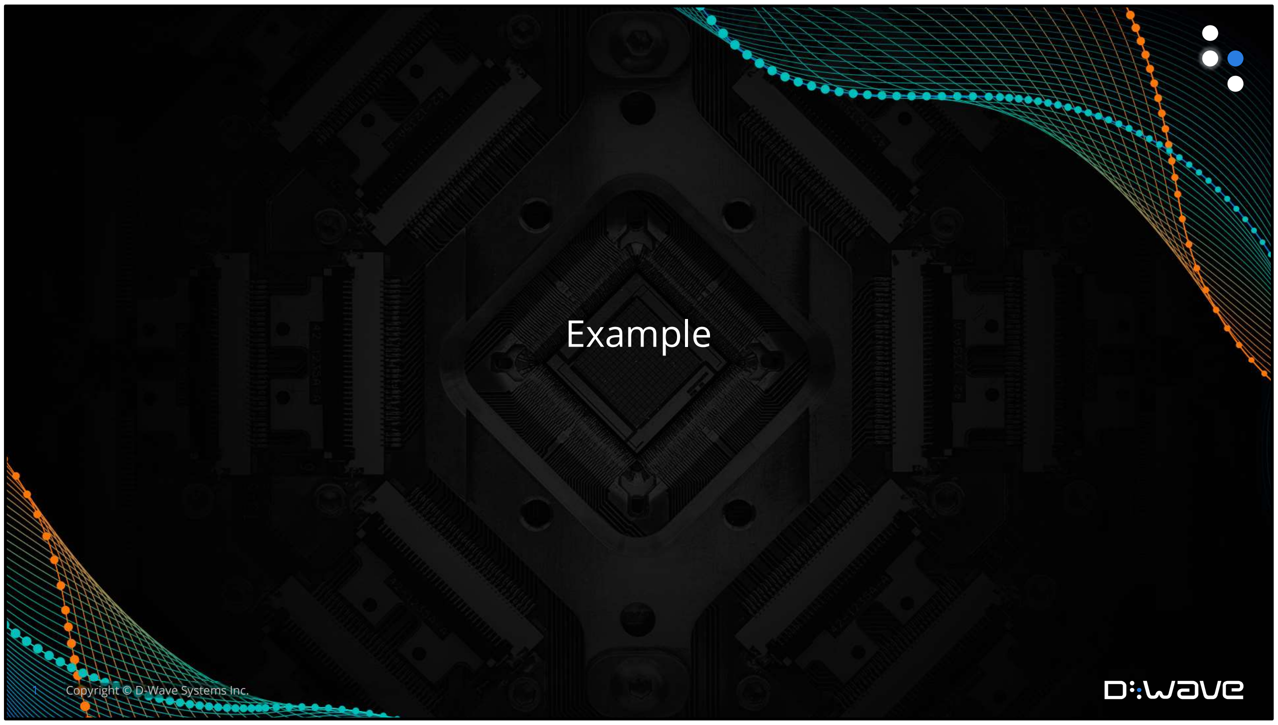
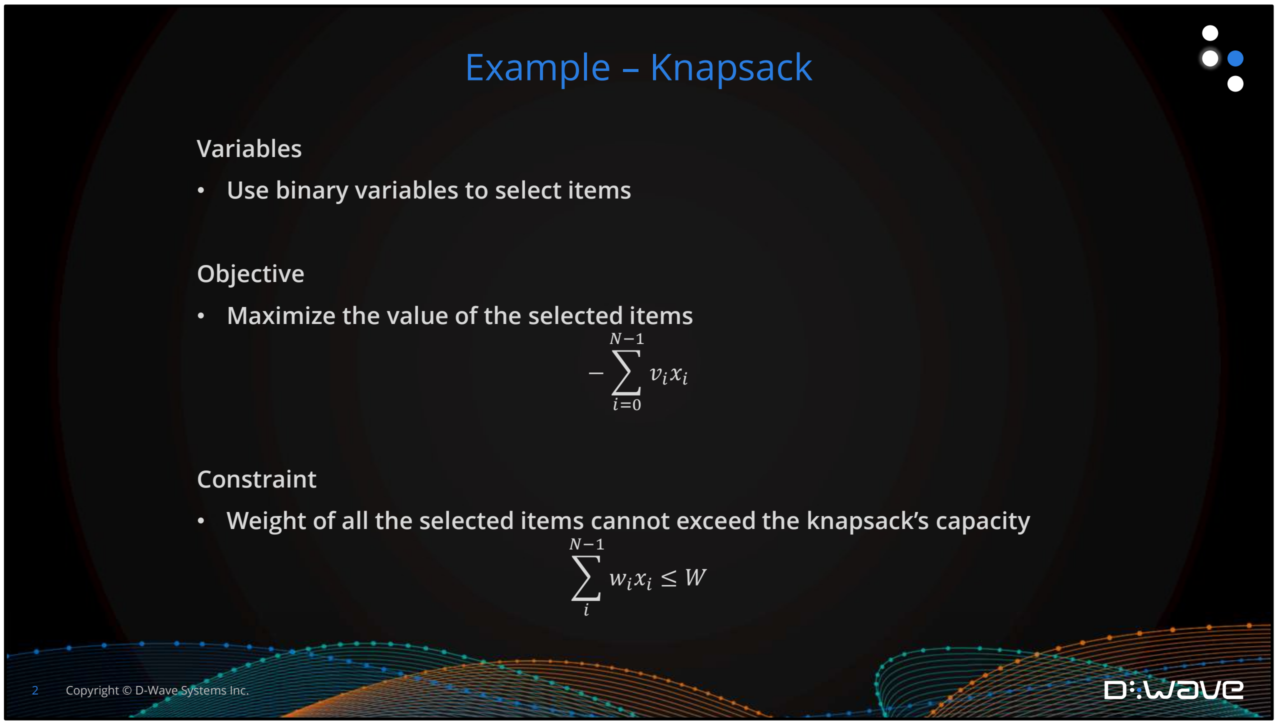


