Examples of Inheritance

```
include <iostream>
using namespace std;

class rectangleType // base class
{
protected:
    double length;
    double width;
public:
    rectangleType() {length = 0;
    width = 0;}
    rectangleType( double  L, double w)
{setDimension( L , w); }
    void setDimension ( double L, double w)
{    if ( L >= 0 )  length = L;
    else             length = 0;
    if ( w >= 0 )width= w;
    else    width = 0;
}
    double getLength()
{    return length;}
    double getWidth()
 {    return width;}
    double area()
{return length * width;}

    double perimeter()
 {    return 2 * ( length + width );}
    void print(){
    cout<<"Length = "<< length << " ; Width = " << width;
}

};


class boxType: public rectangleType {
private:
    double height;
public:
boxType() {    height = 0 ;}
```

```cpp
boxType( double L, double w, double h) {setDimension( L, w, h);   }

~boxType(){}

    void setDimension ( double L, double w, double h)
{    rectangleType::setDimension( L , w );
    if ( h >= 0)
        height = h;
    else
        height = 0;}

double getHeight() {    return height; }

double area() {    return 2 * ( length * width + length * height + width *
height );  }

double volume() {return rectangleType::area() * height;  }

void print() {    rectangleType::print();
    cout << " ; Height = " << height;}
};

int main()
{
rectangleType myRectangle1;    rectangleType myRectangle2(8, 6);
boxType myBox1;    boxType myBox2(10, 7, 3);

    cout << "\n myRectangle1: "<<endl;
myRectangle1.print();
cout << " Area of myRectangle1: " << myRectangle1.area() << endl;

cout << "\n myRectangle2: ";
myRectangle2.print();    cout << endl;
cout << " Area of myRectangle2: " << myRectangle2.area() << endl;

myBox1.print();
    cout<<"surface area of Mybox" <<myBox1.area()<<endl;
    cout<<"volume of mybox1 is " <<myBox1.volume()<<endl;

myBox2.print();
    cout<<"surface area of Mybox" <<myBox2.area()<<endl;
    cout<<"volume of mybox1 is " <<myBox2.volume()<<endl;
}
```

OUTPUT:

```
 myRectangle1:
Length = 0 ; Width = 0 Area of myRectangle1: 0

 myRectangle2: Length = 8 ; Width = 6
 Area of myRectangle2: 48
Length = 0 ; Width = 0 ; Height = 0surface area of Mybox0
volume of mybox1 is 0
Length = 10 ; Width = 7 ; Height = 3surface area of Mybox242
volume of mybox1 is 210
```

