battery(self):
        """Print a statement describing the battery size."""
        print(f"This car has a {self.battery_size}-kWh battery.")

    def get_descriptive_name(self): # overrides parent method!
        """Return a neatly formatted descriptive name."""
        long_name = str(self.year) + ' ' + self.manufacturer + ' ' + self.model + ' [Electric!]'
        return long_name.title()

class Battery():
    """A simple attempt to model a battery for an electric car."""

    def __init__(self, battery_size=60):
        """Initialize the batteery's attributes."""
        self.battery_size = battery_size

    def describe_battery(self):
        """Print a statement describing the battery size."""
        print("This car has a " + str(self.battery_size) + "-kWh battery.")

    def get_range(self):
```

```
        """Print a statement about the range this battery provides."""
        if self.battery_size == 60:
            rangec = 140
        elif self.battery_size == 85:
            rangec = 185

        message = "This car can go approximately " + str(rangec)
        message += " miles on a full charge."
        print(message)

    def upgrade_battery(self):
        if self.battery_size != 100:
            self.battery_size = 100

instance = ElectricCar('toyota','camery','2020')
instance = Battery()
instance.get_range()
instance.upgrade_battery()
instance.get_range()
```

```
This car can go approximately 140 miles on a full charge.
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
Cell In[15], line 51
     49 instance.get_range()
     50 instance.upgrade_battery()
---> 51 instance.get_range()

Cell In[15], line 39, in Battery.get_range(self)
     36 elif self.battery_size == 85:
     37     rangec = 185
---> 39 message = "This car can go approximately " + str(rangec)
     40 message += " miles on a full charge."
     41 print(message)

UnboundLocalError: cannot access local variable 'rangec' where it is not associated with a value
```

9-10. Imported Restaurant: Using your latest `Restaurant` class, store it in a module. Import `Restaurant` and make an instance of the class. Then call one of its methods to show that the `import` statement is working properly.

In [16]:
```
from restaurant import Restaurant

channel_club = Restaurant('the channel club', 'steak and seafood')
```

```
channel_club.describe_restaurant()
channel_club.open_restaurant()
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[16], line 1
----> 1 from restaurant import Restaurant
      3 channel_club = Restaurant('the channel club', 'steak and seafood')
      4 channel_club.describe_restaurant()

ModuleNotFoundError: No module named 'restaurant'
```

9-11. Imported Admin: Start with your work form 9-8. Store the classes `User`, `Privileges`, and `Admin` in one module. Import just the `Admin` class and make an instance of it. Then call `show_privileges()` to show that everything is working properly.

In [20]:
```
class User():
    """Represent a simple user profile."""

    def __init__(self, first_name, last_name, username, email, location):
        """Initialize the user."""
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.username = username
        self.email = email
        self.location = location.title()
        self.login_attempts = 0

    def describe_user(self):
        """Display a summary of the user's information."""
        print(f"\n{self.first_name} {self.last_name}")
        print(f"  Username: {self.username}")
        print(f"  Email: {self.email}")
        print(f"  Location: {self.location}")

    def greet_user(self):
        """Display a personalized greeting to the user."""
        print(f"\nWelcome back, {self.username}!")

    def increment_login_attempts(self):
        """Increment the value of login_attempts."""
        self.login_attempts += 1

    def reset_login_attempts(self):
        """Reset login_attempts to 0."""
```

```
        self.login_attempts = 0


class Admin(User):
    """A user with administrative privileges."""

    def __init__(self, first_name, last_name, username, email, location):
        """Initialize the admin."""
        super().__init__(first_name, last_name, username, email, location)

        # Initialize an empty set of privileges.
        self.privileges = Privileges()


class Privileges():
    """A class to store an admin's privileges."""

    def __init__(self, privileges=[]):
        self.privileges = privileges

    def show_privileges(self):
        print("\nPrivileges:")
        if self.privileges:
            for privilege in self.privileges:
                print(f"- {privilege}")
        else:
            print("- This user has no privileges.")

helper = Admin('erin','kincade','eak','gmail','ATL')
helper.show_privileges()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[20], line 59
     56             print("- This user has no privileges.")
     58 helper = Admin('erin','kincade','eak','gmail','ATL')
---> 59 helper.show_privileges()

AttributeError: 'Admin' object has no attribute 'show_privileges'
```

9-12. Multiple Modules: Store the `User` class in one module, and store the `Privileges` and `Admin` classes in a a seperate module. Set up the import statements so that you can create an instance of an `Admin` below. Then call `show_privileges()` to show that everything is working properly.

In [22]:
```
class User():
    """Represent a simple user profile."""

    def __init__(self, first_name, last_name, username, email, location):
        """Initialize the user."""
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.username = username
        self.email = email
        self.location = location.title()
        self.login_attempts = 0

    def describe_user(self):
        """Display a summary of the user's information."""
        print(f"\n{self.first_name} {self.last_name}")
        print(f"  Username: {self.username}")
        print(f"  Email: {self.email}")
        print(f"  Location: {self.location}")

    def greet_user(self):
        """Display a personalized greeting to the user."""
        print(f"\nWelcome back, {self.username}!")