Example

Now let's put everything we've learned together by looking at an example.



Example – Knapsack

Variables

- Use binary variables to select items

Objective

- Maximize the value of the selected items

$$-\sum_{i=0}^{N-1} v_i x_i$$

Constraint

- Weight of all the selected items cannot exceed the knapsack's capacity

$$\sum_{i}^{N-1} w_i x_i \leq W$$

Let's consider the knapsack problem.

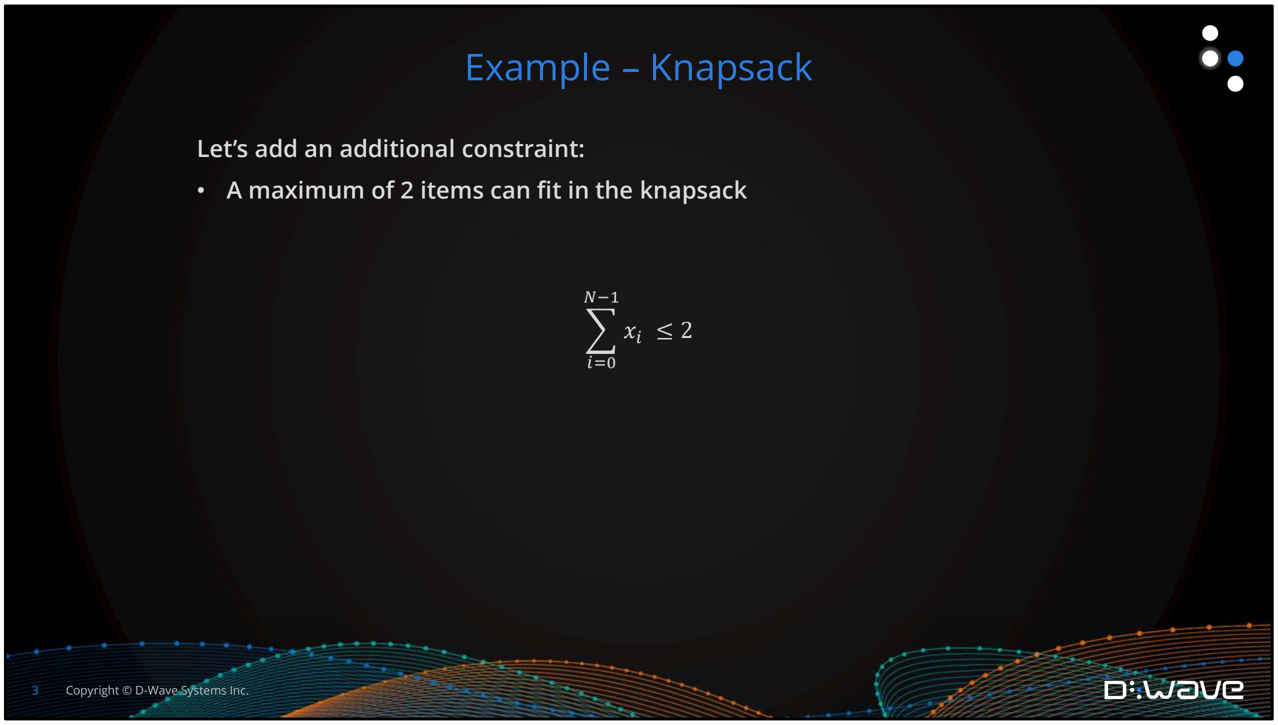
In this problem we want to select items to put into a knapsack, truck or other container so that we maximize the total value of the selected items.
We also need to make sure we don't exceed the weight capacity of the container.

