Harsh Dixit

```
/* OOPS (OBJECT ORIENTED PROGRAMMING)

    WHAT? - OOPS is a programming paradigm or technique in which things revolve
    around object.

    WHY? - OOPS related programming with real-life applications.Increaes readability,
    reusability,manageabilty.


OBJECT - Object is an entity which state/properties and behaviour/functions.


CLASS - Class is a user-defined or custom data type.

    Class is the blue print of the object and object is the instance of class.


*/

// SIZE OF EMPTY class

#include <iostream>
using namespace std;
class animal
{
    //empty
};
int main()
{
    cout << "Size of empty class: " << sizeof(animal) << "byte";
       return 0;
}



/* ACCESS MODIFIERS - They define the scope of class attributes.

    public - By making class attributes public, we can access them inside
        and outside the class.

    private - By making the class attributes private , we can access inside
        the class only.
*/

// Class comsists of ->

// class
// {
//     state/properties
//     int a;
//     string str;

//     Behaviour/functions
//     void func1(){}
//     void fum2{}
// };
```

```
//---------------------------------------------------------------------

// C++ program to demonstrate accessing of data members

#include <bits/stdc++.h>
using namespace std;
class Geeks {
    // Access specifier
public:
    // Data  Members
    string geekname;
    // Member Functions()
    void printname() { cout << "Geekname is:" << geekname; }
};

int main()
{
    // Declare an object of class geeks
    Geeks obj1;
    // accessing data member
    obj1.geekname = "Abhi";
    // accessing member function
    obj1.printname();
    return 0;
}

//---------------------------------------------------------------------

#include<iostream>
using namespace std;

class animal
{
    //state
    public:
    int age;
    int weight;

    //Behaviour
    void eat()
    {
        cout << "Eatng" << endl;
    }
    void sleep()
    {
        cout << "Sleeping" << endl;
    }
};

int main()
{
    // Object Creation
    animal pradeep;

    // static
    pradeep.age = 12;
    pradeep.weight = 43;
    cout << pradeep.age << endl;
    cout << pradeep.weight << endl;

    pradeep.eat();
    pradeep.sleep();

}

//--------------------------------------------------------------------

// C++ program to demonstrate function
// declaration outside class

#include <bits/stdc++.h>
using namespace std;
class Geeks
{
    public:
    string geekname;
    int id;

    // printname is not defined inside class definition
    void printname();

    // printid is defined inside class definition
    void printid()
    {
        cout <<"Geek id is: "<<id;
    }
};

// Definition of printname using scope resolution operator ::
void Geeks::printname()
{
    cout <<"Geekname is: "<<geekname;
}
int main() {

    Geeks obj1;
    obj1.geekname = "xyz";
    obj1.id=15;

    // call printname()
    obj1.printname();
    cout << endl;

    // call printid()
    obj1.printid();
    return 0;
}


//---------------------------------------------------------------------------

// GETTER AND SETTER

/* If we want to access private members outside class, we use getters and
    setters for that.
    Getter and setter are the functions. Getter fetchea the property and
    setter set the value of property.
*/

#include<iostream>
using namespace std;

class getset
{
    private:
    int value;

    public:
    int getValue()
    {
        return value;
    }
    int setValue(int v)
    {
        value = v;
    }
};
int main()
{
    // object
    getset num;
    num.setValue(20);
    cout << num.getValue() << endl;
}

//----------------------------------------------------------------------

// DYNAMIC OBJECT CREATION -->

#include<iostream>
using namespace std;

class animal
{
    public:
    int age;
    void eat()
    {
        cout << "Eating" << endl;
    }
};

int main()
{
    // creating obj dynamically

    animal *dog = new animal;

    // accessing obj using (.) operator

    (*dog).age = 12;
    cout << (*dog).age << endl;
    (*dog).eat();

    // alternative using arrow

    dog -> age = 12;
    cout << dog -> age << endl;
    dog -> eat();
}


//------------------------------------------------------------------------

// This keyword - this is a pointer to the current object.

#include<iostream>
using namespace std;

class animal
{
    private:
    int weight;

    public:
    int getWeight()
    {
        return weight;
    }
    int setWeight(int weight)
    {
        this -> weight = weight;
        // or we can write like this
        // (*this).weight = weight;
    }
};

int main()
{
    animal a;
    a.getWeight();
    a.setWeight(50);

    cout << a.getWeight() << endl;
// }

// NOTE: jb bhi data members ko access krenge class mai, use this keyword.
