Python

OOPS

Chapter 17

Suresh techs

Suresh techs – Technology and Entertainment

Subscribe

Subscribe: You don’t miss our future videos

Doubts

Follow In Instagram
#sureshtechs

- Link in the description
- Clear your doubts chatting with me

OOPS

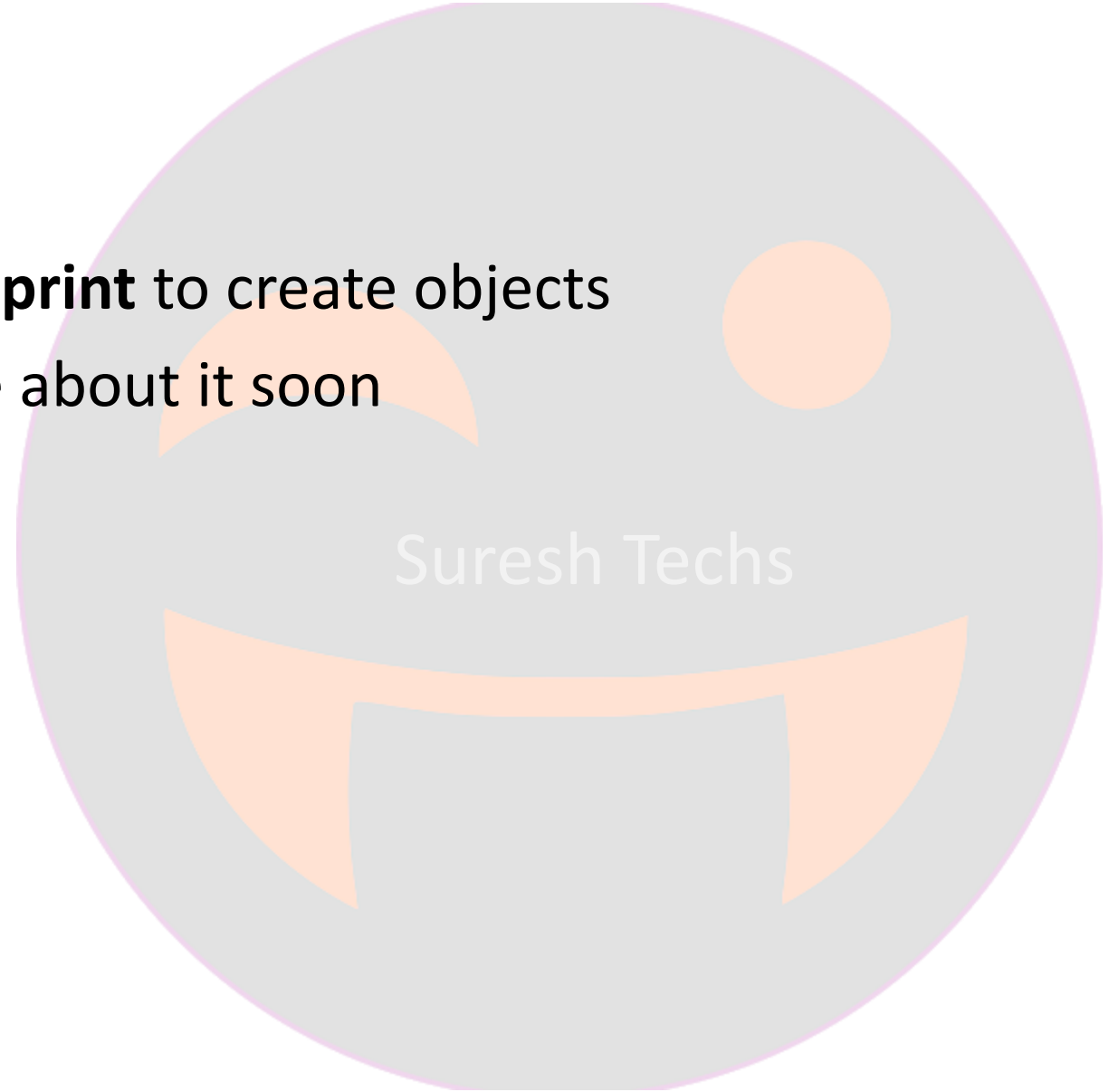
- **O**bject **O**riented **P**rogramming **S**ystem
- Class & Object
- Watch Java In 10 Minutes – Link in the description box
- https://youtu.be/cM82qnE_TPc

Object?

Anything that has state and behavior is called an object

Class?

- Class is a **blueprint** to create objects
- Will talk more about it soon



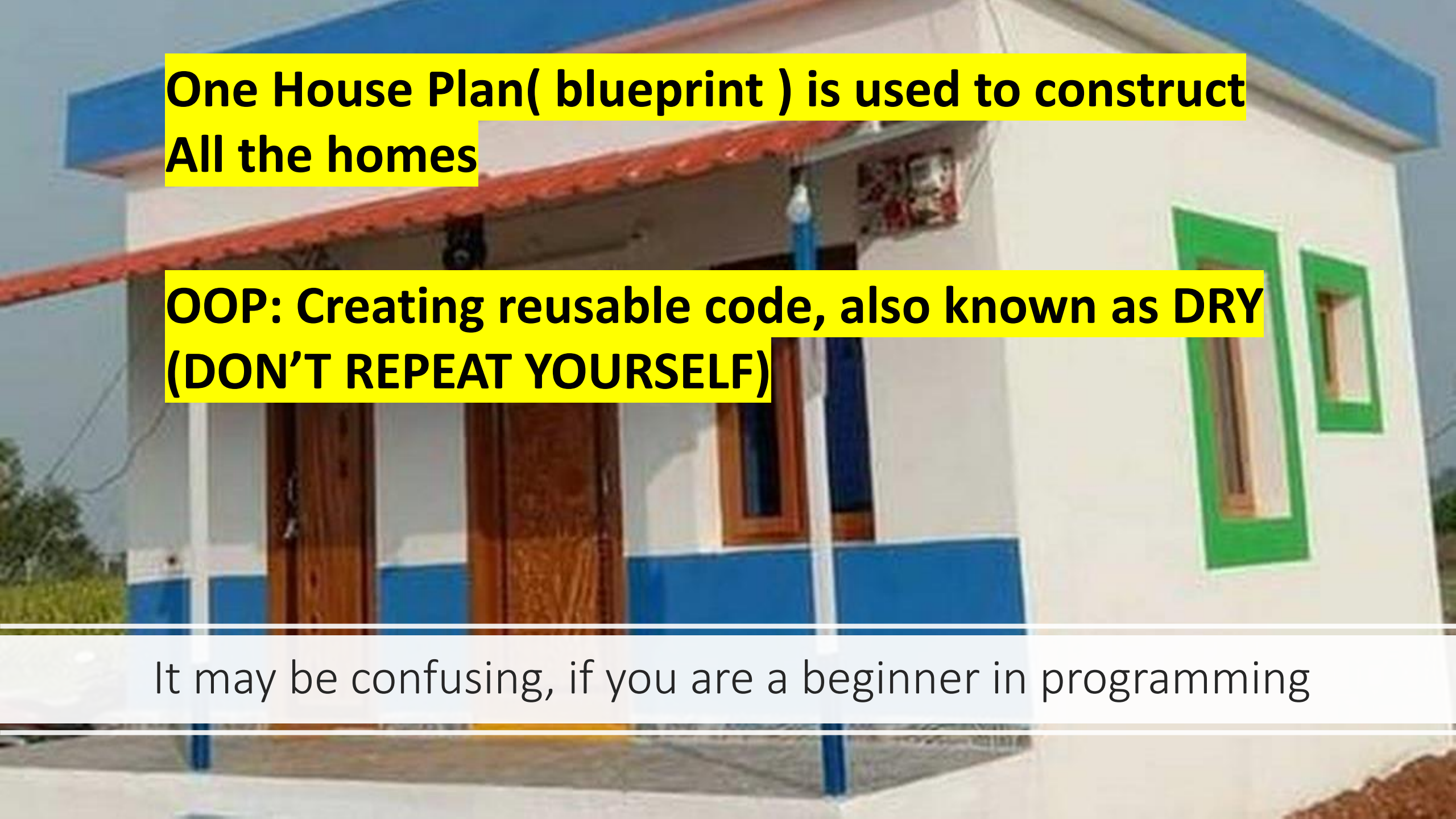
State and behavior of bottle

- Attributes/Properties/State:
  - Id
  - Color
  - Capacity
  - Height
- Functions/Behavior:
  - Wash
  - Set Cap
  - Fill Water

```
id=1
color = 'red'
capacity = 1
height = 30

def wash():
    print('washing')

def setCap():
    print('setting cap')

def fillWater():
    print('fill water')
```

What if you want to create 10 bottles?

