Binary Desision Diagrams

What’s that?

- Method of Boolean function visualization.
- So, you only want to use it if you have a Boolean function, you know)

PyEDA

- Module for million of optimization features
- Has a submodule for BDD visualization

```
import pyeda.inter as pi
```

- Which uses graphviz for visualization

```
from graphviz import
```

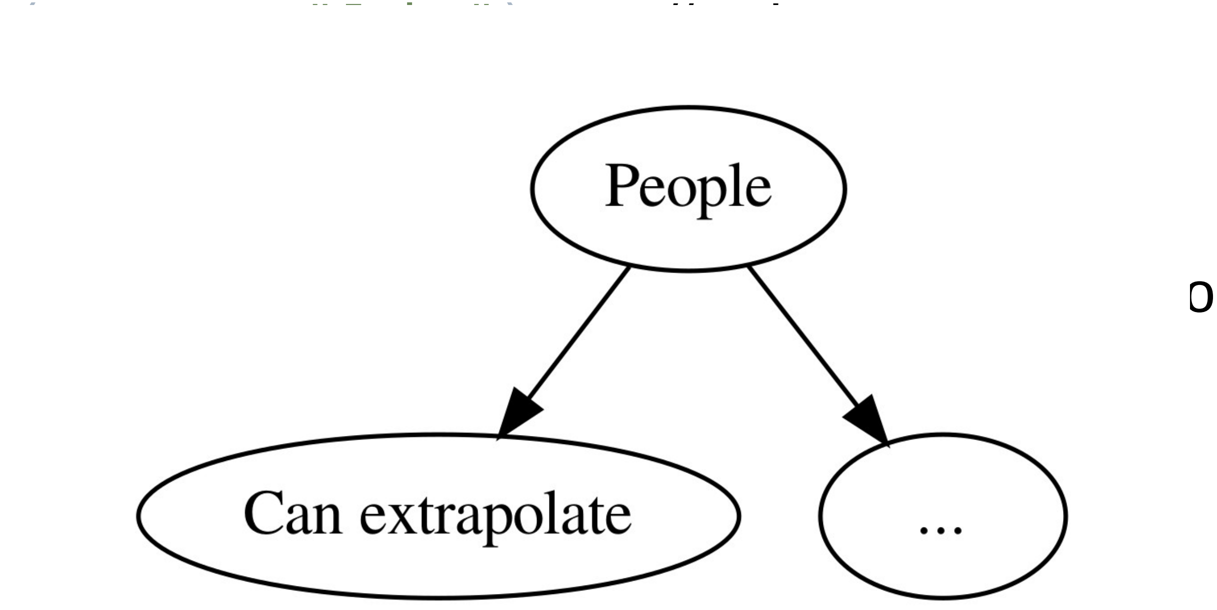
A little graphviz demonstration

```python
gr = Digraph(comment="Joke")
gr.node("A", "People")
gr.node("B", "Can extrapolate")
gr.node("C", "...")
gr.edges(["AC", "AB"])
print(gr.source)
gr.view()
```

```
Out:
// Joke
digraph {
A [label=People]
B [label="Can extrapolate"]
C [label="..."]
A -> C
A -> B
}
```

A little graphviz demonstration

```
gr = Digraph
gr.node("A",
gr.node("B",
extrapolate'
gr.node("C",
gr.edges(["A
print(gr.sou
gr.view()
```



olate"]

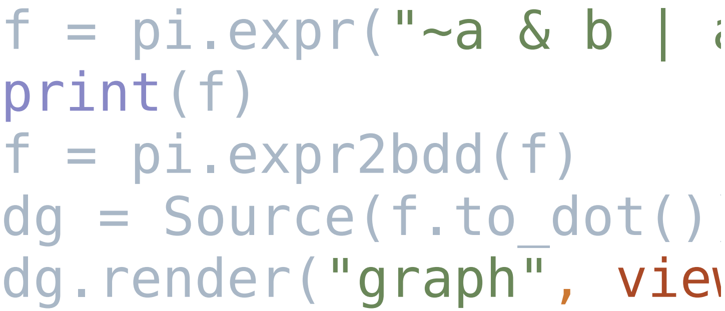
BGG visualisation

```
f = pi.expr("~a & b | a & c | b & ~c")
print(f)
f = pi.expr2bdd(f)
dg = Source(f.to_dot())
dg.render("graph", view=True)
```