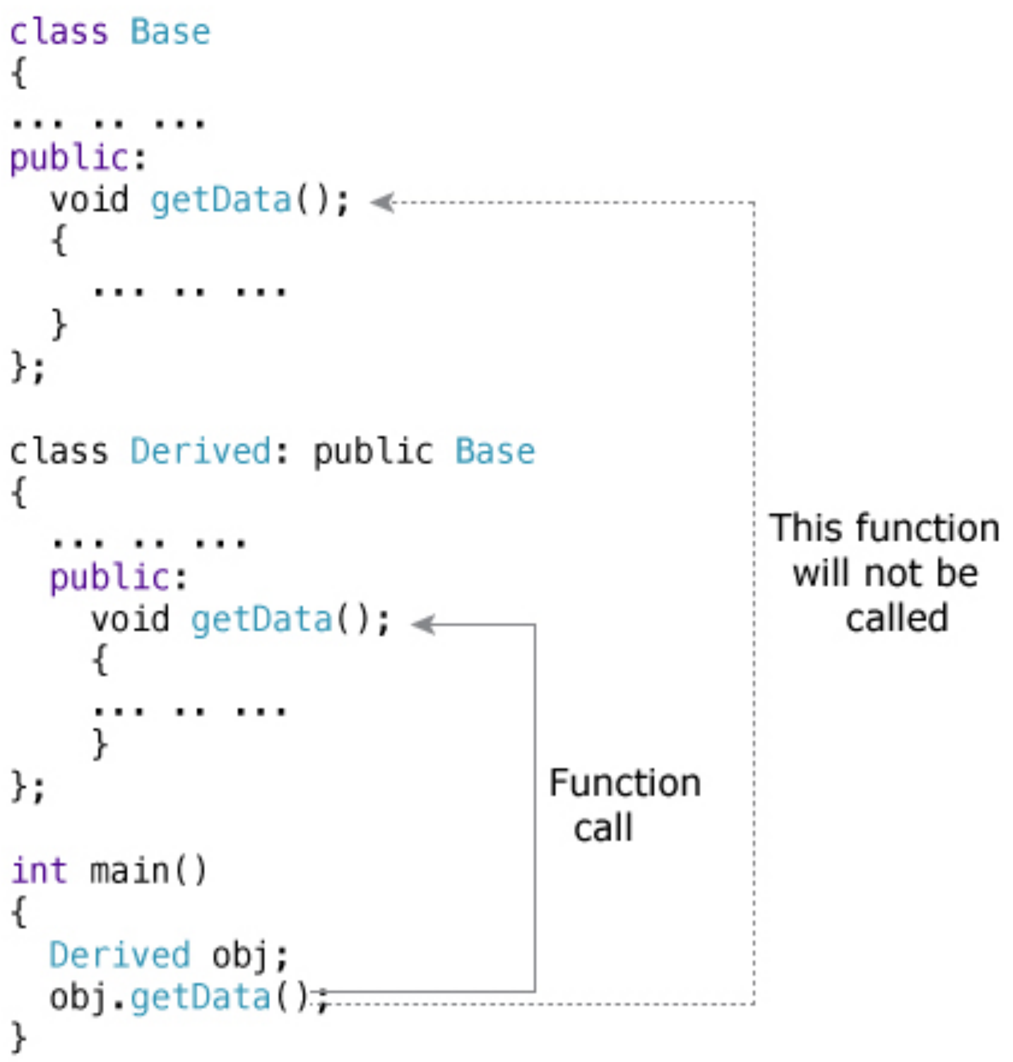
```
#include <iostream>
using namespace std;
class base // base class
{int pri; //private by default
protected:
int prot;
public:
int pub;
void set(int b) { pri=b;}
void setprot(int p) {prot=p;}
void show(){ cout<<"in base pri :"<<pri<<"\n";}
};

class drived: public base
 // drivedclass
{
int k;
public:
drived( int x) {k =x; }
void showK(){
cout<<" in derived k : "<< k << "\n";
cout<<" in deraived prot from base : "<<prot<<endl;
//cout << pri; this is error
}
} ;//end of class


int main(){

drived ob(3);
ob.set(5); // access member of base
ob.show(); // access member of base
ob.showK(); // access member of drived class

//ob.prot=5;error
}
```

OUTPUT:

```
in base pri :5
 in derived k : 3
 in deraived prot from base : 0
```

