DSA BOOTCAMP 2023

OOP

POLYMORPHISM & ABSTRACTION

HARSH DUCHE

I'm a Tech Enthusiast 🧑‍💻 and love to tinker around Neural Networks and Transformers

An open-source fan with little bit of python experience with jupyter notebooks.

As a technical writer, I share my insights and knowledge through Hashnode Blogs.,

Always up for a nerdy conversation about the latest tech trends or deep learning.

Let's chat! 🧑‍💻

@ducheharsh

@harsh-duche

https://codexperiences.hashnode.dev/

Author
HARSH DUCHE

Date
TODAY'S DATE

Try Pitch

Agenda

- OOP Recap
- Introduction to Polymorphism

  - Explanation, code snippet
- Introduction to Abstraction

  - Explanation & code snippet
- Conclusion

OOP: Recap

WHAT THIS SECTION IS ALL ABOUT

Polymorphism

" Polymorphism is the ability of an object to take different forms depending on the context. "

POETIC TAKE

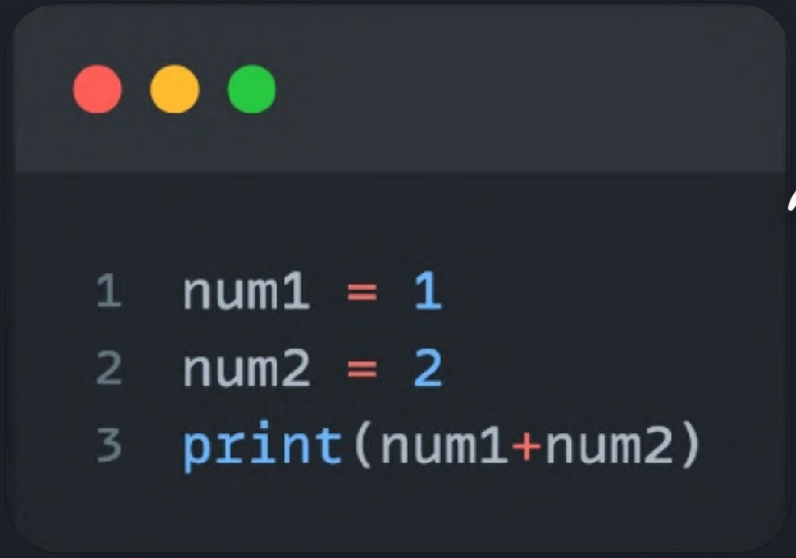
✍️

"A boy starts LOVE with the word **FRIENDSHIP**, but girl ends LOVE with the same word **FRIENDSHIP**. Word is the same but attitude is different. This beautiful concept of OOP is nothing but POLYMORPHISM..."

Try Pitch

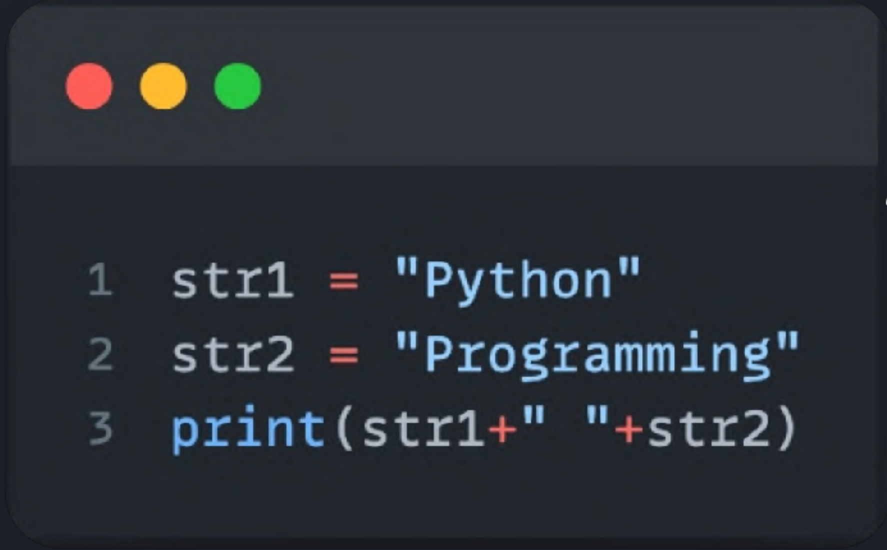
Simple occurrences of polymorphism in Python.