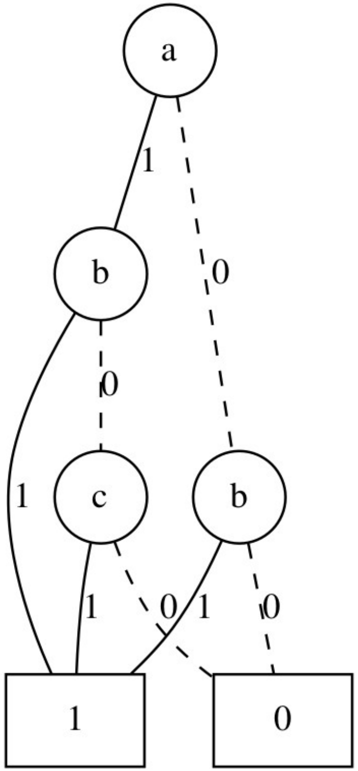
Out:

Or(And(~a, b),
And(a, c),
And(b, ~c))

BGG visualisation



```
f = pi.expr("~a & b | a
print(f)
f = pi.expr2bdd(f)
dg = Source(f.to_dot()
dg.render("graph", vie
```



Out:

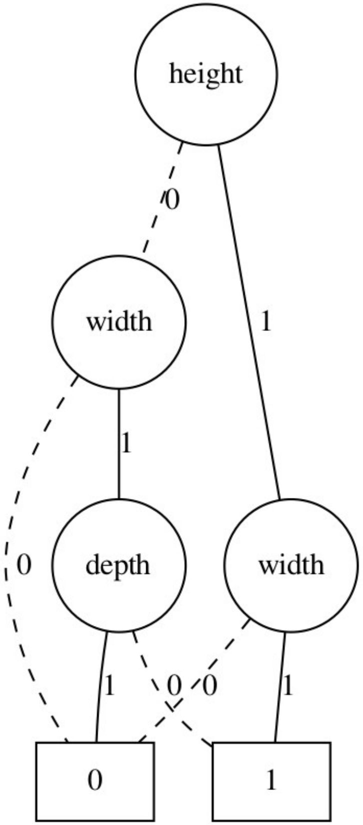
Or(And(~a, b),
And(a, c),
And(b, ~c))

Other way of variable definition

```
a, b, c = map(pi.bddvar, ["height", "width", "depth"])
f = a & b | b & ~ c
dg = Source(f.to_dot())
dg.render("graph", view=True)
```

Other way of variable definition

```
a, b, c = map(pi.bddvar                         width", "depth"])
f = a & b | b & ~ c
dg = Source(f.to_dot())
dg.render("graph", view
```



What if I need million variables?

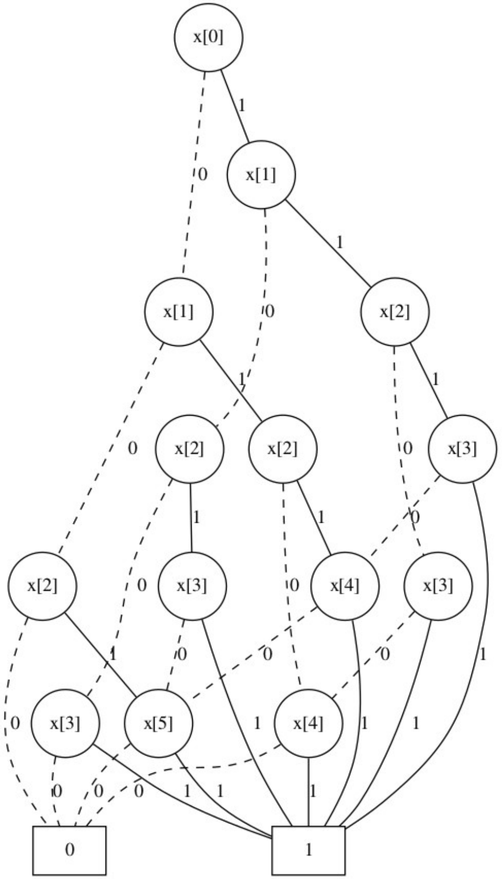
```
X = pi.bddvars('x', 8)

f2 = X[0] & X[3] | X[1] & X[4] | X[2] & X[5]
f2 = pi.expr2bdd(f2)
dg = Source(f2.to_dot())
dg.render("graph", view=True)
```

What if I need million variables?

```
X = pi.bddvars('x', 8

f2 = X[0] & X[3] | X                            & X[5]
f2 = pi.expr2bdd(f2)
dg = Source(f2.to_dot
dg.render("graph", vi
```