- Very difficult right!
- That’s why we use oops (class, object) concept
- **Reusability of the code**, also known as DRY(Don’t repeat yourself)


One House Plan(blueprint) is used to construct All the homes
OOP: Creating reusable code, also known as DRY (DON’T REPEAT YOURSELF)
It may be confusing, if you are a beginner in programming

class?

- Class is a blueprint to create objects.
- One blueprint/class is enough to create n number of objects(bottles)
- Let’s create a class
- Syntax:
  - **class** className:

    #attributes(properties/state)

    #functions(methods/behavior)

Simple class

```
class Bottle:
    pass
```


Can I create doc string in a class?

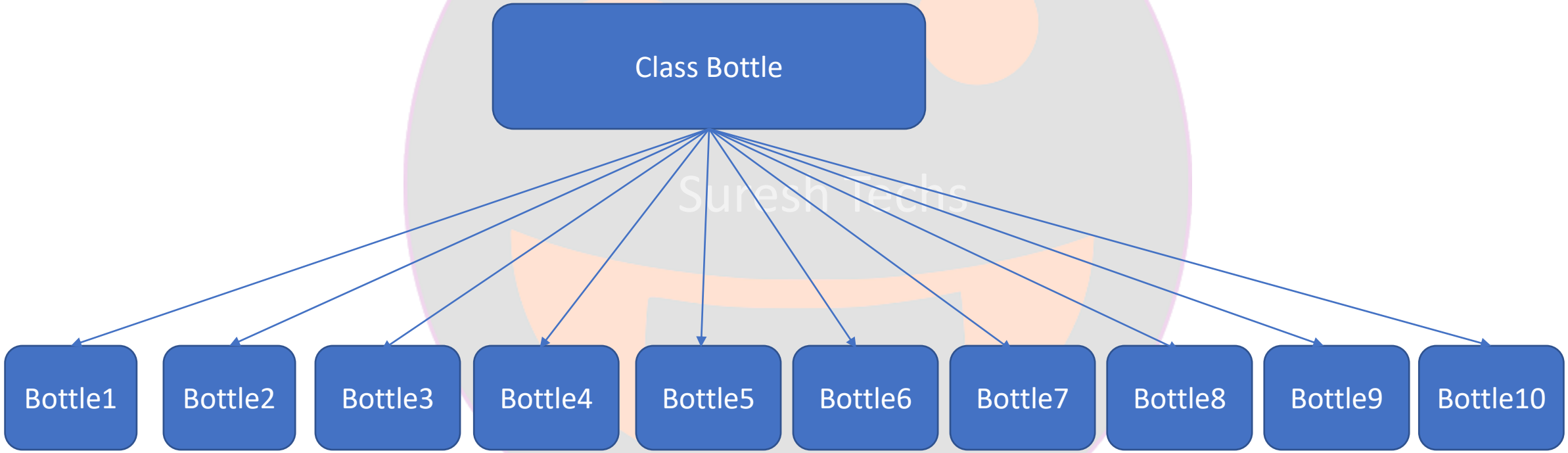
Let’s write Bottle class

```
class Bottle:
    id=1
    color = 'red'
    capacity = 1
    height = 30

    def wash():
        print('washing')

    def setCap():
        print('setting cap')

    def fillWater():
        print('fill water')
```

Blue print of the bottle

We can now create n number bottles using this blueprint/class

10 Bottles using class & Object

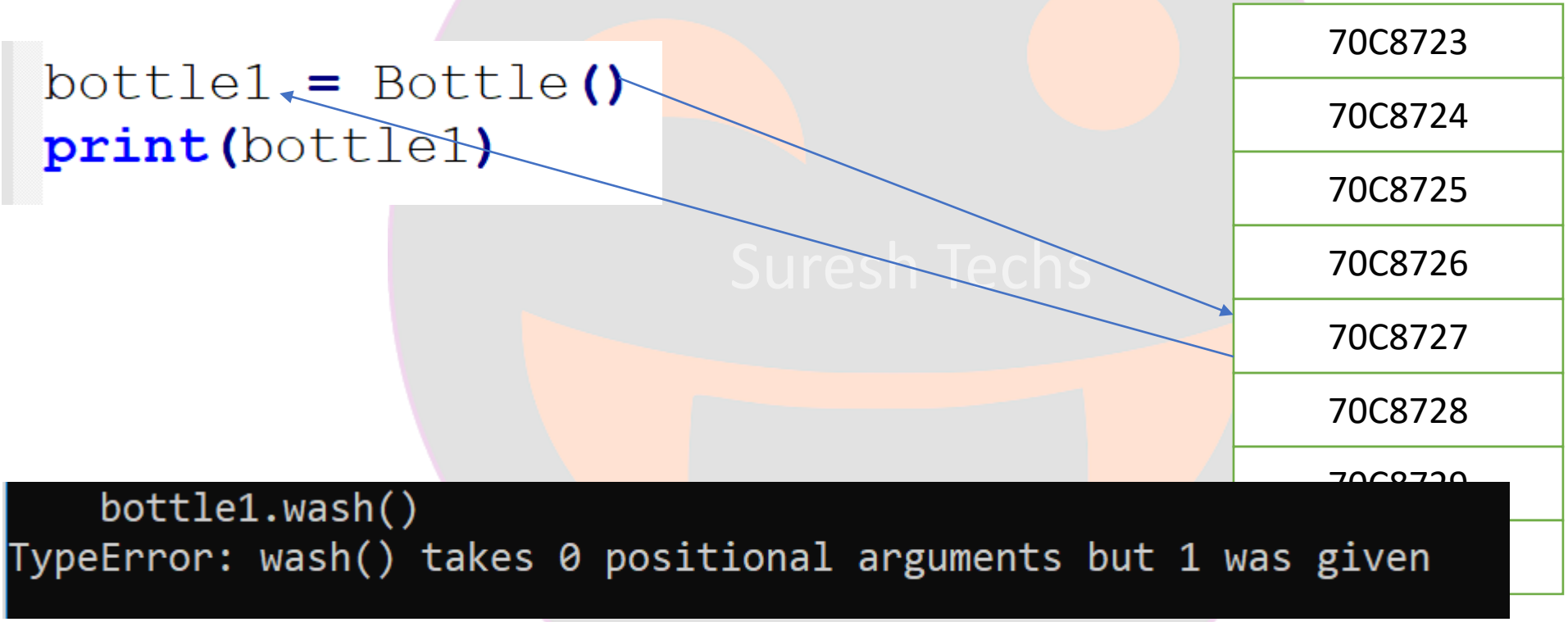


Creating Object?