C++ Function Overriding

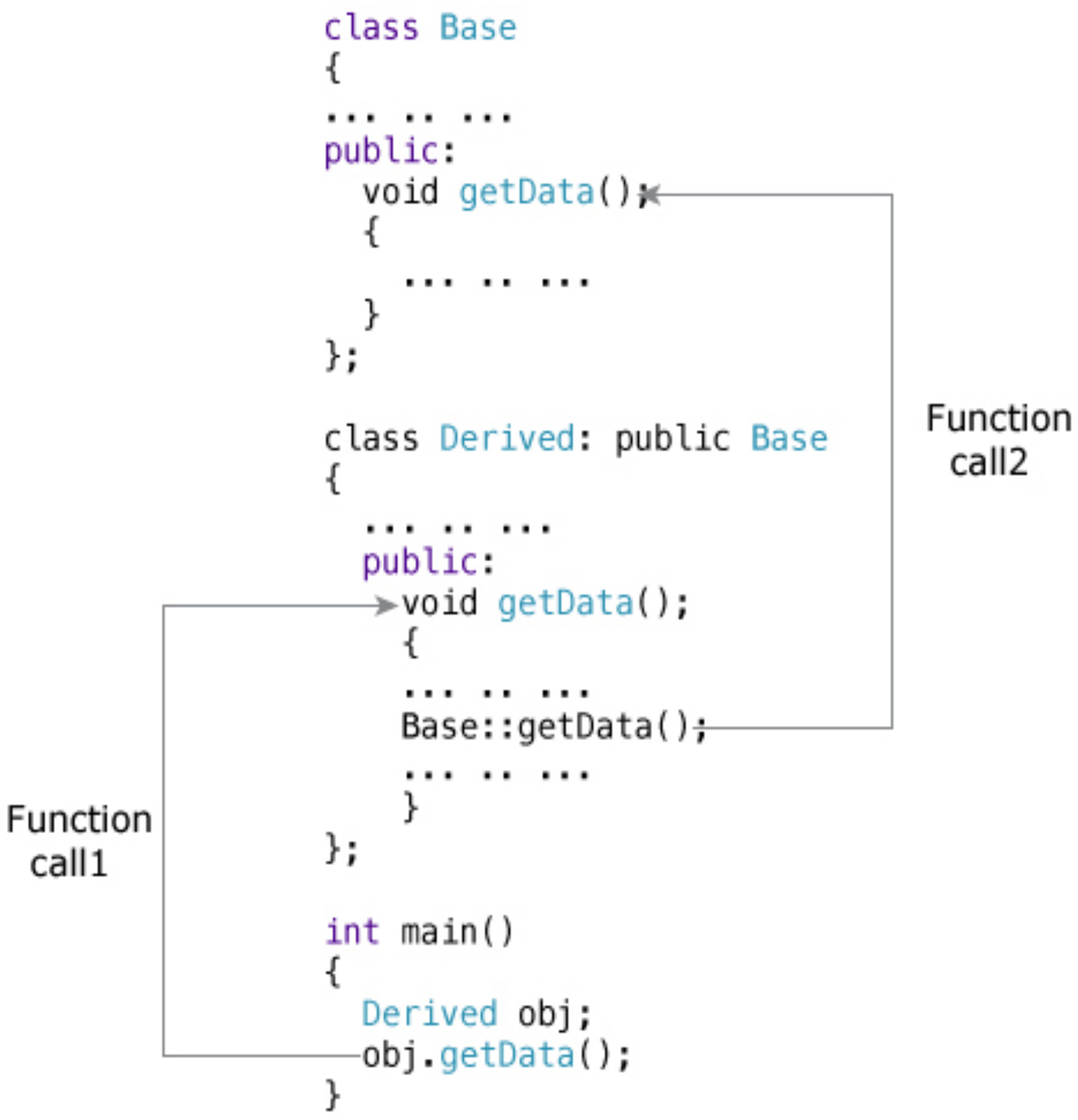
احنا عرفنا ان المطورين او المبرمجين يقدرون ينشئون new classes (Derived Classes) من كلاسات أساسية و موجودة عندنا و تقدر ترث منها مميزاتها، أحيانا يصادف وجودfunctions لها نفس الاسم و البراميتر بالكلاسات سواءً derived or base classes . و اذا قمت بإنشاء Object من الدرايف كلاس و استدعيت الدالة الموجودة بكل الكلاسين فيتم اخذ أي وحدة؟ الموجودة بالدرايف كلاس و يتجاهل البيس كلاس ، هاذي الميزة تُعرف باسم : function overriding.



```
class Base
{
... .. ...
public:
  void getData();
  {
    ... .. ...
  }
};

class Derived: public Base
{
  ... .. ...
  public:
    void getData();
    {
    ... .. ...
    }
};

int main()
{
  Derived obj;
  obj.getData();
}
```

