⇒ Line continuation operator :-

\ it tells python that rest of code is on the next line.

⇒ Nesting :-

```
for i in range(10):
    for j in range(10):
        print('____')
```

10 x 10 = 100

" ____ "

("hmm".replace()) → int, float, string, None

⇒ string ⇐

```
def function_add(a, b):
    return (a+b)
```

⇐ This must have some type

"Rohit bought a car".replace('Rohit', 'Vishal')

↳ "Vishal bought a car".replace('bought', 'sold')

"Vishal sold a car" ⇐

Classes

$$\begin{tikzcd} & \text{Classes} \arrow[dl] \arrow[dr] & \\ \text{Properties} & & \text{Methods} \end{tikzcd}$$

Properties (show some set of values)

Methods (show or do some action)

Examples

Human Being :-

Property :- They have two eyes.

Method :- they can see with the eyes.

Integers :-

property :- $\underbrace{x.\text{real}}$ (property) $\hookrightarrow$ Denotes a value

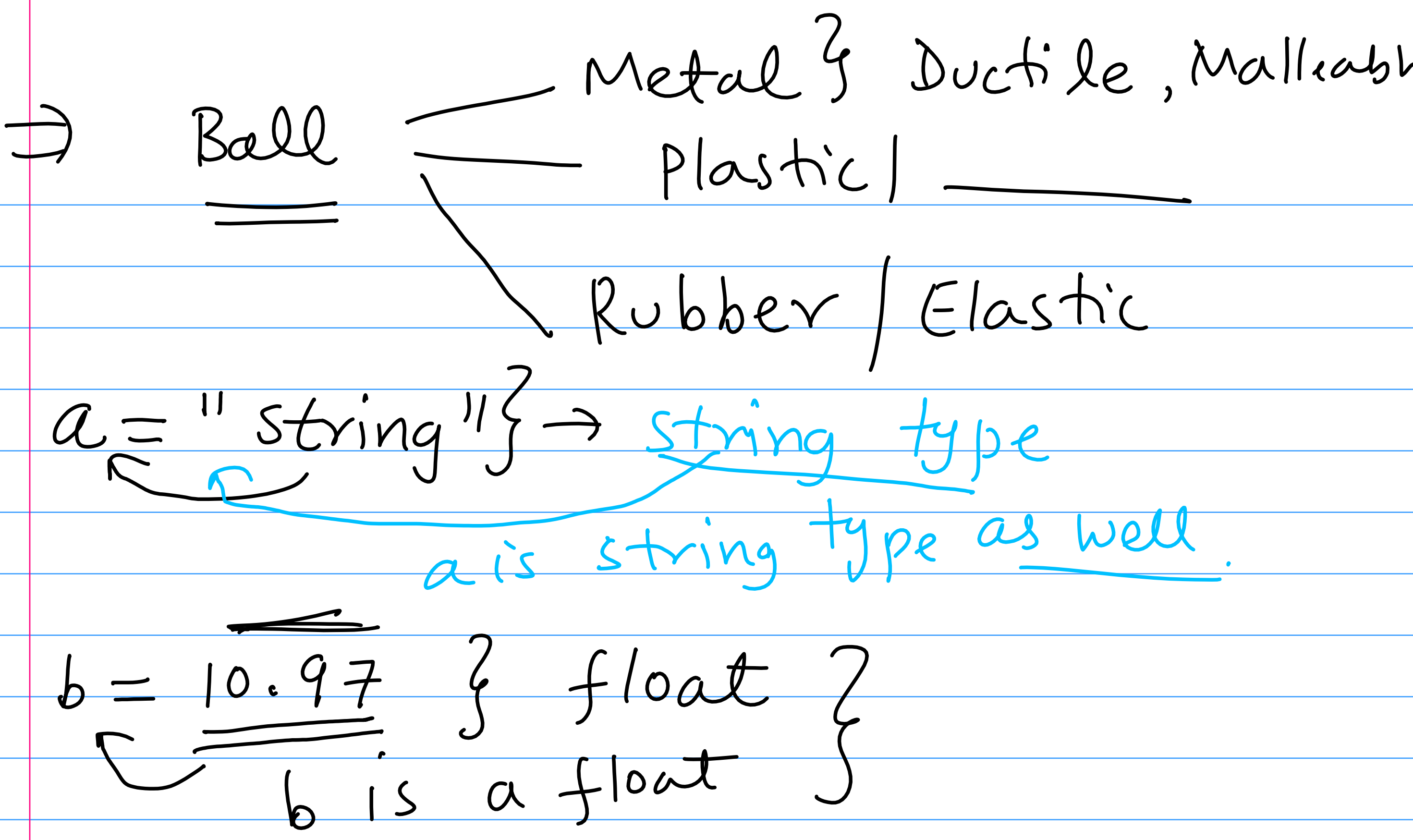
method :- $\underbrace{x.\text{conjugate()}}$ $\hookrightarrow$ action.