- Can we create house without plan?
- Similarly, we can’t create objects without a class
- Let’s create **Bottle object**

Creating object / Instantiating class

```
bottle1 = Bottle()
print(bottle1)
```




```
    bottle1.wash()
TypeError: wash() takes 0 positional arguments but 1 was given
```

Calling methods

```
    bottle1.wash()
TypeError: wash() takes 0 positional arguments but 1 was given
```

Takes 0 positional arguments but 1 was given

Let’s see what is there in that argument

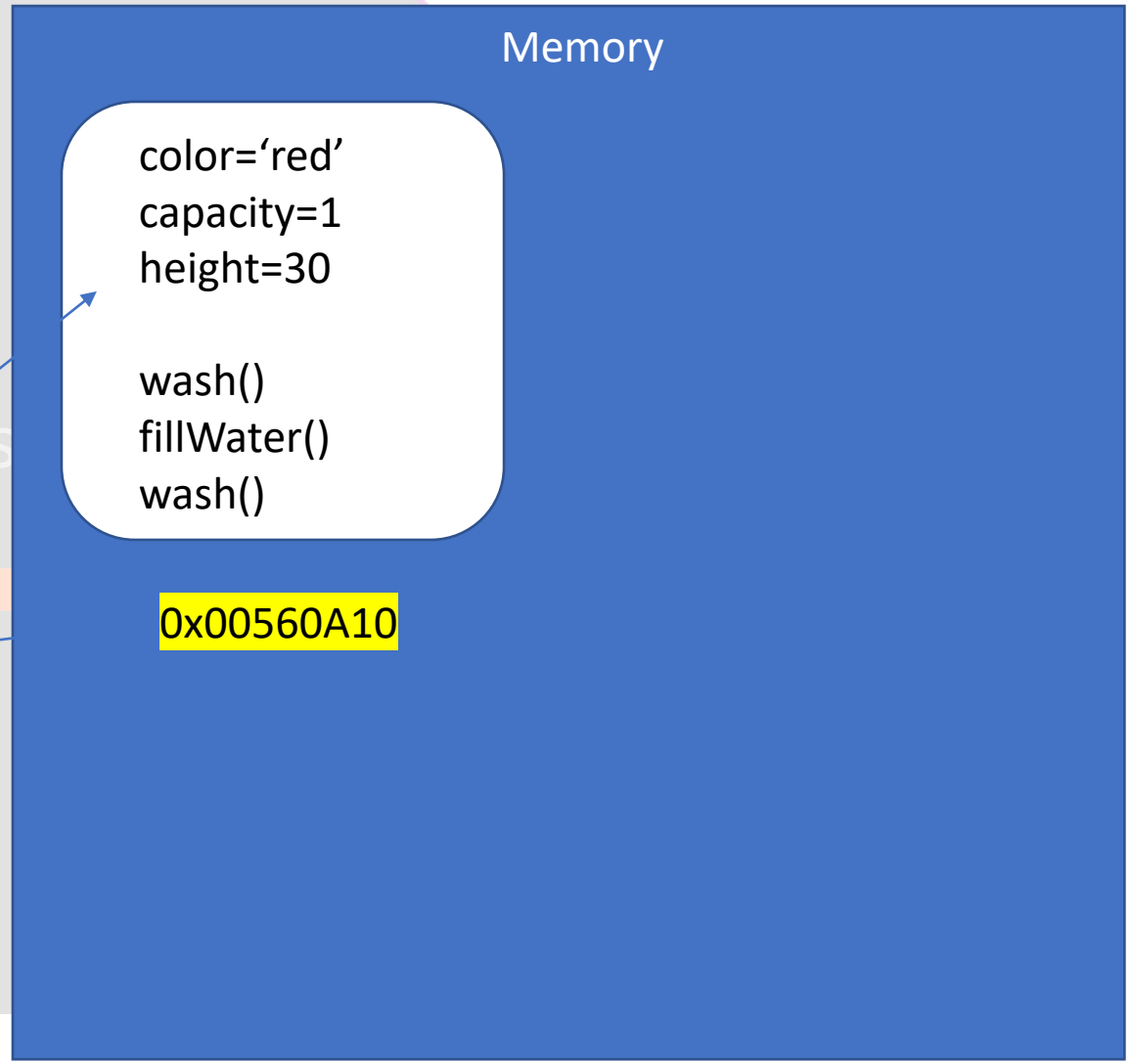
We basically name it as self but not mandatory

self?

- Class methods must and should have an extra first parameter in the method definition
- We don’t need to provide a value to this parameter, python will only provide a value for it
- Even though you have a method without arguments, still you have to provide one argument(self)

What happens when you create an object?

```
bottle1 = Bottle()
print(bottle1)
```



1. **Constructor** gets called and initializes all the attributes/properties with default values
2. All the properties/functions will be placed in a memory location
3. Returns that memory location

bottle1= 0x00560A10

bottle1.color
bottle1.wash()

CONSTRUCTOR

What is constructor?

- Constructor is also a method/function, but a special method given by python itself. (__init__(self))
- If you are coming from c++/java then self is nothing but **this** which refers to the current instance.
- Constructor will automatically be placed when you create a class but it is not be visible to you.


```
#constructor
def __init__(self):
    print(self)
```

Constructors are used to initialize object state

Constructor assigns default values to attributes

- But you can also initialize values in the constructor

```
#constructor
def __init__(self):
    color='red'
    capacity='2'
    print(self)
```

Am I right? What if I call bottle1.colr? Will it print newly assigned value? Think and tell me.

Types of constructors

- Default constructor
- Parameterized constructor
  - Constructor with parameters

class(static) and instance variables(attributes & methods)

- class variables:
  - shared by all the instances of the class
  - Class variable values are assigned in the class
  - ex: bottle company (Sirpa)
- instance variables:
  - can change in each instance of the class
  - instance variable values are assigned either in the constructor or a method with self

What happens if we create a new instance of the bottle?