OPERATOR OVERLOADING



```python
num1 = 1
num2 = 2
print(num1+num2)
```

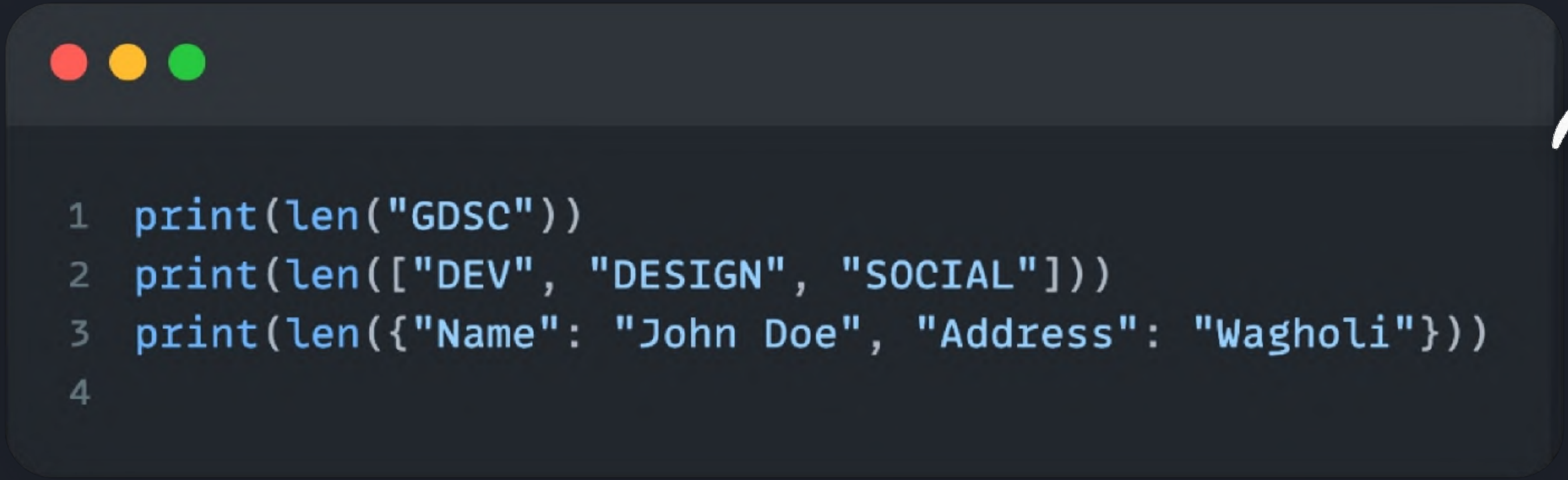
```
3
```



```python
str1 = "Python"
str2 = "Programming"
print(str1+" "+str2)
```

```
Python Programming
```

FUNCTION OVERLOADING



```python
print(len("GDSC"))
print(len(["DEV", "DESIGN", "SOCIAL"]))
print(len({"Name": "John Doe", "Address": "Wagholi"}))

```

```
4
3
2
```

Try Pitch

Code snippets

POLYMORPHISM IN CLASS METHODS

We can use the concept of polymorphism while creating class methods as Python allows different classes to have methods with the same name.
We can then later generalize calling these methods by disregarding the object we are working with.