X

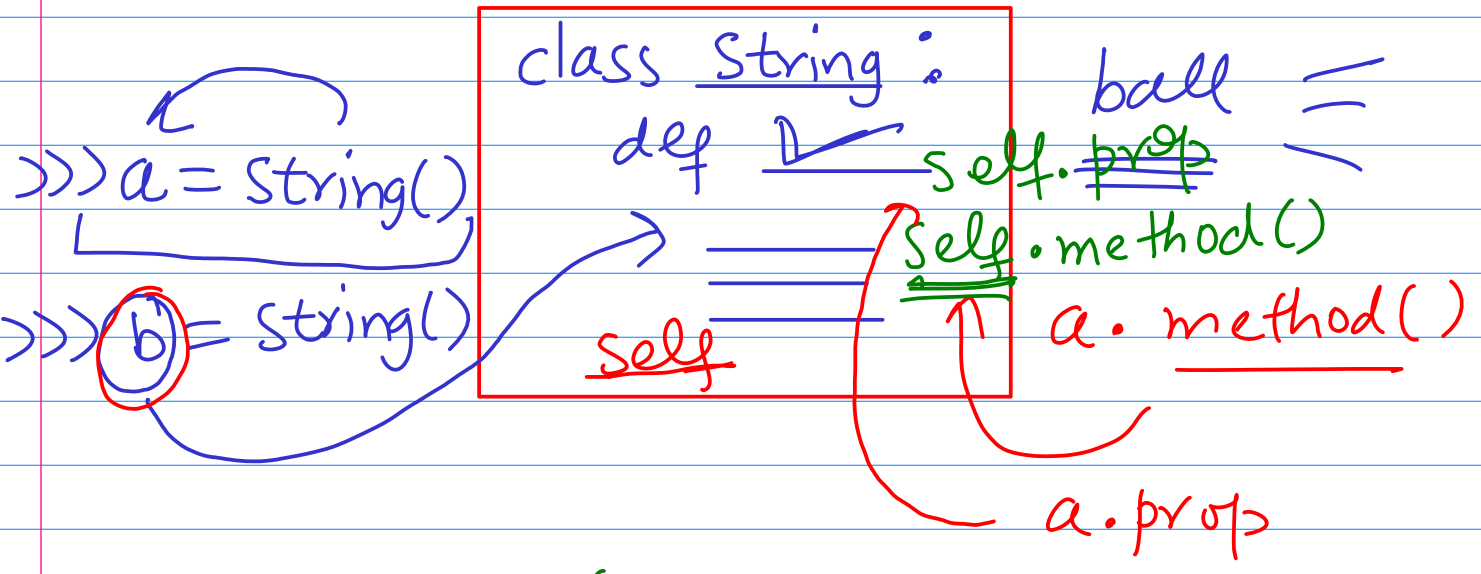
Complex Number :- $10 \boxed{+} 2j$

conjugate :- $10 - 2j$

[This is an action]



⇒ Ball
Metal } Ductile, Malleabl
Plastic |
Rubber | Elastic
a = "string" } → string type
a is string type as well.
b = 10.97 } float
b is a float



def method(self, a, b, c - - - . .)

⇒ A simple class :

```
class Human :          | class Human:
      pass             |     def __init__(self,....):
                       |         ______
                       |         ______
                       |         ______
```

⇒ Student Class :-

- Name
- Roll No.
- Gender
- Address/Phone No./Email.

```
class Student:
    def __init__(self, Name, Roll, Gender, Email)
        self.name = Name
        self.Roll = Roll.
        self. Gender -
```

Initialization

(How is it happening? __add__())

Strings } → a = "Shubham"
b = "Kumar"

$a + b \Rightarrow$ "ShubhamKumar" c

Diff. properties

Diff. props.

This is possible because of special methods

Inheritance is another really important property of classes.

Dunder Methods

Double Underscore Methods