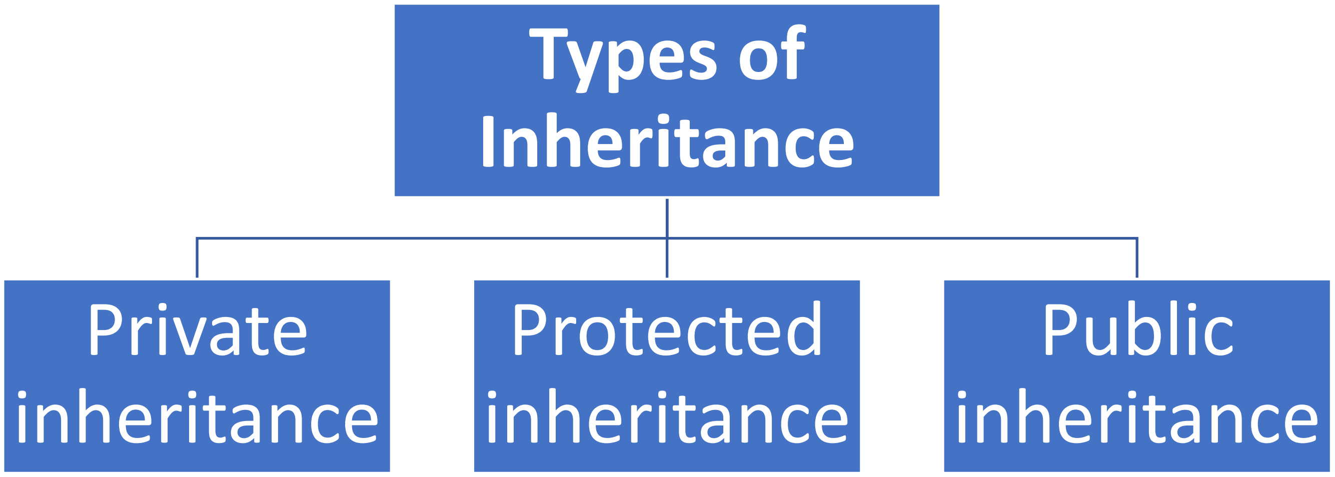
How to access the overridden function in the base class from the derived class?

طيب في حال ابغا أوصل لدالة الموجودة في البيس كلاس ايش اسوي؟ اذا هو تلقائيا بيروح لدرايف كلاس؟
نروح لدرايف كلاس و نروح لدالة الشبيهة و داخلها بنسوي كالتالي:
نستخدم اسم الكلاس الأساسي :: اسم الدالة ، نشوف المثال اذا ابغا أوصل لدالة لـ gateData() الموجودة بالبيس كلاس
نكتب كذا: Base::getData();



```
class Base
{
... .. ...
public:
  void getData();
  {
    ... .. ...
  }
};

class Derived: public Base
{
  ... .. ...
  public:
   void getData();
   {
   ... .. ...
   Base::getData();
   ... .. ...
   }
};

int main()
{
  Derived obj;
  obj.getData();
}
```

Types of Inheritance



Private:

لـمـا يـرث derived class مـن Base class و كـان نـوع الـوراثـة private فـإن كـل خـصـائـص و الـــ functions فـي Base class بـتـصـيـر private فـي Derived class.

Protected:

الـــ data و الـــ functions بـتـكـون protected مـن Base class لـلـــ derived class.

Public:

امـا اذا كـان نـوع الـوراثـة public فـهـي بـتـتـوزع عـلـى مـرحـلـتـيـن:

1. الـــ protected فـي Base class يـصـيـر protected فـي Derived class.
2. الـــ public فـي Base class يـصـيـر publicفـي Derived class.

