You need to crack a 3 digit code based on these hints:

682 – one number is correct and in the correct position
645 – one number is correct but in the wrong position
206 – two numbers are correct but in the wrong positions
738 – nothing is correct
780 – one number is correct but in the wrong position.

Not that difficult with OPL CPLEX.

```plaintext
// Start of code

// The code we look for
dvar int x[1..3] in 0..9;

dexpr int nbGood[i in 0..9][j in 0..9][k in 0..9]=
  (i==x[1]) + (j==x[2]) + (k==x[3]);

dexpr int nbWrongPosition[i in 0..9][j in 0..9][k in 0..9]=
  (i==x[2] || i==x[3]) + (j==x[1] || j==x[3]) + (k==x[1] || k==x[2]);

subject to
{
  //682 – one number is correct and in the correct position
  nbGood[6][8][2]=1;
  nbWrongPosition[6][8][2]==0;

  //645 – one number is correct but in the wrong position
  nbGood[6][4][5]==0;
  nbWrongPosition[6][4][5]==1;

  //206 – two numbers are correct but in the wrong positions
  nbGood[2][0][6]==0;
  nbWrongPosition[2][0][6]==2;

  //738 – nothing is correct
  nbGood[7][3][8]==0;
  nbWrongPosition[7][3][8]==0;

  //780 – one number is correct but in the wrong position.
  nbGood[7][8][0]==0;
  nbWrongPosition[7][8][0]==1;
}
```
```execute
{
writeln(x);
}

// End of code

Which gives

\[0 \ 5 \ 2\]

So the hidden code is 052

**NB:**

There is no other solution since if we add the constraint

\[x[1]!=0 \ || \ x[2]!=5 \ || \ x[3]!=2;\]

Then no solution.

Plus by default OPL relies on CPLEX Mathematical Programming but we can also use Constraint Programming. For this, we simply add

```using CP;```

At the beginning of the model.

Finally, let’s note that the free CPLEX Community Edition is enough for all this.