## Task 03:

### Question no 1:

Yes, a friend function can be used to do operator overloading for a class as shown in the code below

|  |
| --- |
| *#include* <iostream>  using namespace std;  class Ex {  private:  int data;  public:  Ex(int *data* = 0) : data(*data*) {}  int showData(){  *return* data;  }  friend void operator+=(Ex& *a*, const Ex& *b*);  };  void operator+=(Ex& *a*, const Ex& *b*) {  *a*.data += *b*.data;  }  int main() {  Ex a(1), b(2);  a += b;  cout << a.showData() << endl;  } |
| The Output is 3 which is correct |

### Question no 2:

Yes, a friend function can be also be used to do operator overloading for a class and a primitive data type as shown in the code below

|  |
| --- |
| *#include* <iostream>  using namespace std;  class Ex {  private:  int data;  public:  Ex(int *data* = 0) : data(*data*) {}  int showData(){  *return* data;  }  friend void operator+=(Ex& *a*, const int& *b*);  };  void operator+=(Ex& *a*, const int& *b*) {  *a*.data += *b*;  }  int main() {  Ex a(1);  int b=3;  a += b;  cout << a.showData() << endl;  } |
| The Output is 4 which is correct |

### Question no 3:

**No**, a friend function **cannot access** private or protected members **without using an object** of that class. If you call a data member without an instance of an object it will throw out an error “Cannot determine which instance of overloaded function "data" is intended”

|  |
| --- |
| *#include* <iostream>  using namespace std;  class Ex {  private:  int data;  public:  Ex(int *data* = 0) : data(*data*) {}  int showData(){  *return* data;  }  friend void operator+=(const int& *b*);  };  void operator+=(const int& *b*) {  *a*+= *b*; // error will be thrown here  }  int main() {  Ex a(1);  int b=3;  a += b;  cout << a.showData() << endl;  } |
| The Output is 4 which is correct |