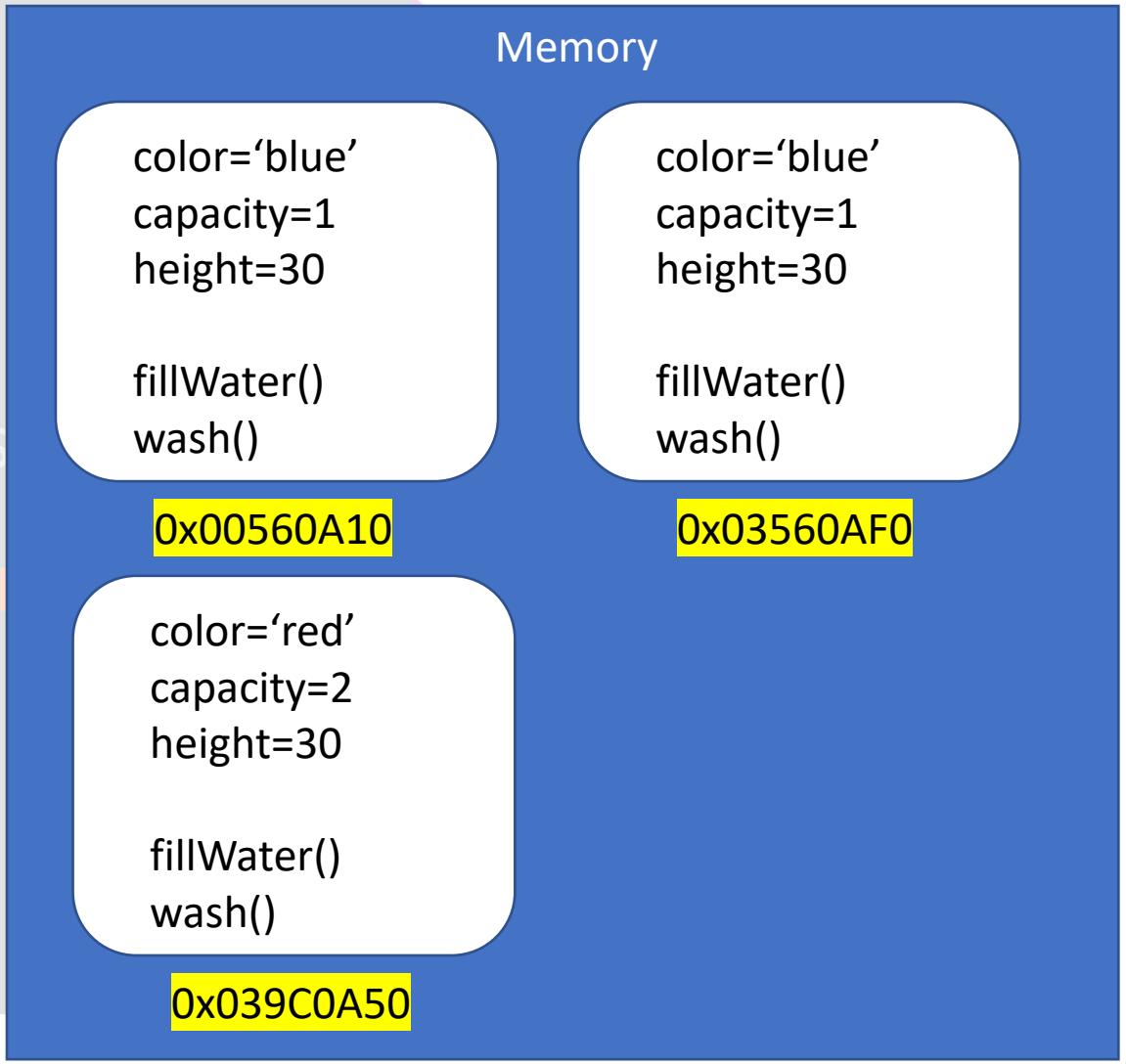
```
bottle2 = Bottle()
```

bottle2=0x03560AF0

```
def __init__(self,color,capacity):
    self.color=color
    self.capacity=capacity
```

```
bottle3 = Bottle('red',2)
```

0x039C0A50



Class variables are like static variables

- If you are coming from java background, you might know it better
- can access class variables using class
- Bottle.company

What we learned so far?

- Class
- Object
- self
- Constructor (__init__)
- Class variables and Instance variables

Every thing will have life

- Humans (God constructs a life for a reason(to help someone, to create something etc) and destructs life.
- Bottle – once water is completed, will be thrown away(constructor and destructor)
- Similarly in programming, just like how we have **constructors** there are also **destructors.**
- Which are used to manage memory.
- If you are from java/c/c++, it is called garbage collection
- **__del__()** is the method responsible for garbage collection in python.

DESTRUCTOR

Destructor

- When constructor gets called?
- Destructors are called when object gets destroyed, simple.
- Who will destroy the object?

Destroying object

- When all the references to the object have been deleted
  - Garbage collection
  - Reference to the object is deleted
    - End of the program,
    - del keyword
  - Let’s add destructor method and see if it calling at the end of the program?
  - **del** objectReference

Don’t worry

- Don’t worry much about the destroying, it’s a way to manage memory
- We use __del__() if we want to do something when an object reference is lost
- But not required for now.

INHERITANCE

Inheritance

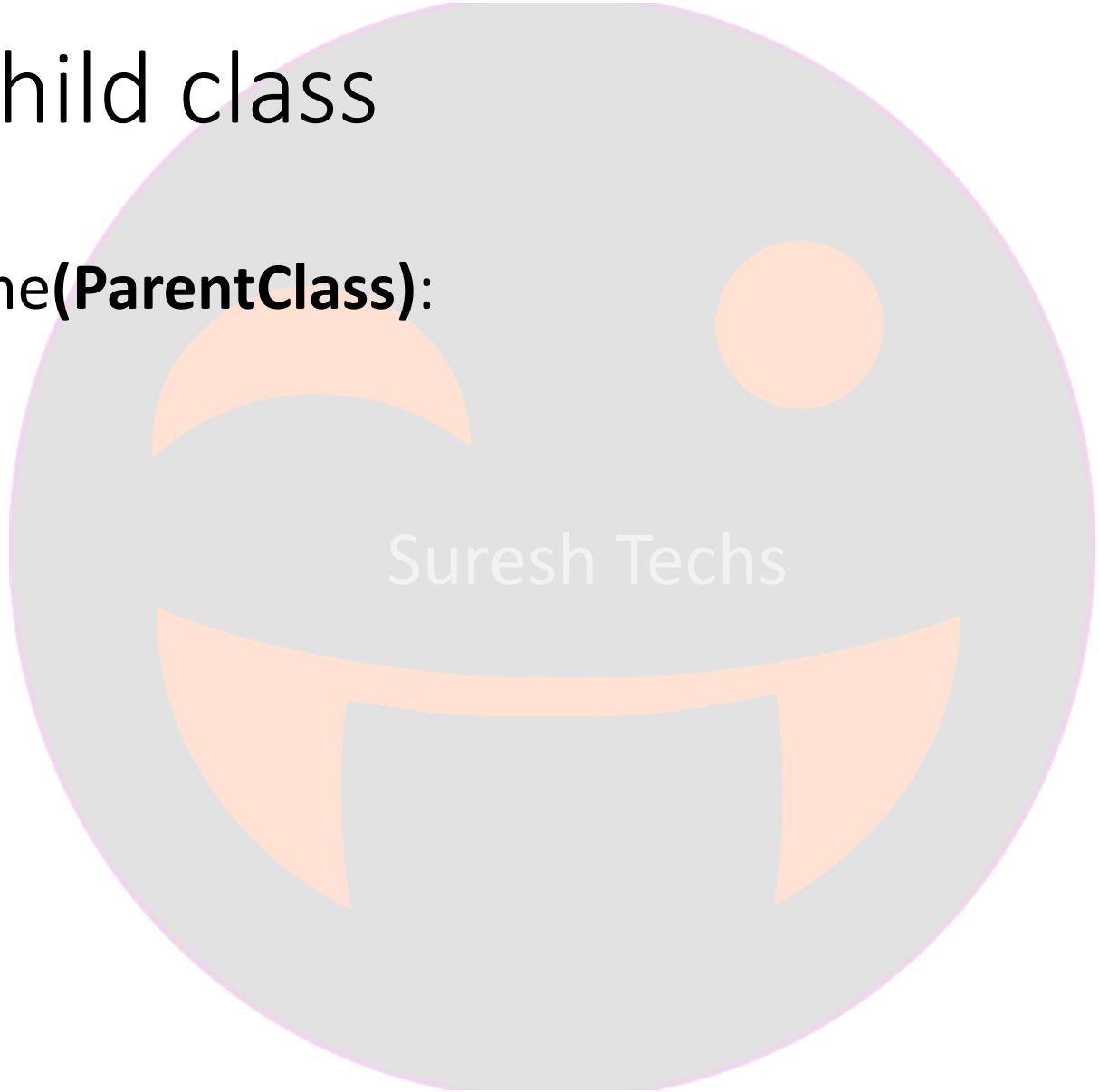
Children inherits characteristics from parents

Similarly in programming

- We can create child classes from parent classes
- Child classes will inherit all the properties and functions from the parent class
- Provides **reusability** of the code
- How to create child class?

Creating child class

- class className**(ParentClass)**:
  - #statements



How children born?

- Childs are of course created from parent
- Every child will have a parent
- Every parent will have a parent – of course

Who is the parent of all the classes?

- Who is the first/top parent?

