## What does it mean to order a set?

When the ordering module is called with a set then all these functions are suddenly available on the set: first, last, next, prev, etc.

first returns the first atom.

last returns the last atom.

first.next returns the second atom.

But wait!

A set is, by definition, unordered. So how can you order a set?

Consider this set of colors:

**abstract** **sig** Color {}
**one** **sig** red **extends** Color {}
**one** **sig** yellow **extends** Color {}
**one** **sig** green **extends** Color {}

What does it mean to order that set of colors? Suppose we call the ordering module:

**open** util/ordering[Color]

What does first return? What does last return? What does first.next return?

Let’s have Alloy generate some instances:

**run** {}

Here are a few of the instances that are generated:

### Instance #1

first returns: yellow
last returns: green
first.next returns: red

### Instance #2

first returns: yellow
last returns: red
first.next returns: green

### Instance #3

first returns: green
last returns: yellow
first.next returns: red

Notice that the ordering is different with each instance.

Now, let’s order a plain signature:

**open** util/ordering[Time]

**sig** Time {}

**run** {}

Only one instance is generated:

### Instance #1

first returns: Time0
last returns: Time2
first.next returns: Time1

No more instances!

### Lessons Learned

1. For a set created by enumerating its atoms, the ordering module orders the set in any way.

2. For a set created by a signature, the ordering module orders the set this way: Blah0, Blah1, Blah2, …, where “Blah” is the signature name.

The real lesson, however, is that the functions provided in the ordering module (first, last, next, etc.) make it *appear* that the set is ordered. But that is an illusion, it is just a view placed on top of the set. The set, in fact, has no ordering.