//       it is considered as good practice.


//---------------------------------------------------------------------------


// CONSTRUCTORS - constructors is called whenever an object is created.

// 1] It initialises object.
// 2] Same name as class name.
// 3] Has no return type.

// As constructor is called by default whenever an object is created but when
// we make constructor by our own,then this constructor overides the default
// one.


// 1] DEFAULT CONSTRUCTOR

#include<iostream>
using namespace std;

class animal
{
    public:
    string type;
    int age;
    int weight;

    //DEFAULT CONSTRUCTOR
    animal()
    {
        this -> type = " ";
        this -> age = 0;
        this -> weight = 0;

        cout << "Constructor called " << endl;
    }
};

int main()
{
    animal a;
}

//-------------------------------------------------------------------------

// PARAMETERIZED CONSTRUCTOR

#include<iostream>
using namespace std;

class animal
{
    public:
    string type;
    int age;
    int weight;

    //PARAMETERIZED CONSTRUCTOR
    //Single parameter
    animal(int age)
    {
        this -> age = 0;
        cout << "PARAMETERIZED Constructor 1 called  " << endl;
    }

    //Two parameter
    animal(int age,int weight)
    {
        this -> age = age;
        this -> weight = weight;
        cout << "PARAMETERIZED Constructor 2 called  " << endl;
    }
};

int main()
{
    animal a(10);
    animal b(10,20);
}


// Note: obj creation mai jitne parameters pass kiye honge uske according
// Constructor call jayegi.


//---------------------------------------------------------------------------

// COPY Constructor

// If we create copy Constructorthen we have to make default Constructor
// otherwise error shows hoga

#include<iostream>
using namespace std;

class animal
{
    public:
    int age;
    int weight;

    //default Constructor
    animal()
    {

    }

    //copy Constructor
    animal(animal &obj)
    {
        this -> age = obj.age;
        this -> weight = obj.weight;
        cout << "I am inside copy Constructor " << endl;
    }
};

int main()
{
    animal a;
    animal b =a;
    animal c(b);
    // line 397 & 398 are 2 methods to copy the objects
    animal *d = new animal(c);
}

/* Note: if obj pass by value error aayega
         Pass by value krne se repeatedly copy bnegi.That's why
         copy Constructor again & again call hoga and infinite loop me
         fss jayega.To prevent pass by refrence kro obj ko inside copy
         Constructor.
*/



//----------------------------------------------------------------------------------

// DESTRUCTOR

/*
Destructor is an instance member function that is invoked automatically
whenever an object is going to be destroyed. Meaning, a destructor is the
last function that is going to be called before an object is destroyed.

1] A destructor is also a special member function like a constructor. Destructor
destroys the class objects created by the constructor.

2] Destructor has the same name as their class name preceded by a tilde (~) symbol.

3] It is not possible to define more than one destructor.

4] The destructor is only one way to destroy the object created by the constructor.
Hence destructor can-not be overloaded.

5] Destructor neither requires any argument nor returns any value.

6] It is automatically called when an object goes out of scope.

7] Destructor release memory space occupied by the objects created by the constructor.

8] In destructor, objects are destroyed in the reverse of an object creation.

*/

// C++ program to demonstrate the execution of constructor
// and destructor

#include <iostream>
using namespace std;

class Test {
public:
    // User-Defined Constructor
    Test() { cout << "\n Constructor executed"; }

    // User-Defined Destructor
    ~Test() { cout << "\nDestructor executed"; }
};
main()
{
    Test t;

    return 0;
}
```

```
//-----------------------------------------------------------------------------------


// GLOBAL VARIABLES ->

// 1] Written outside of function

// 2] Accessible to all functions.Functions mai global variable ki copy nhi bnti,
// actual memory location pr kaam ho rha hota hai.


// LOCAL VARIABLES ->

// 1] Written inside of funtion.

// 2] Accessible inside the function scope only.




#include<iostream>
using namespace std;

int x = 10;

int main()
{
    x += 2;

    int x = 10; // loacal variable
    {
        int x = 200; // loacal variable
        cout << x << endl;
    }
    cout << x << endl;
    cout << ::x << endl;
}


// Output

// 200
// 10
// 12

//==================================================================
```