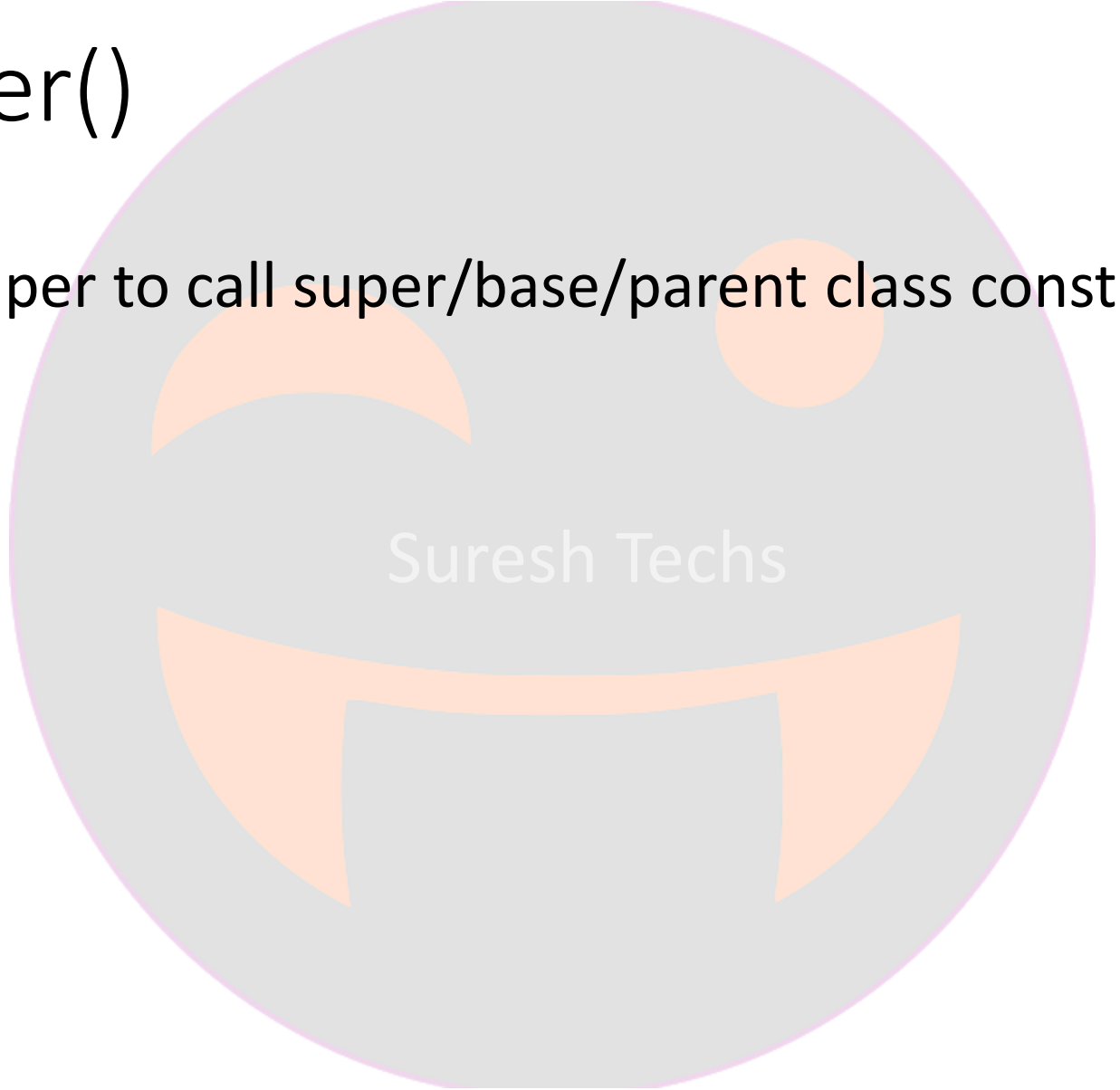
- GOD?
- In programming it is called **object** class
- **Every class that we create is derived from Object class by default**

What if we have constructor in child class?

```
class PlasticBottle(Bottle):
    def __init__(self):
        print('child constructor')
    def fillSoda():
        print('Filling soda')


#creating instance of child class
plasticBottle = PlasticBottle()
```

Using super()

- We can use super to call super/base/parent class construcotr



How to know the parent of any class?

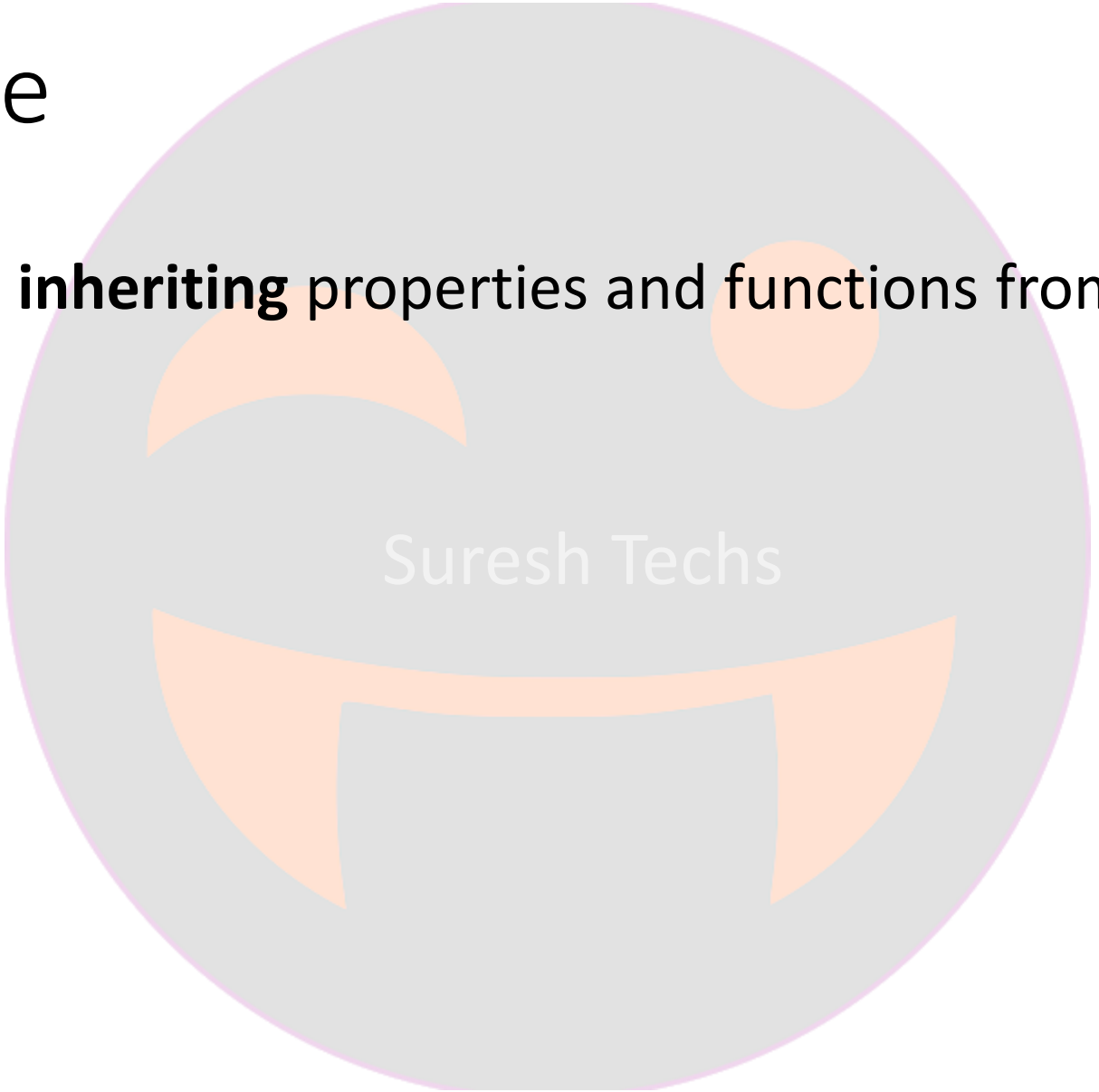
- class.__bases__
- How to know the parent of the class from an object?
- object.__class__.__bases__


```
print(Bottle.__bases__)
print(PlasticBottle.__bases__)
print(plasticBottle.__class__.__bases__)
```

```
print(object.__bases__)
print(object.__class__)
```