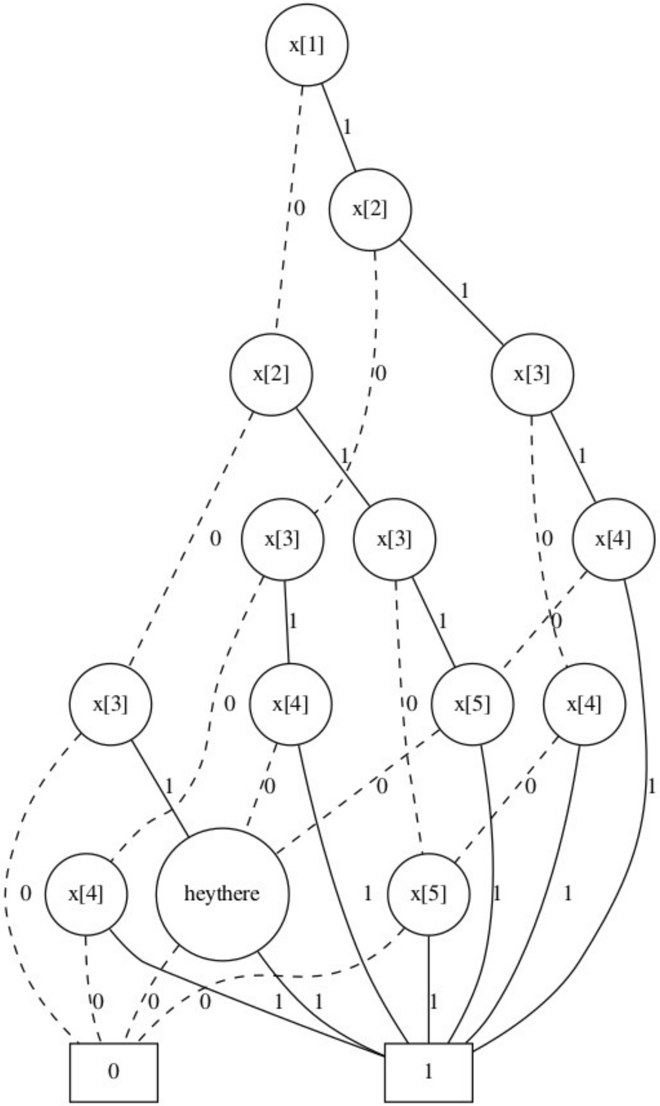


You may even name them!

```
X[0] = pi.bddvar("heythere")
```

You m them!

```
X[0] = pi.bddvar("
```



But here is a thing…

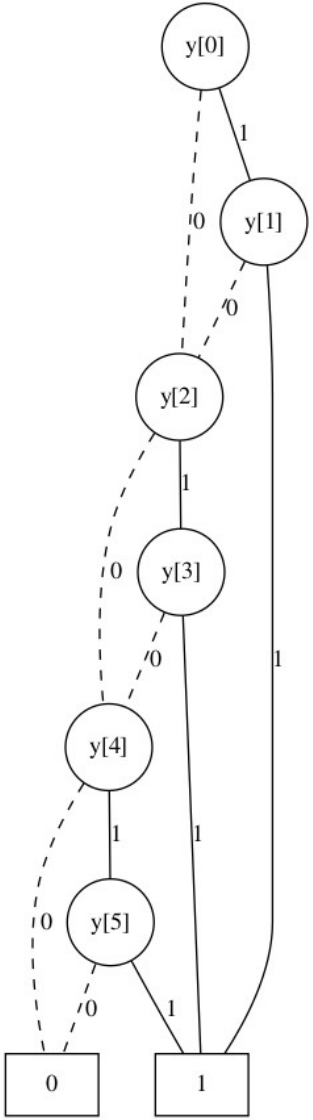
- The visualization depends on the order in which you specify your variables in the Boolean function. And it can be less good-looking (previous slide) and more good-looking (next slide). You can explicitly specify the order you want like this:

```
Y = pi.bddvars('y', 8)
g2 = f2.compose({X[0]: Y[0], X[1]: Y[2], X[2]: Y[4],
                 X[3]: Y[1], X[4]: Y[3], X[5]: Y[5]})
g2 = pi.expr2bdd(g2)
dg = Source(g2.to_dot())
dg.render("graph", view=True)
```

But her ing…

- The visualization depends which you specify your variables in the Boolean fu an be less good-looking (previous slide) and more ext slide). You can explicitly specify the order you want

```
Y = pi.bddvars('y', 8)
g2 = f2.compose({X[0]: Y             [2], X[2]: Y[4],
                 X[3]: Y             [3], X[5]: Y[5]})
g2 = pi.expr2bdd(g2)
dg = Source(g2.to_dot())
dg.render("graph", view=
```


y[0]
y[1]
y[2]
y[3]
y[4]
y[5]
0
1