To model this problem we'll use binary variables since we want to choose items.
If a binary variable is 1 the item it represents goes into the knapsack and if it's 0 the item is left behind.

The objective is to maximize the total value of the items selected, so we're going to add up the value of each item multiplied by the item's binary variable.
In the objective expression here, v is the value of each item and x is the binary variable for the item.
We multiple this summation by -1 to convert it from a maximization to a minimization since the CQM solver seeks the minimum energy solution.

This problem also has an inequality constraint. The total weight of all of the chosen items cannot exceed the maximum weight capacity of the knapsack.

We express this with the inequality expression shown here, where the lowercase w in the summation represents the weight of each item and the upper case W is the knapsack's maximum weight.



Example – Knapsack

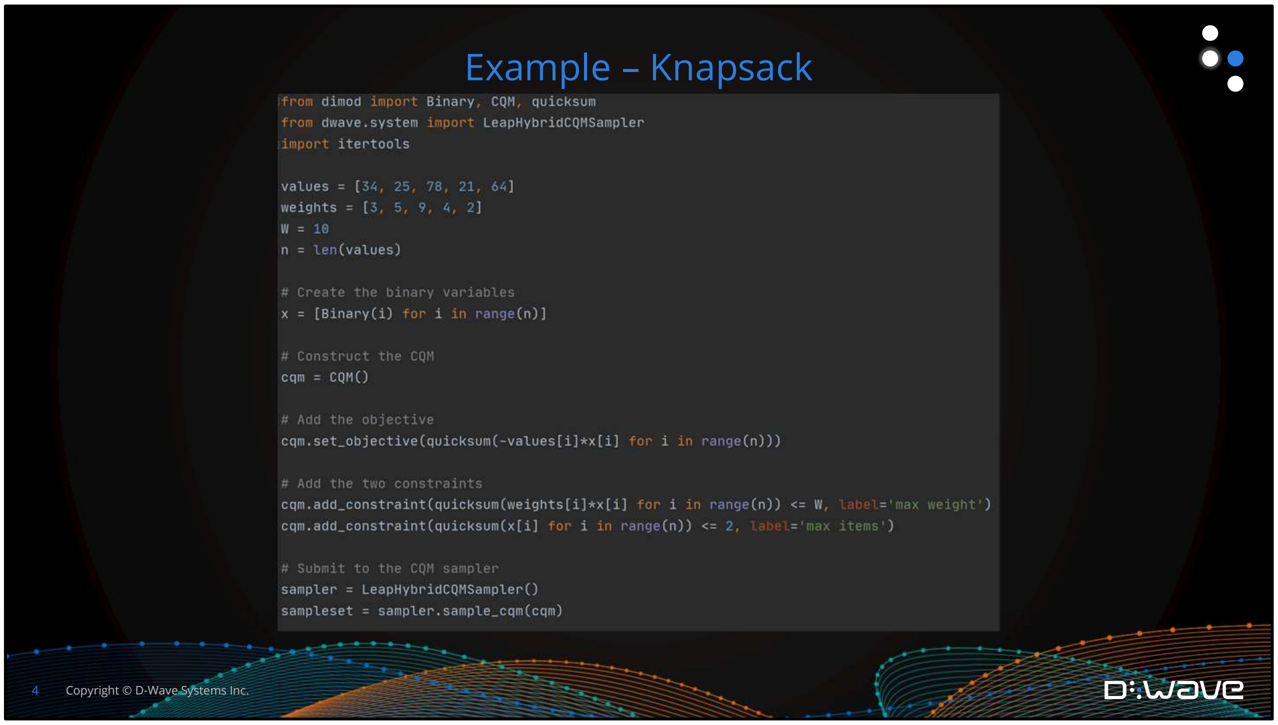
Let's add an additional constraint:

- A maximum of 2 items can fit in the knapsack

$$\sum_{i=0}^{N-1} x_i \leq 2$$

For the sake of this example, let's add a second constraint. A maximum of 2 items can fit in the knapsack.

We'll express this with another inequality expression, where we sum up the binary variables and require that summation to be less than or equal to 2.



Example – Knapsack

```python
from dimod import Binary, CQM, quicksum
from dwave.system import LeapHybridCQMSampler
import itertools

values = [34, 25, 78, 21, 64]
weights = [3, 5, 9, 4, 2]
W = 10
n = len(values)

# Create the binary variables
x = [Binary(i) for i in range(n)]

# Construct the CQM
cqm = CQM()

# Add the objective
cqm.set_objective(quicksum(-values[i]*x[i] for i in range(n)))

# Add the two constraints
cqm.add_constraint(quicksum(weights[i]*x[i] for i in range(n)) <= W, label='max weight')
cqm.add_constraint(quicksum(x[i] for i in range(n)) <= 2, label='max items')

# Submit to the CQM sampler
sampler = LeapHybridCQMSampler()
sampleset = sampler.sample_cqm(cqm)
```

4 Copyright © D-Wave Systems Inc.

Here’s how we would program this example.

For the sake of this example, we’re going to look at a really small problem that only has 5 items.
The value and weight of each item is defined in the values and weights lists.
The maximum weight of the knapsack is set to 10.

We start by creating 5 binary variables, one for each item.

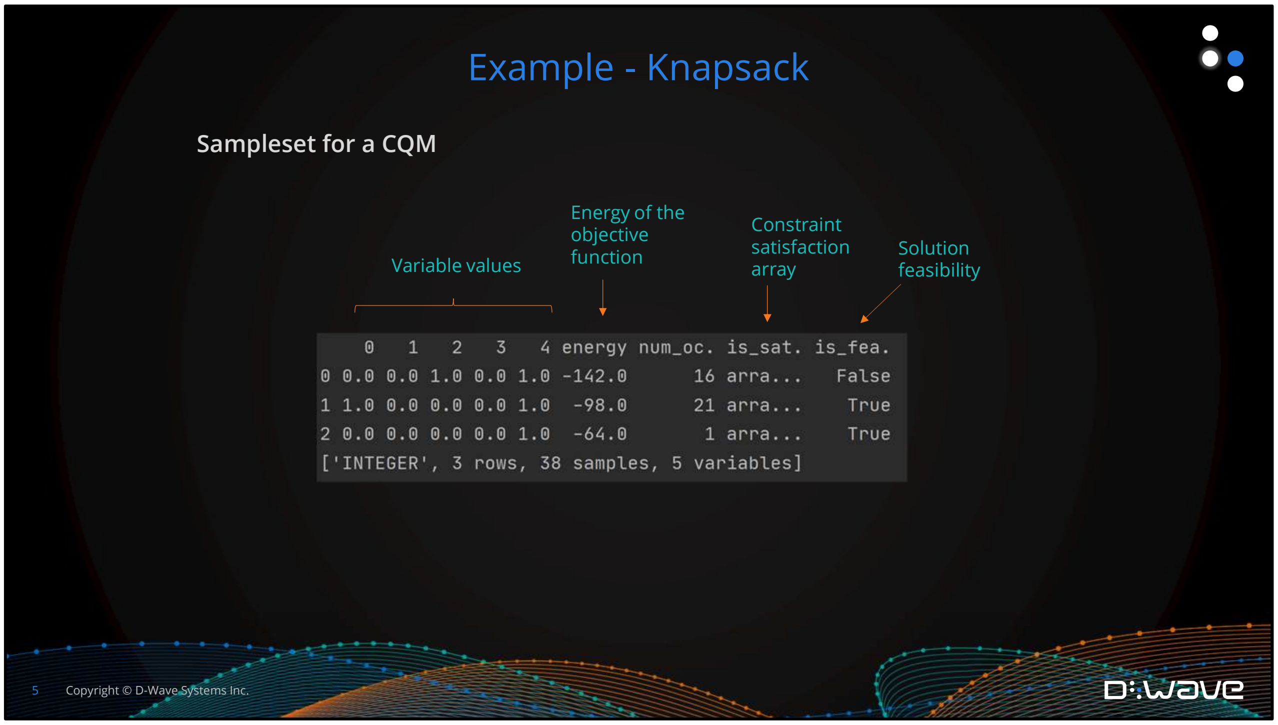
Then we create the CQM and set the objective to maximize the value of the selected items.
Notice the negative sign in front of the multiplication of the values times the binary variables.
Again, that makes the maximization objective compatible with the CQM solver that naturally minimizes.

