Creating and Instantiating a Class

In [1]:

```
# creating your first class:
class Car():
    pass  #simply using a placeholder until we add more code
```

In [2]:

```
# creating your first class:
class Car():          #Parenthesis are optional here
    pass  #simply using a placeholder until we add more code
ford = Car()  #creates an instance of the Car class and stores in Variable Ford
print(ford)
```

```
<__main__.Car object at 0x0000018A823DDF70>
```

In [3]:

```
# creating your first class:
class Car():          #Parenthesis are optional here
    pass  #simply using a placeholder until we add more code
ford = Car()#creates an instance of the Car class and stores in Variable Ford
subaru = Car()
print(hash(ford))
print(hash(subaru))
# hash outputs a numberical representation of the location in memory for the variable
```

```
105900147119
105900144098
```

In [7]:

```
"""
Exercise: Create a class called "Animals," and create two instances from it. Use
two variables with names of "lion" and "tiger."
"""
class Animals:
    pass
lion = Animals()
tiger = Animals()
print(hash(lion))
print(hash(tiger))
```

```
105900149785
105900149824
```

Attributes

In [1]:

```
# how to define a class attribute
class Car():
    sound = "Beep" #all car objects will have this attribute and value
    color = "red"  #all car objects will have this color attribute and value
ford = Car()
print(ford.color)  #known as dot syntax
# attributes are onlyavailable within classes they are defined, so in order to access a
#create an instance
```

```
red
```

In [3]:

```
#changing the value of an attribute
class Car():
    sound = "Beep"
    color = "red"
```

```
ford = Car()
print(ford.sound)
ford.sound = "Honk"
print(ford.sound)
```

```
Beep
Honk
```

In [4]:

```
# using the init method to give instances personalized attributes upon creadtion
class Car():
    def __init__(self,color):
        self.color = color     #sets the attribute color to the value passed
ford = Car("blue")  #instantiating a car class  with color Blue
print(ford.color)
```

```
blue
```

In [6]:

```
#defining different values for multiple instances
class Car():
    def __init__(self,color,year):
        self.color = color
        self.year = year
ford =Car("Blue",2021)
subaru = Car("Red",2020)
print(ford.color,ford.year)
print(subaru.color,subaru.year)
```

```
Blue 2021
Red 2020
```

In [7]:

```
# using and accessing global class attributes

class Car():
    sound = 'beep'  # global attribute, accessible through the class itself

    def __init__(self, color):
        self.color = 'blue'   # instance specific attribute, not accessible through the

print(Car.sound)

# print(Car.color)  will not work, as color is only available to instances of the Car c

ford = Car('blue')

print(ford.sound, ford.color)   # color will work as this is an instance
```

```
beep
beep blue
```

In [10]:

```
"""
Exercise: Create a Dog class that has one global attribute and two instance level
attributes. The global attribute should be “species” with a value of “Canine.”
The two instance attributes should be “name” and “breed.” Then instantiate
two dog objects, a Husky named Sammi and a Chocolate Lab named Casey
"""
class Dogs():
    species = 'Canine'
    def __init__(self,name,breed):
        self.name = name
        self.breed =breed
dog1 = Dogs('Sammi',"Husky")
dog2 = Dogs('Casey','Chocolate Lab')
```

```
print(Dogs.species)
print(dog1.name,dog1.breed)
print(dog2.name,dog2.breed)
```

```
Canine
Sammi Husky
Casey Chocolate Lab
```

In [14]:

```
"""
Exercise: Create a Person class that has a single instance level attribute of
“name.” Ask the user to input their name, and create an instance of the Person
class with the name they typed in. Then print out their name.
"""
class Person():
    def __init__(self,name):
         self.name =name
name = input("Enter Your Name Please: ")
p1 = Person(name)
print(p1.name)
```

```
Enter Your Name Please: Hrishikesh
Hrishikesh
```

Methods

In [15]:

```
# defining and calling out first class method
class Dog():
    def makeSound(self):
        print("bark")
sam = Dog()
sam.makeSound()
```

```
bark
```

In [2]:

```
#using the self keyword to access attributes within class methods
class Dog():
    sound = 'Bark'
    def makeSound(self):
        print(self.sound)  #self required to access attributes defined in class
sam = Dog()
sam.makeSound()
```

```
Bark
```

In [4]:

```
# understanding which methods are accessible via the class itself and class instances
class Dog():
    sound = "bark"
    def makeSound(self):
        print(self.sound)
    def printInfo():
        print("I am a Dog")
Dog.printInfo() #able to run printInfo as self is not its parameter
# Dog.makeSound() #would produce an error as self is in reference to instances only
sam = Dog()
sam.makeSound()
#sam.printInfo() will produce error as instances require self as parameter
```

```
I am a Dog
bark
```