OUTPUT

```
I am Tech Lead Nilesh Telang from Computer
We can code anything and we are the best
I am Design Lead Anshusingh Rajput from ENTC
We can design cool digital content
```

```python
class dev_team:
    def __init__(self, name, dept):
        self.name = name
        self.dept = dept

    def introduce(self):
        print(f"I am Tech Lead {self.name} from {self.dept}")

    def skills(self):
        print("We can code anything and we are the best")

class design_team:
    def __init__(self, name, dept):
        self.name = name
        self.dept = dept

    def introduce(self):
        print(f"I am Design Lead {self.name} from {self.dept}")

    def skills(self):
        print("We can design cool digital content")


dev1 = dev_team("Nilesh Telang", "Computer")
designer1 = design_team("Anshusingh Rajput", "ENTC")

dev1.introduce()
dev1.skills()

designer1.introduce()
designer1.skills()
```

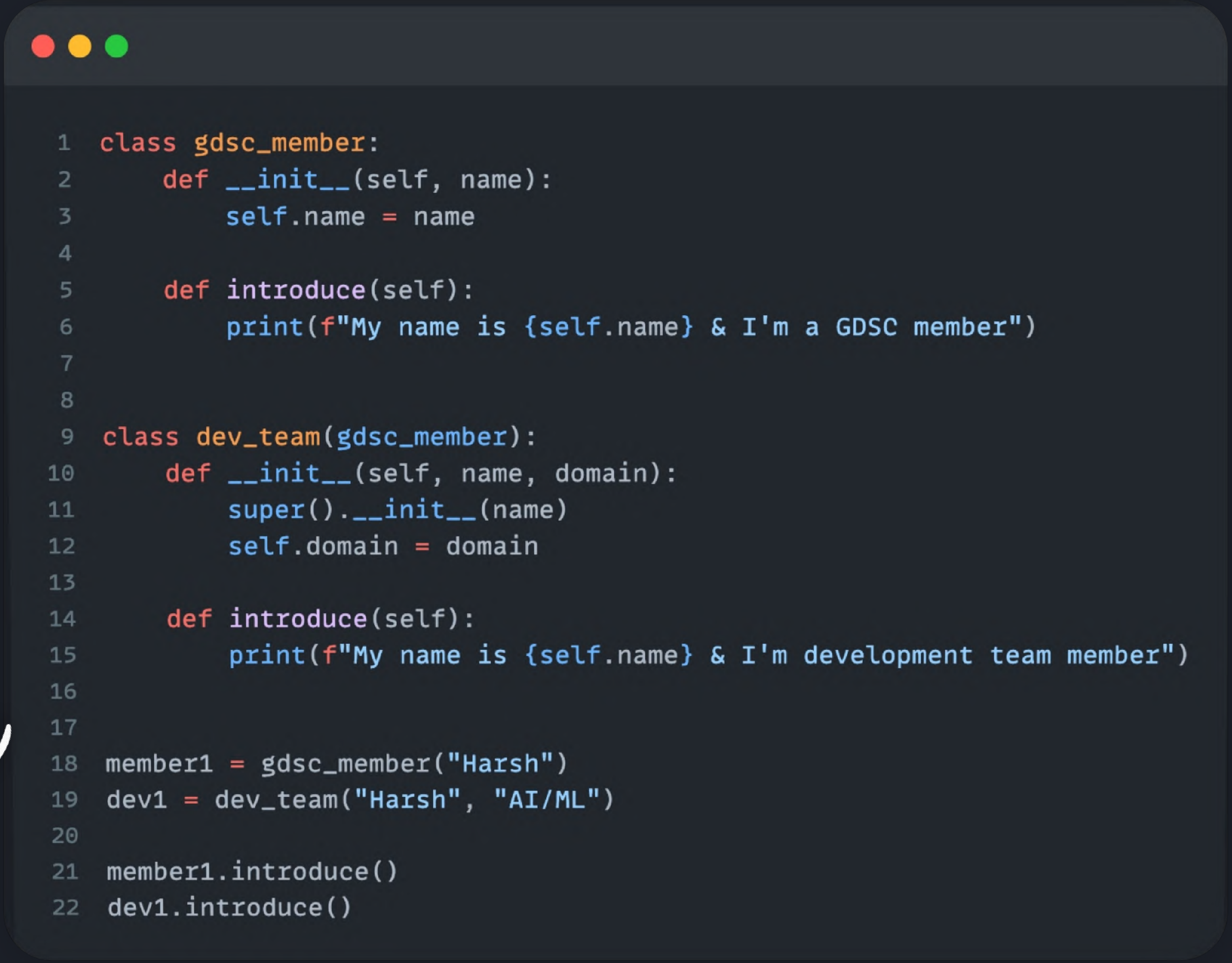
Code snippets

METHOD OVERRIDING

We can redefine certain methods and attributes specifically to fit the child class, which is known as **Method Overriding**.
Polymorphism allows us to access these overridden methods and attributes that have the same name as the parent class.

