print("happy birthday".upper())

print("happy birthday".lower())

print("happy birthday".capitalize())

Conditions

if condition:

elif cond2:

else:

statements

out of if condition

for I in rang(10):

states

while cond:

statements

items = ["book", "pen", "pencil"]

items.insert(2,"marker")

print(items)

print(items[2])

items.append("Liza")

print (len(items), items)

class Dog:

    def \_\_init\_\_(self, name, color):

        self.name = name

        self.color = color

    def bark(self):

        print("Woof!")

fido = Dog("Fido", "brown")

print(fido.name)

fido.bark()

class Dog:

    def \_\_init\_\_(self, name, color):

        self.name = name

        self.color = color

    def bark(self):

        print("Woof! My name is ", self.name)

class Cat:

    def \_\_init\_\_(self, name, eyes):

        self.name = name

        self.eyes = eyes

    def mao(self):

        print("Miaw...My name is ", self.name, "my eyes are ", self.eyes)

fido = Dog("Fido", "brown")

print(fido.name)

fido.bark()

# super class

class Wolf:

    def \_\_init\_\_(self, name, color):

        self.name = name

        self.color = color

    def bark(self):

        print("Grr... I am ", self.name)

# sub class

class Dog(Wolf):

    def bark(self):

        print("Woof")

husky = Dog("Max", "grey")

husky.bark()

Foxy=Wolf("Capo", "white")

Foxy.bark()

jiji = Cat("jiji", "blue")

jiji.mao()

More magic methods for common operators:

**\_\_sub\_\_**for -

**\_\_mul\_\_**for \*

**\_\_truediv\_\_**for /

**\_\_floordiv\_\_** for //

**\_\_mod\_\_**for %

**\_\_pow\_\_**for \*\*

**\_\_and\_\_**for &

**\_\_xor\_\_**for ^

**\_\_or\_\_**for |

def sayHi():

class Rectangle:

def \_\_init\_\_(self, width, height):

self.width = width

self.height = height

def calculate\_area(self):

return self.width \* self.height

@classmethod

def new\_square(cls, side\_length):

return cls(side\_length, side\_length)

square = Rectangle.new\_square(5)

print(square.calculate\_area())

class Pizza:

    def \_\_init\_\_(self, toppings):

        self.toppings = toppings

    @staticmethod

    def validate\_topping(topping):

        if topping == "pineapple":

            raise ValueError("No pineapples!")

        else:

            return True

ingredients = ["cheese", "onions", "shit"]

if all(Pizza.validate\_topping(i) for i in ingredients):

    pizza = Pizza(ingredients)

    print(ingredients)

class Pizza:

    def \_\_init\_\_(self, toppings):

        self.toppings = toppings

    @property

    def pineapple\_allowed(self):

        return False

pizza = Pizza(["cheese", "tomato","pineapple"])

print(pizza.pineapple\_allowed)

pizza.pineapple\_allowed = True