Inheritance

- So, the child is **inheriting** properties and functions from parent class

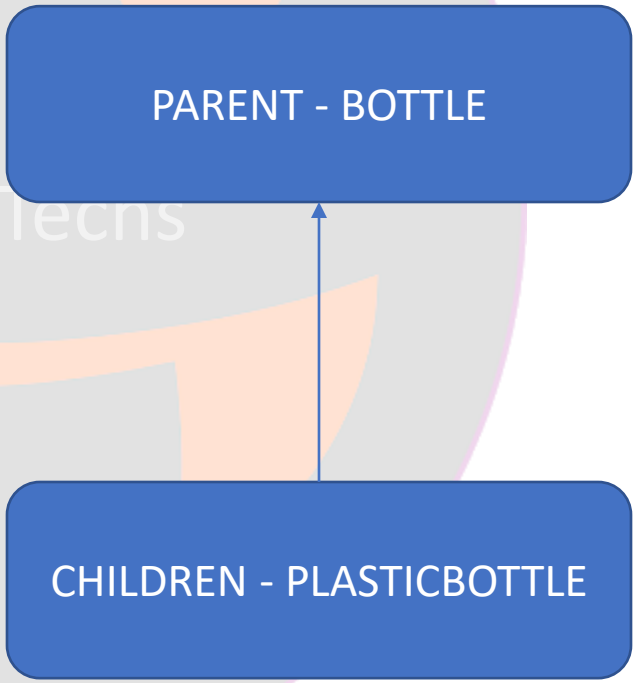


Types of inheritance

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

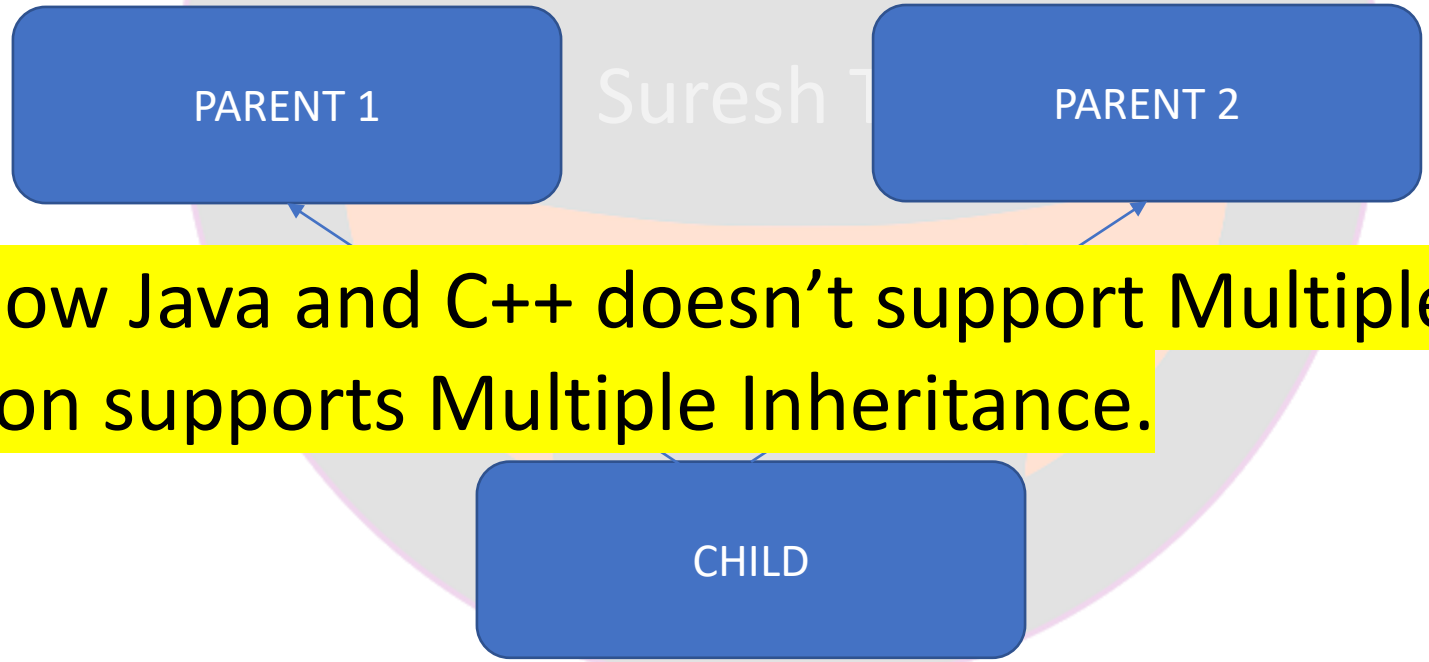
Single Inheritance

- When a child class inherits from only one parent class, it is called Single Inheritance



Multiple Inheritance

- When a child class inherits from multiple parent classes, it is called multiple inheritance



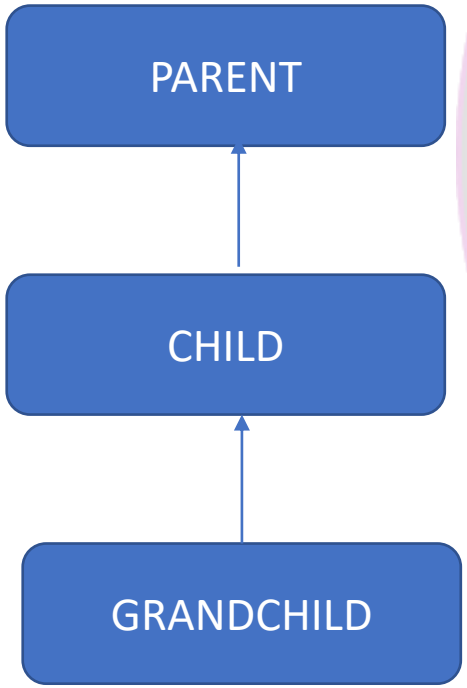
As we know Java and C++ doesn’t support Multiple Inheritance, But Python supports Multiple Inheritance.

Multiple Inheritance

```
class Parent1:
    def __init__(self):
        print('parent1')

    def singing(self):
        print('singing')


class Parent2:
    def __init__(self):
        print('parent2')

    def dancing(self):
        print('dancing')


class Child(Parent1,Parent2):
    def __init__(self):
        print('child')

child = Child()
child.singing()
child.dancing()
```

Multilevel Inheritance

- When there is a chain of inheritance, also known as child and grand child relationship



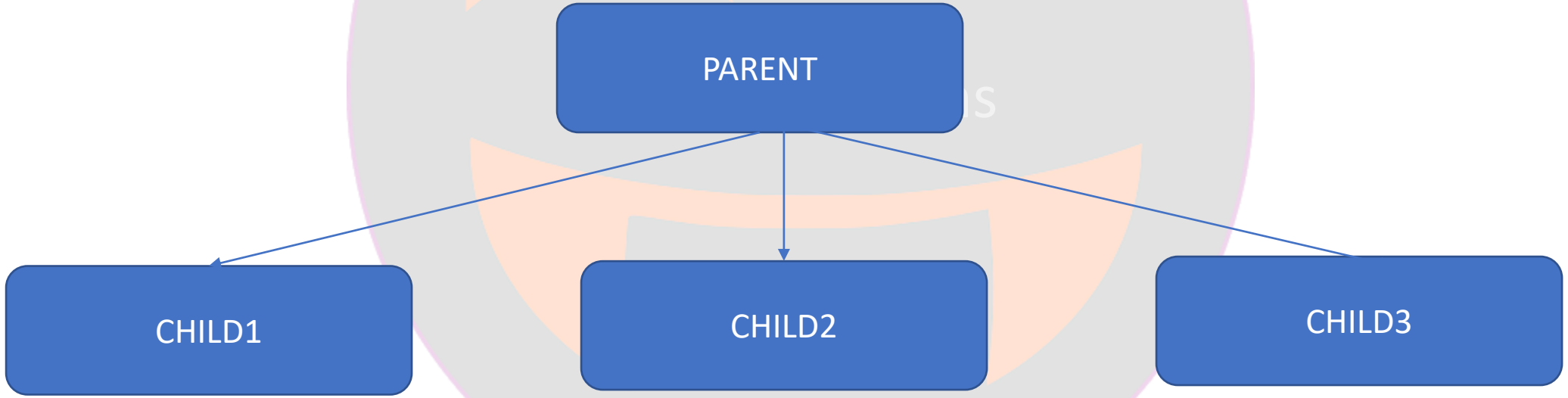

```
class Class1:
    def subscribe(self):
        print('subscribe')


class Class2(Class1):
    def join(self):
        print('joined')

class Class3(Class2):
    def details(self):
        print('class 3 details')
```

```
cl3 = Class3()
cl3.details()
cl3.join()
cl3.subscribe()
```