OUTPUT

```
My name is Harsh & I'm a
GDSC member
My name is Harsh & I'm
development team member
```

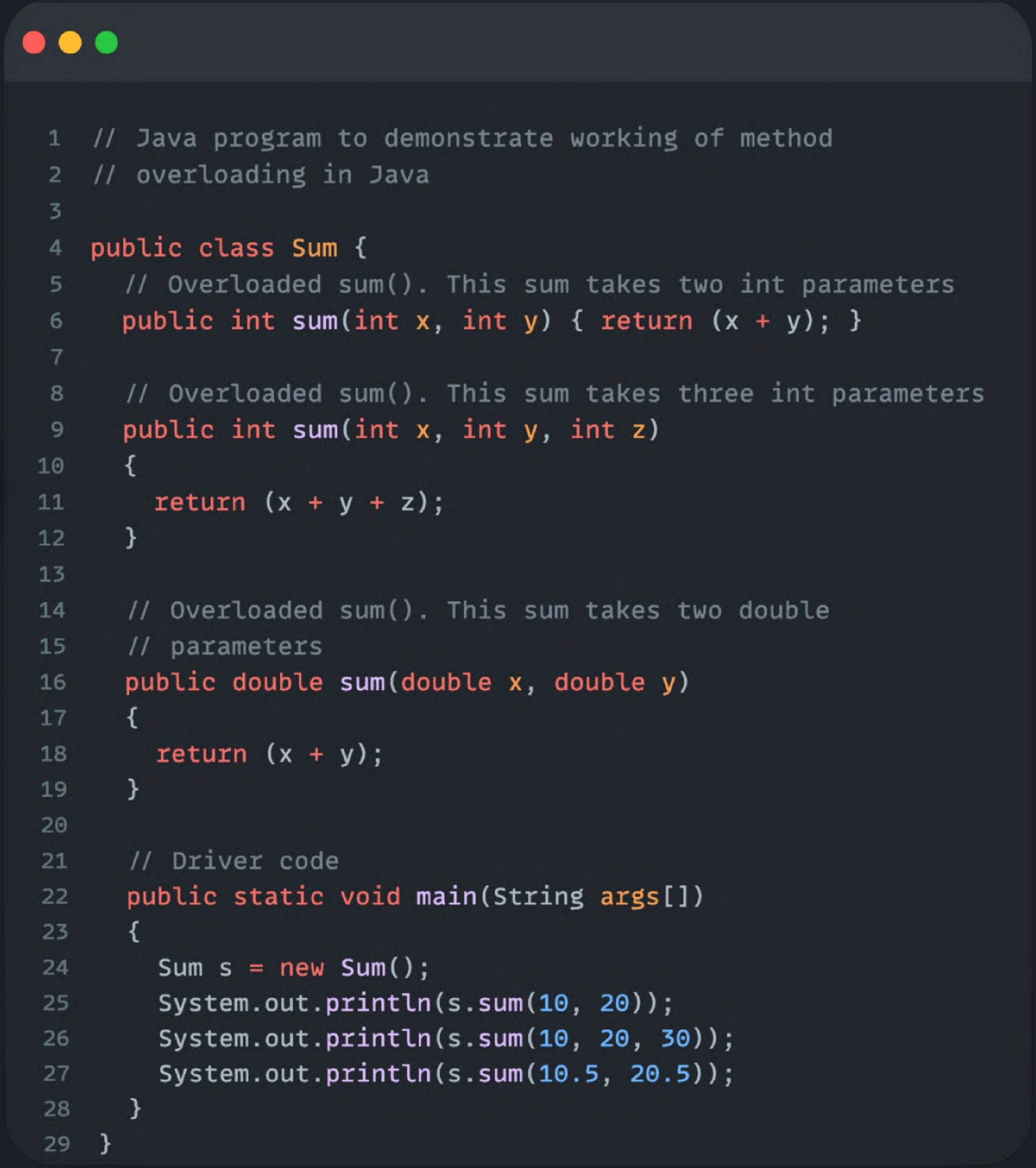


```python
class gdsc_member:
    def __init__(self, name):
        self.name = name

    def introduce(self):
        print(f"My name is {self.name} & I'm a GDSC member")


class dev_team(gdsc_member):
    def __init__(self, name, domain):
        super().__init__(name)
        self.domain = domain

    def introduce(self):
        print(f"My name is {self.name} & I'm development team member")


member1 = gdsc_member("Harsh")
dev1 = dev_team("Harsh", "AI/ML")

member1.introduce()
dev1.introduce()
```