In [5]:

```
# writing methods that accept parameters
```

```
class Dog():
    def showAge(self,age):
        print(age)   #does not need self, age is referencing the parameter not an attri
sam = Dog()
sam.showAge(6) # passing the int 6 as an argument to showAge method
```

```
6
```

In [6]:

```
# using methods to set or return attribute values, proper programming practice
class Dog():
    name = " " #would normally use the init method to declare, this is for testing purp
    def setName(self,new_name):
        self.name = new_name #declares the new value for the name attribute
    def getName(self):
        return self.name  #returns the value of the name attribute
sam = Dog()
sam.setName("Noburu Wataya")
print(sam.getName())
```

```
Noburu Wataya
```

In [9]:

```
# incrementing/decrementing attribute values with method, best programming practice
class Dog():
    age =5
    def happyBday(self):
        self.age += 1
sam = Dog()
print(sam.age)
sam.happyBday() #calls method to increment value by one
print(sam.age)
```

```
5
6
```

In [10]:

```
# calling a class method from another method
class Dog():
    age = 6
    def getAge(self):
        return self.age
    def printInfo(self):
        if self.getAge() < 10: #need self to call other method for an instance
            print("Doggo is just a Kid")
sam = Dog()
sam.printInfo()
```

```
Doggo is just a Kid
```

In [11]:

```
# using magic methods
class Dog():
    def __str__(self):
        return "This is a Dog class"
sam = Dog()
print(sam) # will print the return of string magic method
```

```
This is a Dog class
```

In [19]:

```
"""
Exercise :Create a class definition of an animal that has a species attribute and
both a setter and getter to change or access the attributes value. Create an
instance called “lion,” and call the setter method with an argument of “feline.”
Then print out the species by calling the getter method
"""
class Animal():
    def __init__(self):
        species = ''

    def setSpecies(self, species):
        self.species = species

    def getSpecies(self):
        return self.species

lion = Animal()

lion.setSpecies('Feline')

lion.getSpecies()
```

Out[19]:

```
'Feline'
```

In [20]:

```
"""
Exercise: Create a class Person that takes in a name when instantiated but
sets an age to 0. Within the class definition setup, a setter and getter that will
ask the user to input their age and set the age attribute to the value input.
Then output the information in a formatted string as “You are 64 years old.”
Assuming the user inputs 64 as their age.
"""
class Person():
    def __init__(self, name):
        self.name = name
        self.age = 0

    def setAge(self, age):
        self.age = age

    def getAge(self):
        return self.age

p = Person('Kathy')

num = input('How old are you? ')

p.setAge(num)

print('You are {} years old.'.format(p.getAge()))
```

```
How old are you? 64
You are 64 years old.
```

Inheritance

In [1]:

```
# inheriting a class and accessing the inherited method
class Animal():
    def makeSound(self):
        print("Roar")
class Dog(Animal):   #inheriting animal class
    species = "Canine"
sam = Dog()
sam.makeSound()
lion = Animal()
```

```
Roar
```

In [2]:

```
#using the super() method to declare inherited attributes
class Animal():
    def __init__(self,species):
        self.species = species
class Dog(Animal):
    def __init__(self, species, name):
        self.name = name
        super().__init__(species)  #using super to declare the species attribute define

sam = Dog("Canine","Sammi")
print(sam.species)
```

```
Canine
```

In [3]:

```
# overriding methods defined in the superclass
class Animal():
    def makeSound(self):
        print("roar")
class Dog(Animal):
    def makeSound(self):
        print("bark")
sam , lion = Dog(), Animal()  #multiple declaration on single line
sam.makeSound()
lion.makeSound()
```

```
bark
roar
```

In [4]:

```
# Inheriting multiple classes
class Physics():
    gravity = 9.8
class Automobile():
    def __init__(self, make, model,year):
        self.make,self.model,self.year = make,model,year
class Ford (Physics,Automobile):
    def __init__(self,model,year):
        Automobile.__init__(self,"Ford",model,year)
truck = Ford("F-150",2018)
print(truck.gravity, truck.make)
```

```
9.8 Ford
```