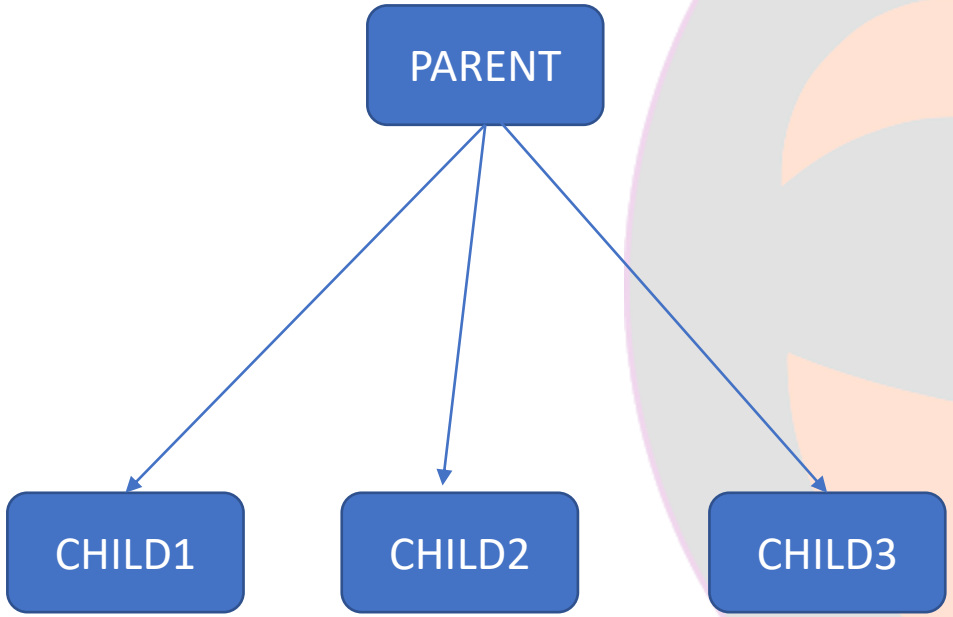
```
cl2 = Class2()
cl2.details()
```

Hierarchical Inheritance

- Multiple classes derived from a single parent/base class is called hierarchical inheritance



Hierarchical Inheritance




```
class Class1:
    def subscribe(self):
        print('subscribe')

class Class2(Class1):
    def join(self):
        print('joined')

class Class3(Class1):
    def details(self):
        print('class 3 details')

class Class4(Class1):
    def details(self):
        print('class 4 details')
```

Hybrid Inheritance

- It consists of multiple types(single, multiple, multilevel, hierarchical) of inheritance