NOTE:

Method Overloading, a way to create multiple methods with the same name but different arguments, is not possible in Python.

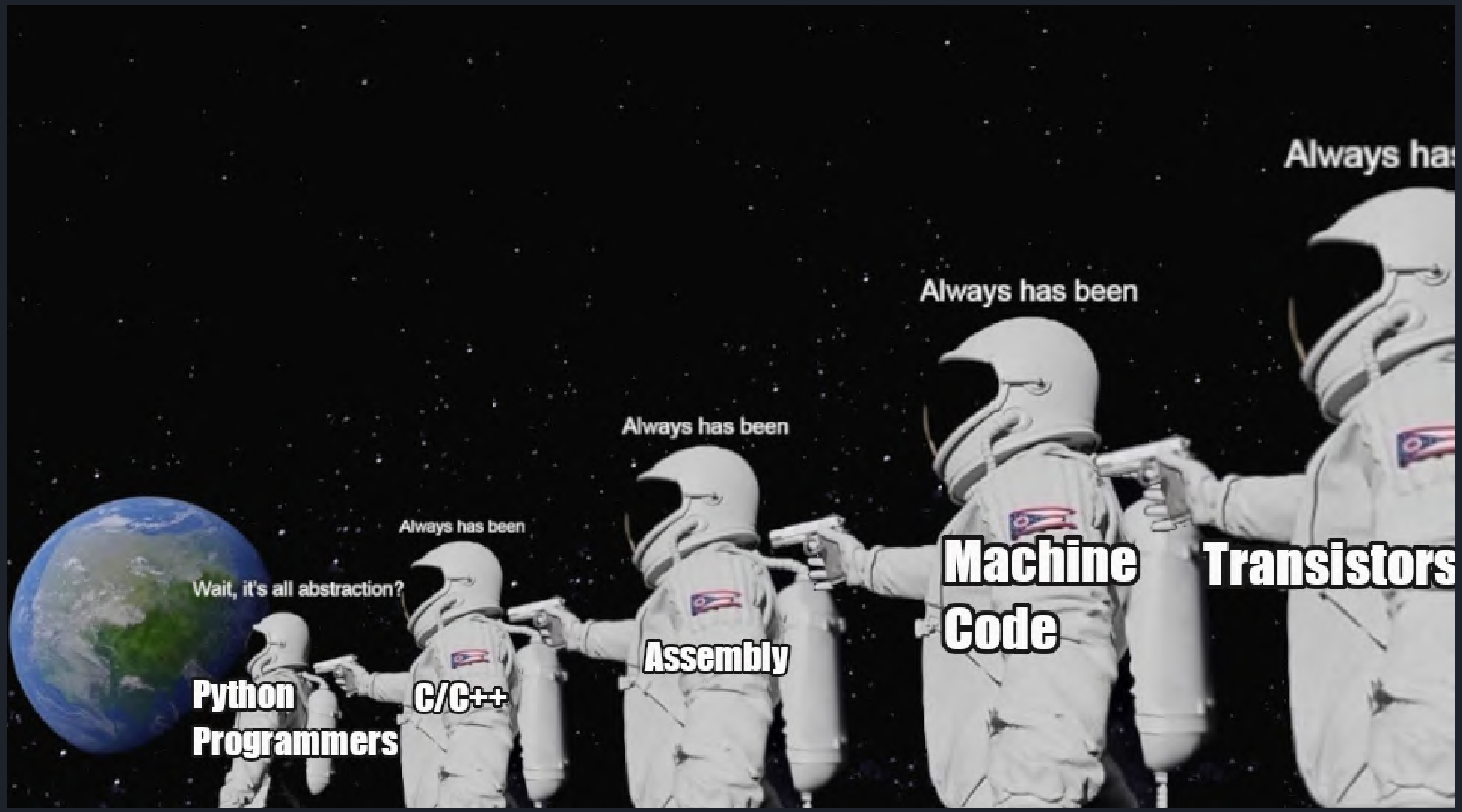
Here's the JAVA Implementation for the same (i.e, Method Overloading)



```java
// Java program to demonstrate working of method
// overloading in Java

public class Sum {
  // Overloaded sum(). This sum takes two int parameters
  public int sum(int x, int y) { return (x + y); }

  // Overloaded sum(). This sum takes three int parameters
  public int sum(int x, int y, int z)
  {
    return (x + y + z);
  }

  // Overloaded sum(). This sum takes two double
  // parameters
  public double sum(double x, double y)
  {
    return (x + y);
  }

  // Driver code
  public static void main(String args[])
  {
    Sum s = new Sum();
    System.out.println(s.sum(10, 20));
    System.out.println(s.sum(10, 20, 30));
    System.out.println(s.sum(10.5, 20.5));
  }
}
```

