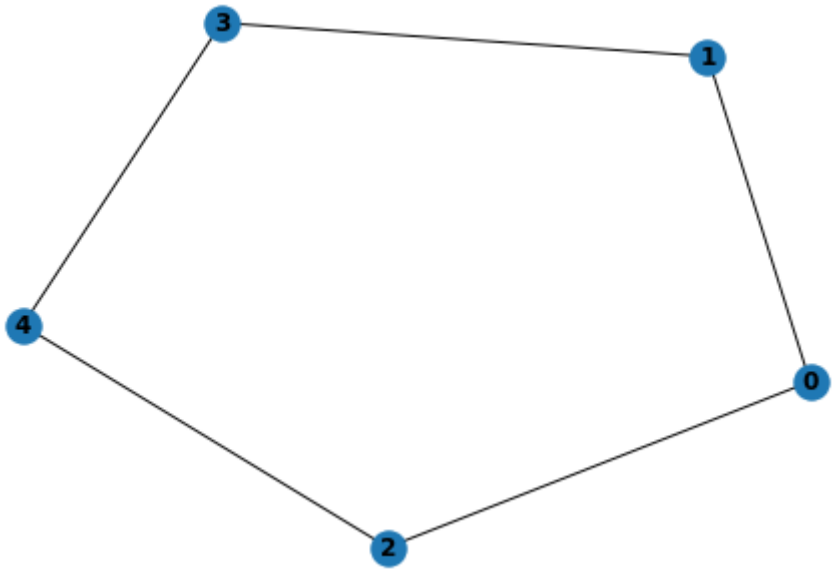
Max cut -

- https://www.youtube.com/watch?v=claoY57eVIc

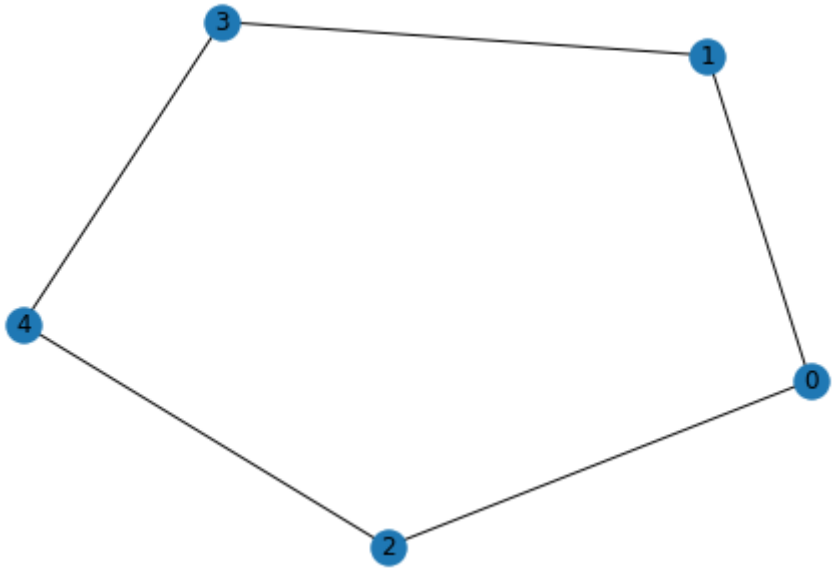
Method for solving:-

- Step 0: Convert max-cut to quadratic problem
- Step 1: Converting `QuadraticProgram` to to QUBO
- Step 2: QUBO to Ising Hamiltonian
- Step 3: Minimum Eigen Optimizer - Used to solve Ising Hamiltonion
  - Step 3-Options: Available solvers are:-
    - `QAOA`
    - `VQE`
    - `CplexSolver` (classical exact method)

In [47]:
```
import networkx as nx
# Make a graph with degree=2 and #node=5
graph = nx.random_regular_graph(d=2, n=5, seed=111) # n*d must be even & d<n
pos = nx.spring_layout(graph, seed=111)
nx.draw(graph, pos=pos, with_labels=True, font_weight='bold')
```



In [48]:
```
# Application class for a Max-cut problem
# Make a Max-cut problem from the graph
from qiskit_optimization.applications import Maxcut
maxcut = Maxcut(graph)       #passed graph to constructor of Maxcut
maxcut.draw(pos=pos)
```



In [49]:

```
# Step 0: Convert max-cut to quadratic problem
# Make a QuadraticProgram by calling to_quadratic_program()
qp = maxcut.to_quadratic_program()
print(qp)
```

```
\ This file has been generated by DOcplex
\ ENCODING=ISO-8859-1
\Problem name: Max-cut

Maximize
 obj: 2 x_0 + 2 x_1 + 2 x_2 + 2 x_3 + 2 x_4 + [ - 4 x_0*x_1 - 4 x_0*x_2
      - 4 x_1*x_3 - 4 x_2*x_4 - 4 x_3*x_4 ]/2
Subject To

Bounds
 0 <= x_0 <= 1
 0 <= x_1 <= 1
 0 <= x_2 <= 1
 0 <= x_3 <= 1
 0 <= x_4 <= 1

Binaries
 x_0 x_1 x_2 x_3 x_4
End
```

In [50]:

```
# Step 1: Converting `QuadraticProgram` to to QUBO
# Step 2: QUBO to Ising Hamiltonian
# Step 3: Minimum Eigen Optimizer - Used to solve Ising Hamiltonion
from qiskit import Aer
from qiskit.utils import QuantumInstance
from qiskit_optimization.algorithms import MinimumEigenOptimizer
from qiskit.algorithms import QAOA, NumPyMinimumEigensolver

qins = QuantumInstance(backend=Aer.get_backend('qasm_simulator'), shots=1000,
# Define QAOA solver
meo = MinimumEigenOptimizer(min_eigen_solver=QAOA(reps=1, quantum_instance=qi
result = meo.solve(qp)
print('result:\n', result)
print('\nsolution:\n', maxcut.interpret(result))
print('\ntime:', result.min_eigen_solver_result.optimizer_time)
```

```
result:
 optimal function value: 4.0
optimal value: [0. 0. 1. 1. 0.]
```

```
status: SUCCESS

solution:
 [[0, 1, 4], [2, 3]]

time: 0.09071207046508789
```

In [51]:

```
maxcut.draw(result, pos=pos)
```