9_classes

create and use a class

class can represent a group of objects in which they all share certain attributes. so each individual is already equipped with the general traits and can be further personalized.
A function that is a part of the class is a method

```
class Cat:
        """it imitates cats"""
        def __init__ (self, name, age):
                """provides name and age"""
                self.name=name
                self.age=age
        # you can replace self with any specific instances of the class. age and 
name is the attributes assigned with this class.
        def being_cute(self):
                """cat will act cutely"""
                print(f'look how cute {self.name} are')
        # we assgin a method, being_cute(), to let the function perform certain 
task.
my_cat=Cat('kelly','7')
print(my_cat.name)
```

```
#kelly
my_cat.being_cute()
#look how cute kelly are
```

working with class and instance

set default value for an attribute

```
#setting a default value does not require us to set up a parameter
class Car:
        def __init__(self,make,model,year):
                """it describes the car"""
                self.make=make
                self.model=model
                self.year=year
                self.mileage_reading=0
        def read_odometer(self):
                """it shows the mileage of the car"""
                print(f'this car has {self.mileage_reading}miles on it')
mycar=Car('audi','A4','2019')
mycar.read_odometer()
```

modify attribute value

directly

```
mycar.mileage_reading=66
mycar.read_odometer()
```

Use method to update

```
#or we could call a specific method to modify a value
class Car:
        def __init__(self,make,model,year):
                """it describes the car"""
                self.make=make
                self.model=model
                self.year=year
                self.mileage_reading=0
        def read_odometer(self):
                """it shows the mileage of the car"""
                print(f'this car has {self.mileage_reading}miles on it')
        def update_odometer(self,new_mileage):
                """enters the new mileage to update """
                self.mileage_reading=new_mileage
```

```
mycar=Car('audi','A4','2019')
mycar.update_odometer(34)
mycar.read_odometer()
#this car has 34miles on it
```

increment

```
#sometimes we only know the increment but not the final value we want
class Car:
        def __init__(self,make,model,year):
                """it describes the car"""
                self.make=make
                self.model=model
                self.year=year
                self.mileage_reading=0
        def read_odometer(self):
                """it shows the mileage of the car"""
                print(f'this car has {self.mileage_reading}miles on it')
        def increase_odometer(self,increment):
                """it increase the odometer"""
                self.mileage_reading+= increment
mycar=Car('audi','A4','2019')
mycar.increase_odometer(88)
mycar.read_odometer()
#this car has 88miles on it
```

Inheritance

if the class you want to create is similar with an existing class, you can use inheritance to copy all the methods and attributes of the "parent" class and add new attributes and methods on your child class.

init method for child class

```
#this time we want to create a child class, electric car, without adding any new
methods or attributes.
class Electric_Car(Car):
        """a specific kind of car"""
        def __init__(self,make,model,year):
                """initialize attributes"""
                super().__init__(make,model,year)
my_byd=Electric_Car('byd','han','2018')
my_byd.read_odometer()
#this car has 0miles on it
```

```
# 1. the child class has to include parent class in parentheses
# 2. we use super()to call the method in the "super class" and inherit it.
```

new methods and attributes for child class

```
class Electric_Car(Car):
        """a specific kind of car"""
        def __init__(self,make,model,year):
                """initialize attributes"""
                super().__init__(make,model,year)
                self.battery_size=100
        def describe_battery(self):
                """shows battery's info"""
                print(f'The size of the battery is {self.battery_size}')
my_byd=Electric_Car('byd','han','2018')
my_byd.describe_battery()
#The size of the battery is 100
```

override method

```
#some method may be obsolete. override it with a new method in child class.
class Car:
        def __init__(self,make,model,year):
                """it describes the car"""
                self.make=make
                self.model=model
                self.year=year
                self.mileage_reading=0
        def read_odometer(self):
                """it shows the mileage of the car"""
                print(f'this car has {self.mileage_reading}miles on it')
        def increase_odometer(self,increment):
                """it increase the odometer"""
                self.mileage_reading+= increment
        def gas_storage(self,gas_storage):
                """it shows the maximum storage"""
                print(f'this car is equipped with {gas_storage} storage place')
class Electric_Car(Car):
        """a specific kind of car"""
        def __init__(self,make,model,year):
                """initialize attributes"""
                super().__init__(make,model,year)
                self.battery_size=100
        def gas_storage(self,gas_storage):
                """bro, this is not a gas filled car"""
                print('this car does not have a gas storage')
my_byd=Electric_Car('byd','han','2018')
my_byd.gas_storage(15)
#this car does not have a gas storage
```

instances as attributes

when we add more details to the class we have, we may notice that there are more and more information about certain subject. in situations like this, part of one class can be written as a separate class. Later, we can use the new instance as an attribute of the original class.

```

class Battery:
        """singles out battery as a class"""
        def __init__(self,volume=100):
                """initialize its attribute"""
                self.volume=volume
        def charge(self):
                """charge the battery"""
                print('the battery is charging')

class Car:
        def __init__(self,make,model,year):
                """it describes the car"""
                self.make=make
                self.model=model
                self.year=year
                self.mileage_reading=0
        def read_odometer(self):
                """it shows the mileage of the car"""
                print(f'this car has {self.mileage_reading}miles on it')
        def increase_odometer(self,increment):
                """it increase the odometer"""
                self.mileage_reading+= increment
        def gas_storage(self,gas_storage):
                """it shows the maximum storage"""
                print(f'this car is equipped with {gas_storage} storage place')



class Electric_Car(Car):
        """a specific kind of car"""
        def __init__(self,make,model,year):
                """initialize attributes"""
                super().__init__(make,model,year)
                self.battery=Battery()
        def gas_storage(self,gas_storage):
                """bro, this is not a gas filled car"""
                print('this car does not have a gas storage')
my_byd=Electric_Car('byd','han','2018')



my_byd.battery.charge()
```

import class

import classes

we can store classes in modules and import them whenever needed.

```
#if you save your file as car.py and the class is Car
from car import Car
# if you want to import multiple classes, juse use commas to separate
from car import Car, Electric_Car
```

import entire module

```
import car
my_byd=car.Electric_Car('byd','han','2018')
```

python library

as we get more familiar with import, we could import a variety of outside resources in python library to complement our own work.