Next we add the two inequality constraints.
First we sum up the weight of each item times the binary variable associated with the item and set the sum as less than or equal to the maximum weight of the knapsack

We call this constraint ‘max weight’.

Then we add the constraint that limits the number of items chosen to a maximum of 2.
That constraint is labelled ‘max items’

Lastly, we instantiate the CQM sampler and submit the CQM to it.



Example - Knapsack

Sampleset for a CQM

Variable values
Energy of the objective function
Constraint satisfaction array
Solution feasibility

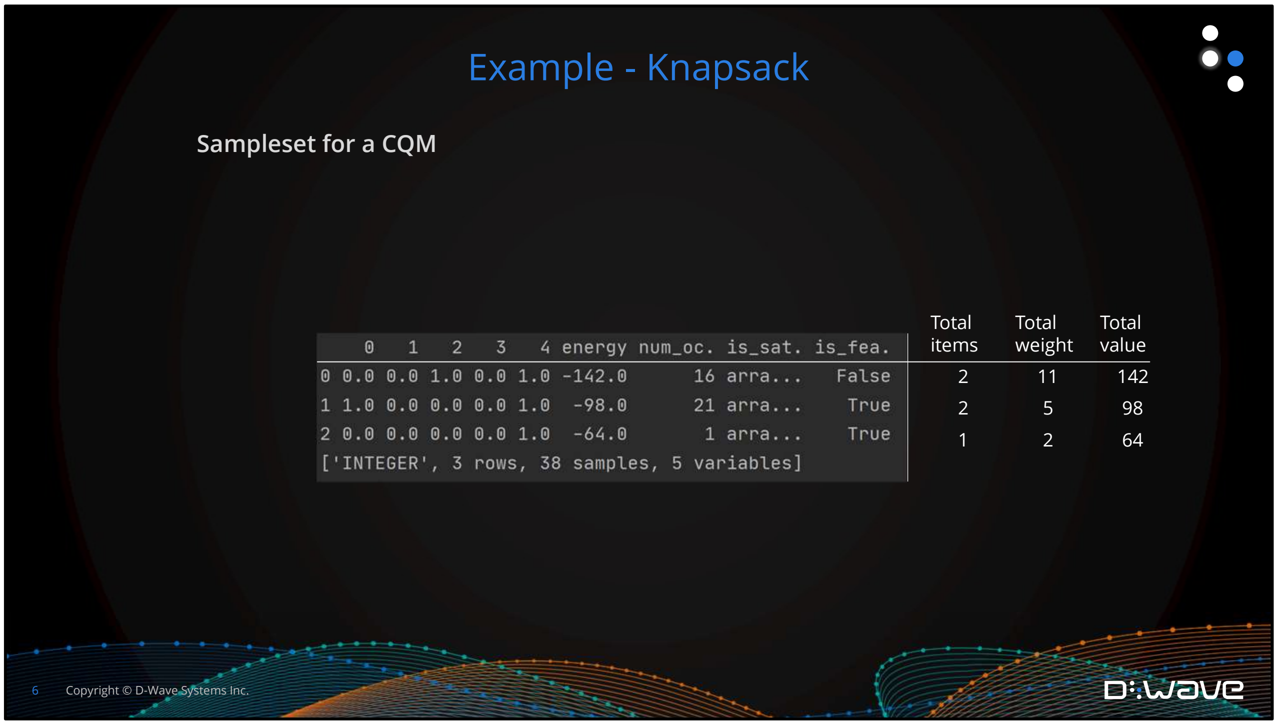
```
    0   1   2   3   4 energy num_oc. is_sat. is_fea.
0 0.0 0.0 1.0 0.0 1.0 -142.0      16 arra...   False
1 1.0 0.0 0.0 0.0 1.0  -98.0      21 arra...    True
2 0.0 0.0 0.0 0.0 1.0  -64.0       1 arra...    True
['INTEGER', 3 rows, 38 samples, 5 variables]
```

Now we're going to look at the solution.
The CQM sampler returns a sample set.