In [5]:

```
"""
Execise:Good Guys/Bad Guys: Create three classes, a superclass called “Characters”
that will be defined with the following attributes and methods:
a. Attributes: name, team, height, weight
b. Methods: sayHello
The sayHello method should output the statement “Hello, my name is Max and
I’m on the good guys”. The team attribute should be declared to a string of
either “good” or “bad.” The other two classes, which will be subclasses, will
be “GoodPlayers” and “BadPlayers.” Both classes will inherit “Characters” and
super all the attributes that the superclass requires. The subclasses do not need
any other methods or attributes. Instantiate one player on each team, and call
the sayHello method for each. The output should result in the following:
 >>> "Hello, my name is Max and I'm on the good guys"
 >>> "Hello, my name is Tony and I'm on the bad guys"
"""
```

```
class Characters():
    def __init__(self, name, team, height, weight):
        self.name = name
        self.team = team
        self.height = height
        self.weight = weight
    def sayHello(self):
        print("Hello, my name is {} and I'm on the {} guys.".format(self.name, self.tea
class Good(Characters):
    def __init__(self, name, height, weight):
        super().__init__(name, 'good', height, weight)

class Bad(Characters):
    def __init__(self, name, height, weight):
        super().__init__(name, 'bad', height, weight)

char1 = Good('Max', '5\'11"', 183)
char2 = Bad('Tony', '6\'3"', 201)

char1.sayHello()
char2.sayHello()
```

```
Hello, my name is Max and I'm on the good guys.
Hello, my name is Tony and I'm on the bad guys.
```

Project : BlackJack

In [ ]:

```
# importing necessary functions
from random import randint
from IPython.display import clear_output

#create the blackjack class, which will hold all game method and attributes
class Blackjack():
    def __init__(self):
        self.deck = []    #empty list
        self.suits = ("Spades","Hearts","Diamonds","Clubs")
        self.values=(2,3,4,5,6,7,8,9,10,"J","Q","K","A")

#create method that creates a deck of 52 cards,each card should be a tuple with a value
    def makeDeck(self):
        for suits in self.suits:
            for value in self.values:
                self.deck.append((value,suits))
#method to pop a card from deck using a random index value
    def pullCard(self):
        return self.deck.pop(randint(0,len(self.deck) - 1))
#create a class for the dealer and player objects
class Player():
    def __init__(self, name):
        self.name = name
        self.hand = []
    def addCard(self,card):
        self.hand.append(card)
# if not dealers's turn then only show one of his cards, otehrwise show all
    def showHand(self,dealer_start = True):
        print(f"\n{self.name}")
        print("==========")
        for i in range(len(self.hand)):
            if self.name == "Dealer" and i == 0 and dealer_start:
                print("- of -")#hide first card
            else:
                card =self.hand[i]
                print(f"{card[0]} of {card[1]}")
        print("Total = {}".format(self.calcHand(dealer_start)))
# if not dealer's turn them only give back total of second card
    def calcHand(self, dealer_start = True):
        total = 0
        aces = 0 #calculate aces afterwards
        card_values = {1:1,2:2,3:3,4:4,5:5,6:6,7:7,8:8,9:9,10:10, "J":10,"Q":10,"K":10,
        if self.name == "Dealer" and dealer_start:
            card = self.hand[1]
            return card_values[card[0]]
        for card in self.hand:
            if card[0] == "A":
                aces += 1
            else:
                total += card_values[card[0]]
        for i in range(aces):
            if total + 11 > 21:
                total +=1
            else :
                total += 11
        return total

game = Blackjack()
game.makeDeck()

name = input("What is your name?")
playing = True
while playing:
    player = Player(name)
    dealer = Player("Dealer")
    #add two cards to the dealer and player hand
    for i in range(2):
        player.addCard(game.pullCard())
        dealer.addCard(game.pullCard())

    player.showHand()
    dealer.showHand()

    player_bust = False #variable to keep track of player going over 21
    while input("Would you like to stay or hit?").lower() != "stay":
        clear_output()
        #pull card and put into player's hand
        player.addCard(game.pullCard())
        #show both hands using method
        player.showHand()
        dealer.showHand()
        # check if over 21
        if player.calcHand() > 21:
            player_bust = True    # player busted, keep track for later
            print("You Lose!")
            break
    #handling the dealer's turn, only if the player did'nt bust
    dealer_bust = False
    if not player_bust:
        while dealer.calcHand(False) < 17 :
            #pull card and put into player's hand
            dealer.addCard(game.pullCard())
            #check if over 21
            if dealer.calcHand(False) > 21:
                dealer_bust = True
                print("You win!")
                break

    clear_output()

    # show both hands using method
    player.showHand()
    dealer.showHand(False)

    # calculate a winner
    if player_bust:
        print('You busted, better luck next time!')
    elif dealer_bust:
        print('The dealer busted, you win!')
    elif dealer.calcHand(False) > player.calcHand():
        print('Dealer has higher cards, you lose!')
    elif dealer.calcHand(False) < player.calcHand():
        print('You beat the dealer! Congrats!')
    else:
        print('You pushed, no one wins!')

    print('Type "quit" to stop playing...')
    ans = input('Would you like to play again? ').lower()

    if ans == 'quit':
        playing = False

    clear_output()
```

```
Jackal
==========
2 of Spades
10 of Spades
9 of Clubs
Total = 21

Dealer
==========
3 of Spades
10 of Clubs
8 of Clubs
Total = 21
You pushed, no one wins!
Type "quit" to stop playing...
```