جدول يوضح تفاصيل الوراثة :

نوع الوراثة / نوع بيانات صنف القاعدة	Private	Protected	Public
Private	لا يورث	لا يورث	لا يورث
Protected	Private	Protected	Protected
Public	Private	Protected	Public

Public:

```cpp
#include <iostream>
using namespace std;

//Base class
class Parent
{
    public:
    int id_p;
};

// Sub class inheriting from Base Class(Parent)
class Child : public Parent
{
    public:
    int id_c;
};

//main function
int main()
{

        Child obj1;

        // An object of class child has all data members
        // and member functions of class parent
        obj1.id_c = 7;
        obj1.id_p = 91;
        cout << "Child id is " << obj1.id_c << endl;
        cout << "Parent id is " << obj1.id_p << endl;

        return 0;
}
```

OUTPUT:

```
Child id is 7
Parent id is 91
```

```cpp
#include <iostream>

using namespace std;

// Base class
class Shape {
   public:
      void setWidth(int w) {
         width = w;
      }
      void setHeight(int h) {
         height = h;
      }

   protected:
      int width;
      int height;
};

// Derived class
class Rectangle: public Shape {
   public:
      int getArea() {
         return (width * height);
      }
};

int main(void) {
   Rectangle Rect;

   Rect.setWidth(5);
   Rect.setHeight(7);

   // Print the area of the object.
   cout << "Total area: " << Rect.getArea() << endl;

   return 0;
}
```

OUTPUT:

```
Total area: 35
```

```cpp
#include <iostream>
using namespace std;

class Base
{
public:
int base_value;
void base_input()
{
cout<<"Enter the integer value of base class: ";
cin>>base_value;
}
};

class Derived : public Base
{
// private by default
int derived_value;
public:
void derived_input()
{
cout<<"Enter the integer value of derived class: ";
cin>>derived_value;
}
void sum()
{
cout << "The sum of the two integer values is: " << base_value +
derived_value<<endl;
}
};

int main()
{
cout<<"Welcome to DataFlair tutorials!"<<endl<<endl;
Derived d; // Object of the derived class
d.base_input();
d.derived_input();
d.sum();
return 0;
}
```

OUTPUT:

```
Welcome to DataFlair tutorials!

Enter the integer value of base class: 6
Enter the integer value of derived class: 8
The sum of the two integer values is: 14
```

```cpp
#include <iostream>
#include <string>
using namespace std;

class Person
{
     public:
        string profession;
        int age;

        void display()
        {
             cout << "My profession is: " << profession << endl;
             cout << "My age is: " << age << endl;
             walk();
             talk();
        }
        void walk() { cout << "I can walk." << endl; }
        void talk() { cout << "I can talk." << endl; }
};

// MathsTeacher class is derived from base class Person.
class MathsTeacher : public Person
{
    public:
       void teachMaths() { cout << "I can teach Maths." << endl; }
};

// Footballer class is derived from base class Person.
class Footballer : public Person
{
    public:
       void playFootball() { cout << "I can play Football." << endl; }
};

int main()
{
     MathsTeacher teacher;
     teacher.profession = "Teacher";
     teacher.age = 23;
     teacher.display();
     teacher.teachMaths();

     Footballer footballer;
     footballer.profession = "Footballer";
     footballer.age = 19;
     footballer.display();
     footballer.playFootball();

     return 0;
}
```

OUTPUT:

```
My profession is: Teacher
My age is: 23
I can walk.
I can talk.
I can teach Maths.
My profession is: Footballer
My age is: 19
I can walk.
I can talk.
I can play Football.
```

```
using namespace std;
class Base
{
public:
    int m_nPublic; // can be accessed by anybody
private:
    int m_nPrivate; // can only be accessed by Base member functions (but not
derived classes)
protected:
    int m_nProtected; // can be accessed by Base member functions, or derived
classes.
};

class Derived: public Base
{
public:
    Derived()
    {
        // Derived's access to Base members is not influenced by the type of
inheritance used,
        // so the following is always true:

        m_nPublic = 1; // allowed: can access public base members from
derived class
        m_nPrivate = 2; // not allowed: can not access private base members
from derived class
        m_nProtected = 3; // allowed: can access protected base members from
derived class
    }
};

int main()
{
    Base cBase;
    cBase.m_nPublic = 1; // allowed: can access public members from outside
class
    cBase.m_nPrivate = 2; // not allowed: can not access private members from
outside class
    cBase.m_nProtected = 3; // not allowed: can not access protected members
from outside class
}
```