ABSTRACTION

Abstraction is used to hide the internal functionality of the function from the users. The users only interact with the basic implementation of the function, but inner working is hidden. User is familiar with that **"what function does"** but they don't know **"how it does."**



HEY, WELCOME TO MY ISLAND. WHAT'S YOUR STORY?
MY STORY IS INCREDIBLE BUT I'M VERY BUSY RIGHT NOW.
SO, TO YOU I WILL REMAIN A MYSTERY.
ARE YOU A CRAB WHO ROWS A BOAT AROUND?
THAT'S THE BASIC PREMISE, BUT ANY DETAILS WILL REMAIN A MYSTERY.
poorlydrawnlines

ABSTRACTION

DEFINATIONS:

- **Abstract class** = a class which contains one or more abstract methods.
- **Abstract method** = a method that has a declaration but does not have an implementation.

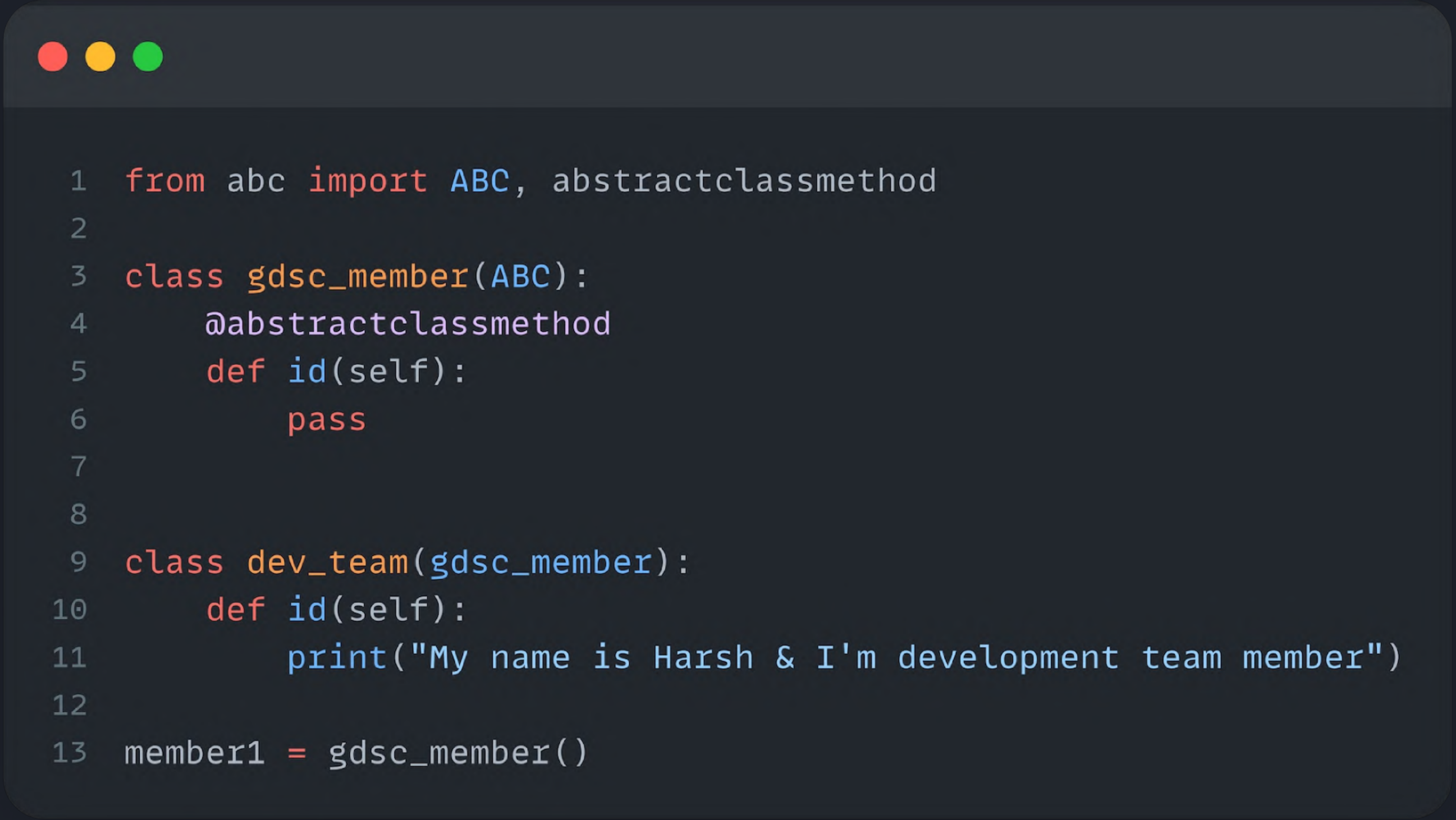
USAGE:

- Prevents a user from creating an object of that class
- Compels a user to override abstract methods in a child class

Code snippets

PREVENTS A USER FROM CREATING AN OBJECT OF THAT CLASS

- This gives an ERROR
- Example of information hiding
- Provides an control over access



```python
from abc import ABC, abstractclassmethod

class gdsc_member(ABC):
    @abstractclassmethod
    def id(self):
        pass


class dev_team(gdsc_member):
    def id(self):
        print("My name is Harsh & I'm development team member")

member1 = gdsc_member()
```