There are a few differences between this sampleset and those returned by the other solvers.
The first is that the energy is only calculated from the objective function, so the constraints don't impact the energy.
The second difference is that we have two additional columns in the sampleset returned by the CQM solver.
The is_satisfied column contains an array for each sample. The array contains the state of each constraint in the sample. In other words it tells us whether or not each constraint was satisfied.
The is_feasible column tells us if the sample is feasible. A sample is only feasible if all constraints are satisfied.

You'll notice that the CQM solver returns feasible solutions, as well as low energy infeasible solutions. We report individual constraint satisfaction so you can examine the low energy infeasible solutions.
Depending on the application, there may be cases where a low energy infeasible solution is worth considering depending on how many and which

constraints are violated.



Example - Knapsack

Sampleset for a CQM

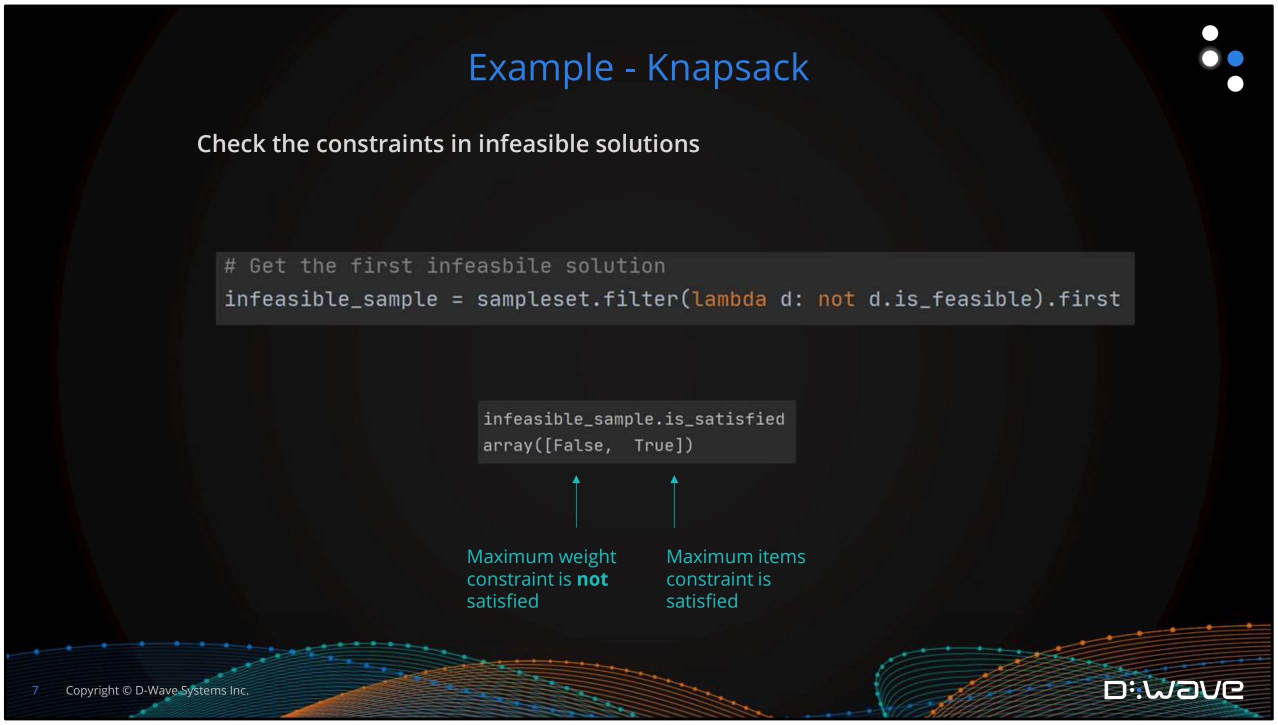
```
    0   1   2   3   4 energy num_oc. is_sat. is_fea.
0 0.0 0.0 1.0 0.0 1.0 -142.0      16 arra...   False
1 1.0 0.0 0.0 0.0 1.0  -98.0      21 arra...    True
2 0.0 0.0 0.0 0.0 1.0  -64.0       1 arra...    True
['INTEGER', 3 rows, 38 samples, 5 variables]
```