ENCAPSULATION

Access Modifiers

echs

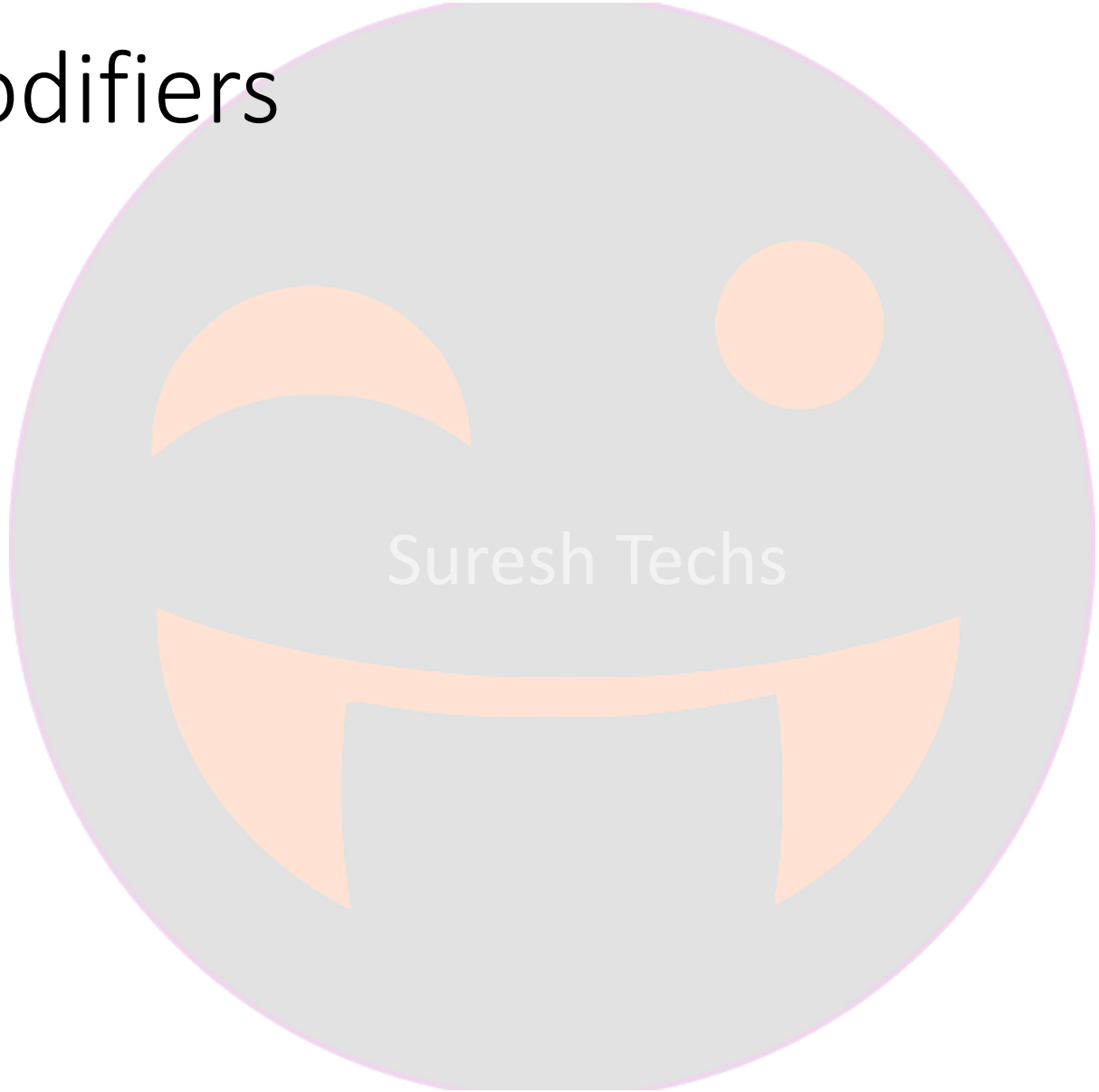
Encapsulation

Process of wrapping code and data together into a single unit

Prevents accidental modification of the data

Access modifiers

- Public
- Protected
- Private



Public

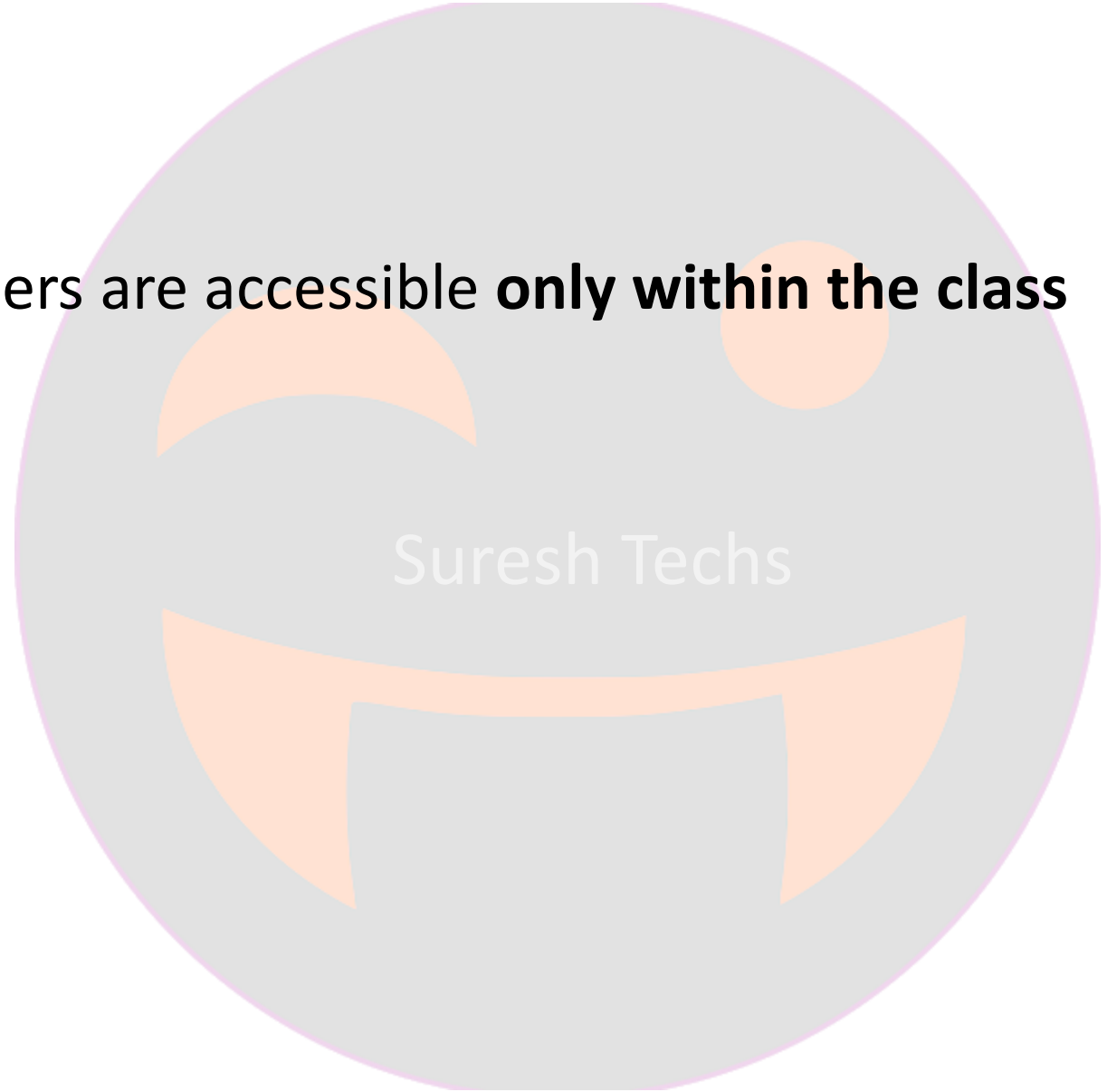
- Public members can be accessible from **any part of the program**

Protected

- Cannot be accessed outside the class but can be accessed from **within the class and its subclasses**

Private

- Private members are accessible **only within the class**



POLYMORPHISM

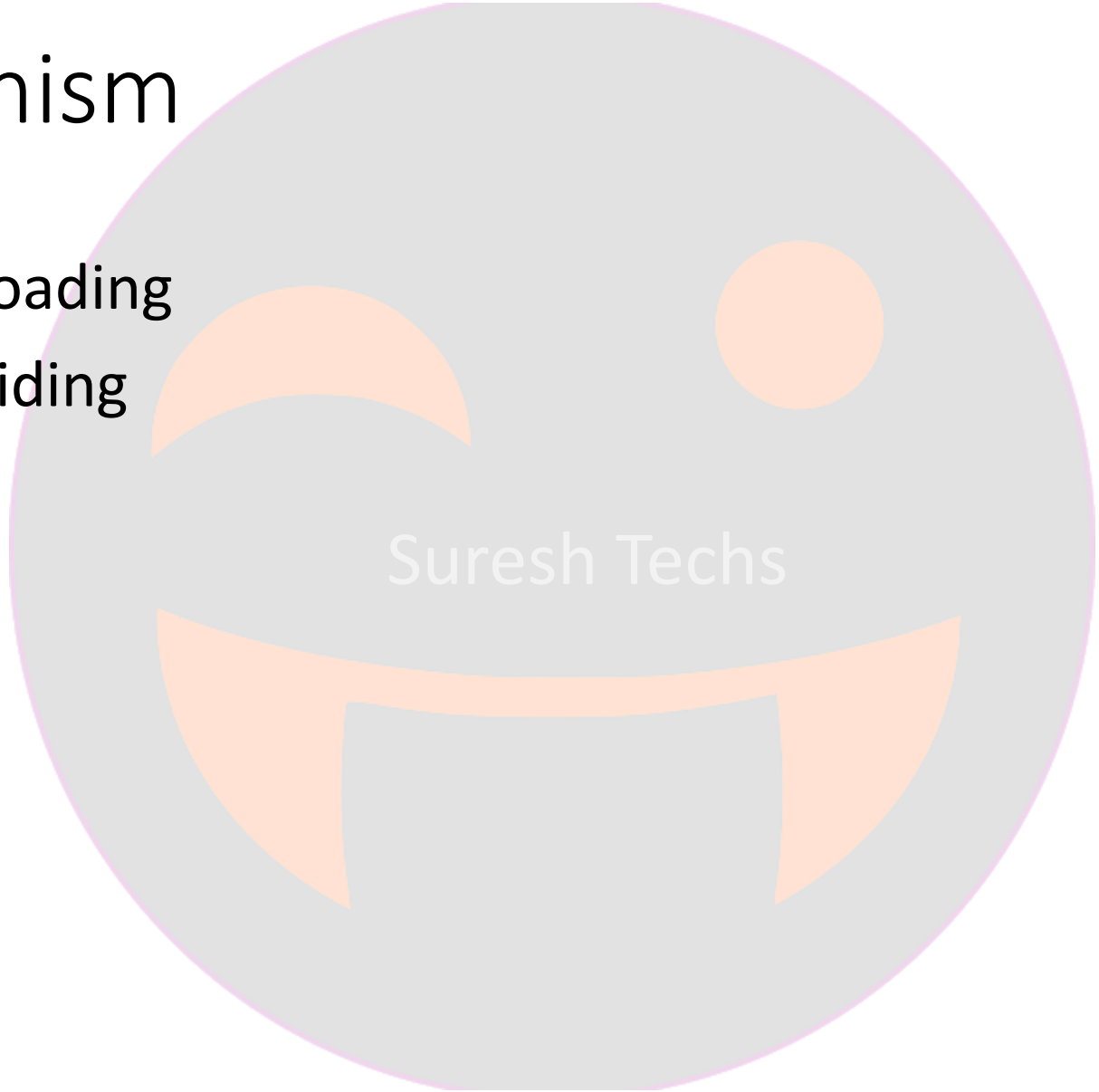
Polymorphism

- Poly + morphism
- many + forms

Same name but different forms/functionalities

Polymorphism

- Method overloading
- Method overriding



METHOD OVERLOADING

Method overloading

- Two methods having same name is called method overloading
- Python does not support method overloading
- We have work arounds to solve this issue

METHOD OVERRIDING

Method overriding

- If sub class/child class has the same method as declared in the parent class then it is called method overriding

PYTHON

చిన్న బ్రేక్

చిటికలో వచ్చేస్తా

#sureshtechs