    def increment_login_attempts(self):
        """Increment the value of login_attempts."""
        self.login_attempts += 1

    def reset_login_attempts(self):
        """Reset login_attempts to 0."""
        self.login_attempts = 0
```

9-13. Dice: Make a class `Die` with one attribute called `sides`, which has a default value of 6. Write a method called `roll_die()` that prints a random number between 1 and the number of sides the die has. Make a 6-sided die and roll it 10 times.

Make a 10-sided die and a 20-sided die. Roll each die 10 times.

In [23]:
```
from random import randint
x = randint(1, 6)

class Die():
    """Represent a die, which can be rolled."""

    def __init__(self, sides=6):
```

```
        """Initialize the die."""
        self.sides = sides

    def roll_die(self):
        """Return a number between 1 and the number of sides."""
        return randint(1, self.sides)

# Make a 6-sided die, and show the results of 10 rolls.
d6 = Die()

results = []
for roll_num in range(10):
    result = d6.roll_die()
    results.append(result)
print("10 rolls of a 6-sided die:")
print(results)

# Make a 10-sided die, and show the results of 10 rolls.
d10 = Die(sides=10)

results = []
for roll_num in range(10):
    result = d10.roll_die()
    results.append(result)
print("\n10 rolls of a 10-sided die:")
print(results)

# Make a 20-sided die, and show the results of 10 rolls.
d20 = Die(sides=20)

results = []
for roll_num in range(10):
    result = d20.roll_die()
    results.append(result)
print("\n10 rolls of a 20-sided die:")
print(results)
```

```
10 rolls of a 6-sided die:
[2, 2, 5, 3, 3, 3, 6, 5, 6, 6]

10 rolls of a 10-sided die:
[4, 2, 4, 10, 1, 1, 8, 10, 7, 5]

10 rolls of a 20-sided die:
[16, 15, 5, 13, 19, 18, 1, 10, 7, 19]
```

9-14. Lottery: Make a list or tuple containing a series of 10 numbers and five letters. Randomly select four numbers or letters from the list and print a message saying that any ticket matching these four numbers or letters wins a prize.

In [24]:
```
from random import choice

possibilities = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 'a', 'b', 'c', 'd', 'e']

winning_ticket = []
print("Let's see what the winning ticket is...")

# We don't want to repeat winning numbers or letters, so we'll use a
#   while loop.
while len(winning_ticket) < 4:
    pulled_item = choice(possibilities)

    # Only add the pulled item to the winning ticket if it hasn't
    #   already been pulled.
    if pulled_item not in winning_ticket:
        print(f"  We pulled a {pulled_item}!")
        winning_ticket.append(pulled_item)
```

```
Let's see what the winning ticket is...
  We pulled a e!
  We pulled a c!
  We pulled a 10!
  We pulled a 9!
```

9-15. Lottery Analysis: You can use a loop to see how hard it might be to win the kind of lottery you just modeled. Make a list or tuple called `my_ticket`. Write a loop that keeps pulling numbers until your ticket wins. Print a message reporting how many times the loop had to run to give you a winning ticket.

In [25]:
```
from random import choice

def get_winning_ticket(possibilities):
    """Return a winning ticket from a set of possibilities."""
    winning_ticket = []

    # We don't want to repeat winning numbers or letters, so we'll use a
    #   while loop.
    while len(winning_ticket) < 4:
        pulled_item = choice(possibilities)

        # Only add the pulled item to the winning ticket if it hasn't
```

```
        #   already been pulled.
        if pulled_item not in winning_ticket:
            winning_ticket.append(pulled_item)

    return winning_ticket

def check_ticket(played_ticket, winning_ticket):
    # Check all elements in the played ticket. If any are not in the
    #   winning ticket, return False.
    for element in played_ticket:
        if element not in winning_ticket:
            return False

    # We must have a winning ticket!
    return True

def make_random_ticket(possibilities):
    """Return a random ticket from a set of possibilities."""
    ticket = []
    # We don't want to repeat numbers or letters, so we'll use a while loop.
    while len(ticket) < 4:
        pulled_item = choice(possibilities)

        # Only add the pulled item to the ticket if it hasn't already
        #   been pulled.
        if pulled_item not in ticket:
            ticket.append(pulled_item)

    return ticket


possibilities = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 'a', 'b', 'c', 'd', 'e']
winning_ticket = get_winning_ticket(possibilities)

plays = 0
won = False

# Let's set a max number of tries, in case this takes forever!
max_tries = 1_000_000

while not won:
    new_ticket = make_random_ticket(possibilities)
    won = check_ticket(new_ticket, winning_ticket)
    plays += 1
    if plays >= max_tries:
```

```
        break

if won:
    print("We have a winning ticket!")
    print(f"Your ticket: {new_ticket}")
    print(f"Winning ticket: {winning_ticket}")
    print(f"It only took {plays} tries to win!")
else:
    print(f"Tried {plays} times, without pulling a winner. :(")
    print(f"Your ticket: {new_ticket}")
    print(f"Winning ticket: {winning_ticket}")
```

```
We have a winning ticket!
Your ticket: [6, 9, 1, 'c']
Winning ticket: ['c', 1, 9, 6]
It only took 1400 tries to win!
```

9-16. Python Module of the Week: One excellent resource for exploring the Python standard library is a site called `Python Module of the Week`. Go to https://pymotw.com/ and look at the table of contents. Find a module that looks interesting to you and read about it, perhaps starting with the `random` module. Write three things you learned about the module below.

1. random returns the next random FLOATING point 2. you can set a range for random 3. to do this use a for loop