Total items	Total weight	Total value
2	11	142
2	5	98
1	2	64

6

If we look at each sample in the samplset, we can see that all samples returned solutions that select 1 or 2 items, so the 'max items' constraint is satisfied in all.

Remember that the maximum weight capacity for the knapsack is 10.
In this example, the best solutions that satisfy both constraints (the last two samples in the sampleset) have a relatively low total value. They values come in at 98 and 64 as is shown on the right.
If we can violate the maximum capacity constraint just a little, we can increase the total value to 142.
In some cases we might have that flexibility and it might be worth violating the constraint a little in order to increase the total value provided by the objective.



Example - Knapsack

Check the constraints in infeasible solutions

```
# Get the first infeasbile solution
infeasible_sample = sampleset.filter(lambda d: not d.is_feasible).first
```

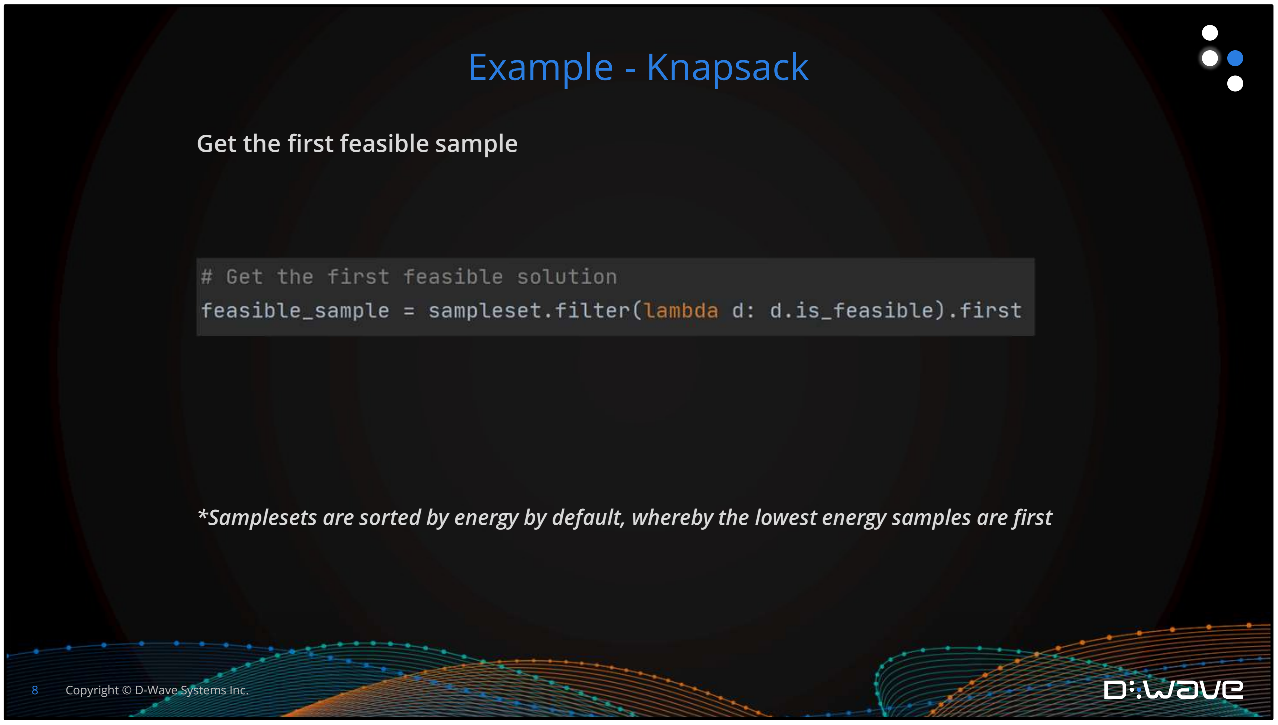
```
infeasible_sample.is_satisfied
array([False,  True])
```

Maximum weight constraint is **not** satisfied

Maximum items constraint is satisfied

7 Copyright © D-Wave Systems Inc.

If this is the case and you want to examine the infeasible solutions, you can get the lowest energy infeasible solution using the filter function on the sampleset. Here we see what we discerned on the previous slide, that the maximum weight constraint is not satisfied, but the maximum items constraint is.