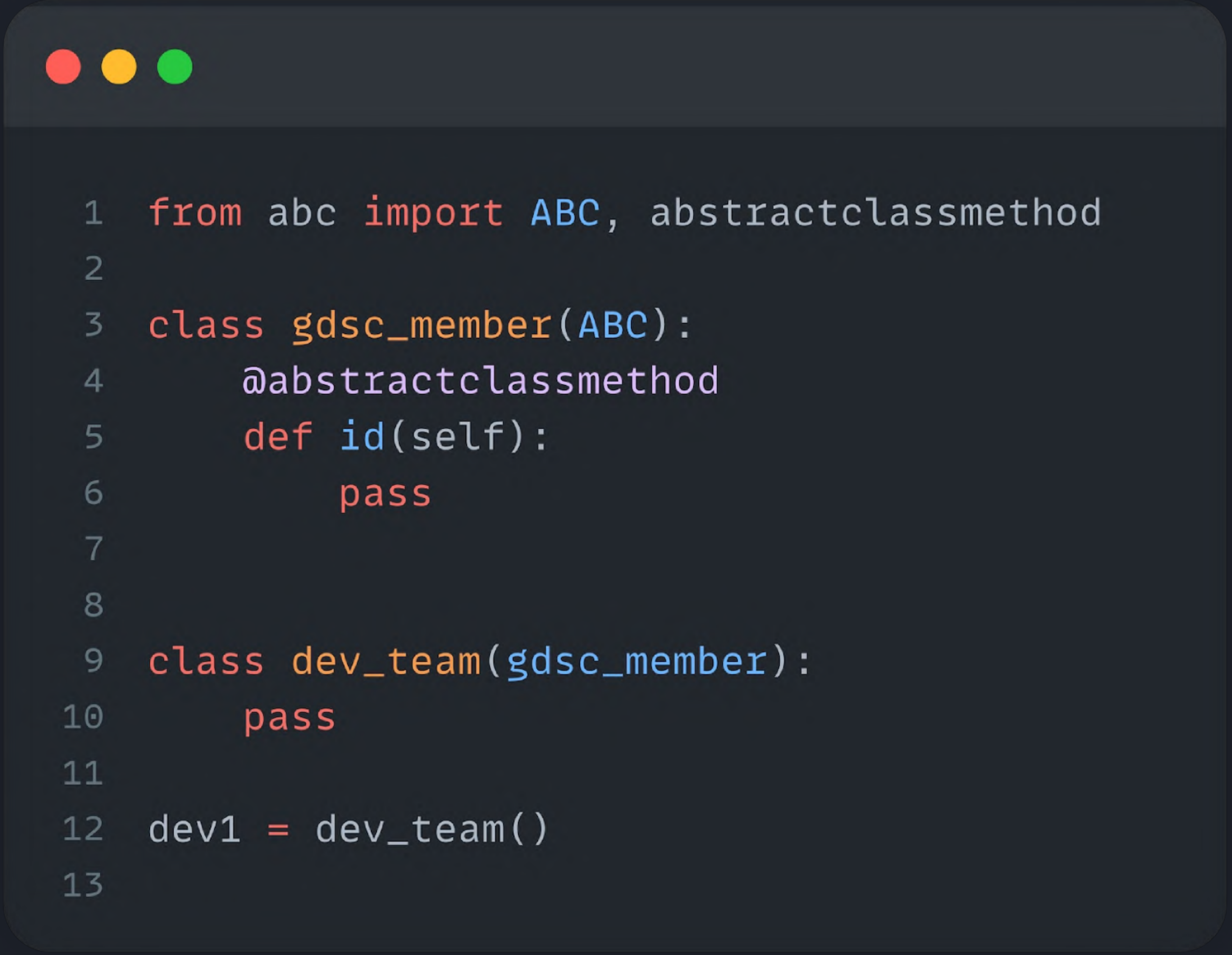
OUTPUT

```
  member1 = gdsc_member()
            ^^^^^^^^^^^^^
TypeError: Can't instantiate abstract class gdsc_member with abstract method id
```

Code snippets

COMPELS A USER TO OVERRIDE ABSTRACT METHODS IN A CHILD CLASS

- This gives an ERROR
- Prevents creation of improper methods
- Follows OOP methodology



```python
from abc import ABC, abstractclassmethod

class gdsc_member(ABC):
    @abstractclassmethod
    def id(self):
        pass


class dev_team(gdsc_member):
    pass

dev1 = dev_team()

```

OUTPUT

```
dev1 = dev_team()
^^^^^^^^^^
TypeError: Can't instantiate abstract class dev_team with abstract method id
```

Code snippets

COMPELS A USER TO OVERRIDE ABSTRACT METHODS IN A CHILD CLASS

Correct Impelementation

```python
from abc import ABC, abstractclassmethod

class gdsc_member(ABC):
    @abstractclassmethod
    def id(self):
        pass


class dev_team(gdsc_member):
    def id(self):
        print("My name is Harsh & I'm development team member")

dev1 = dev_team()

dev1.id()
```

✓

OUTPUT

```
My name is Harsh & I'm development team member
```

CONCLUSION

OOP RECAP

Just a refresher for the further context

POLYMORPHISM

Introduction with code snippets

ABSTRACTION

Introduction with code snippets

Conclusion

Concluding rn.......
QnA

THANK YOU