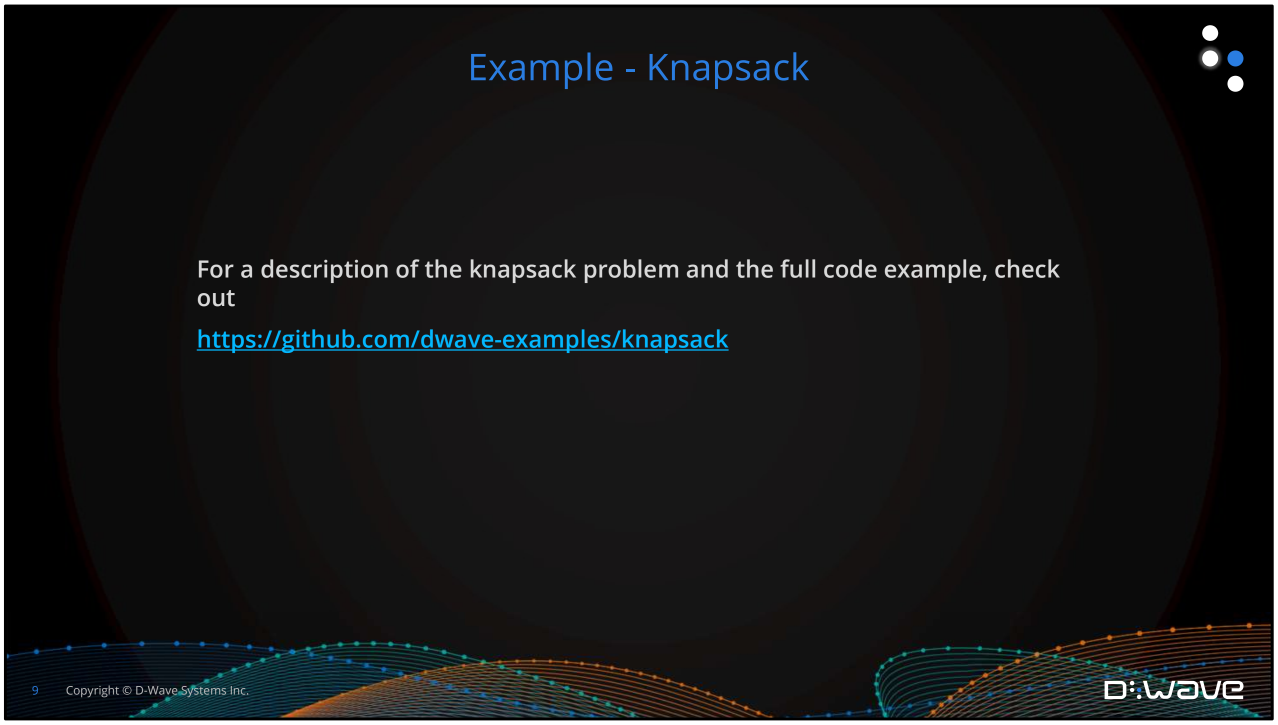
Example - Knapsack

Get the first feasible sample

```
# Get the first feasible solution
feasible_sample = sampleset.filter(lambda d: d.is_feasible).first
```

**Samplesets are sorted by energy by default, whereby the lowest energy samples are first*

If we want to retrieve the lowest energy feasible solution, we can also do this with the filter function.
As you saw above, the first sample isn't necessarily the lowest energy feasible solution.
The samples in samplesets are sorted by energy by default.



Example - Knapsack

For a description of the knapsack problem and the full code example, check out

https://github.com/dwave-examples/knapsack

9 Copyright © D-Wave Systems Inc.

For a full description of the knapsack problem and a code example with a larger problem, check out the knapsack example in our dwave-examples repo on github or in the collection of examples on Leap