Multi-input Comprehensions

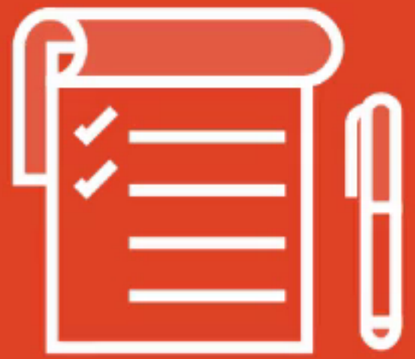
Austin Bingham
COFOUNDER - SIXTY NORTH

@austin_bingham

Robert Smallshire
COFOUNDER - SIXTY NORTH

@robsmallshire

Overview



Multi-input comprehensions

Equivalence to for-loops

Nested comprehensions

These are simply applying existing syntax and semantics

Multiple inputs and nesting apply to all comprehension types

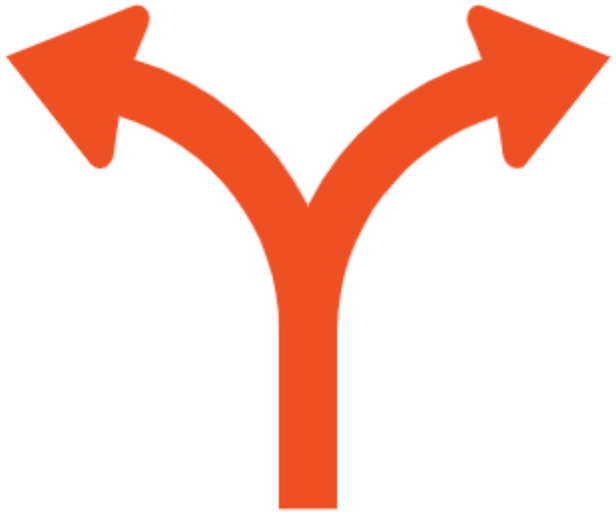
Comprehensions

```
>>> l = [i * 2 for i in range(10)]
>>> d = {i: i * 2 for i in range(10)}
>>> type(d)
<class 'dict'>
>>> s = {i for i in range(10)}
>>> type(s)
<class 'set'>
>>> g = (i for i in range(10))
>>> type(g)
<class 'generator'>
>>>
```

Comprehensions can have multiple input iterables and if-clauses.

Multi-input Comprehensions

```
>>> [(x, y) for x in range(5) for y in range(5)]
[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 0), (1, 1), (1, 2), (1, 3), (1, 4),
 (2, 0), (2, 1), (2, 2), (2, 3), (2, 4), (3, 0), (3, 1), (3, 2), (3, 3), (3, 4),
 (4, 0), (4, 1), (4, 2), (4, 3), (4, 4)]
>>> points = []
>>> for x in range(5):
...     for y in range(5):
...         points.append((x, y))
...
>>>
```



You can have multiple if-clauses in a comprehension as well

Later clauses are nested inside earlier clauses

Multiple if-clauses

```
>>> values = [x / (x - y)
...           for x in range(100)
...           if x > 50
...           for y in range(100)
...           if x - y != 0]
>>> values = []
>>> for x in range(100):
...     if x > 50:
...         for y in range(100):
...             if x - y != 0:
...                 values.append(x / (x - y))
...
>>> [(x, y) for x in range(10) for y in range(x)]
[(1, 0), (2, 0), (2, 1), (3, 0), (3, 1), (3, 2), (4, 0), (4, 1), (4, 2), (4, 3),
 (5, 0), (5, 1), (5, 2), (5, 3), (5, 4), (6, 0), (6, 1), (6, 2), (6, 3), (6, 4),
 (6, 5), (7, 0), (7, 1), (7, 2), (7, 3), (7, 4), (7, 5), (7, 6), (8, 0), (8, 1),
 (8, 2), (8, 3), (8, 4), (8, 5), (8, 6), (8, 7), (9, 0), (9, 1), (9, 2), (9, 3),
 (9, 4), (9, 5), (9, 6), (9, 7), (9, 8)]
>>> result = []
>>> for x in range(10):
...     for y in range(x):
...         result.append((x, y))
...
>>>
```

Nested Comprehensions

We've seen how to use multiple input iterables in a comprehension

It's also possible to use comprehensions in the output expression

Each element of the collection can be a comprehension

Nested Comprehensions

```
>>> vals = [[y * 3 for y in range(x)] for x in range(10)]
>>> outer = []
>>> for x in range(10):
...     inner = []
...     for y in range(x):
...         inner.append(y * 3)
...     outer.append(inner)
...
>>> vals
[[], [0], [0, 3], [0, 3, 6], [0, 3, 6, 9], [0, 3, 6, 9, 12], [0, 3, 6, 9, 12, 15
], [0, 3, 6, 9, 12, 15, 18], [0, 3, 6, 9, 12, 15, 18, 21], [0, 3, 6, 9, 12, 15,
18, 21, 24]]
>>>
```

We've been using list comprehensions for demonstrations so far.

Everything we've discussed applies to all types of comprehensions.

Other Comprehension Types

```
>>> {x * y for x in range(10) for y in range(10)}
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 24, 25, 27, 28, 3
0, 32, 35, 36, 40, 42, 45, 48, 49, 54, 56, 63, 64, 72, 81}
>>> g = ((x, y) for x in range(10) for y in range(x))
>>> type(g)
<class 'generator'>
>>> list(g)
[(1, 0), (2, 0), (2, 1), (3, 0), (3, 1), (3, 2), (4, 0), (4, 1), (4, 2), (4, 3),
 (5, 0), (5, 1), (5, 2), (5, 3), (5, 4), (6, 0), (6, 1), (6, 2), (6, 3), (6, 4),
 (6, 5), (7, 0), (7, 1), (7, 2), (7, 3), (7, 4), (7, 5), (7, 6), (8, 0), (8, 1),
 (8, 2), (8, 3), (8, 4), (8, 5), (8, 6), (8, 7), (9, 0), (9, 1), (9, 2), (9, 3),
 (9, 4), (9, 5), (9, 6), (9, 7), (9, 8)]
>>>
```

Summary

Using multiple input iterable for comprehensions

Similarity to nested for-loops

Nesting comprehensions

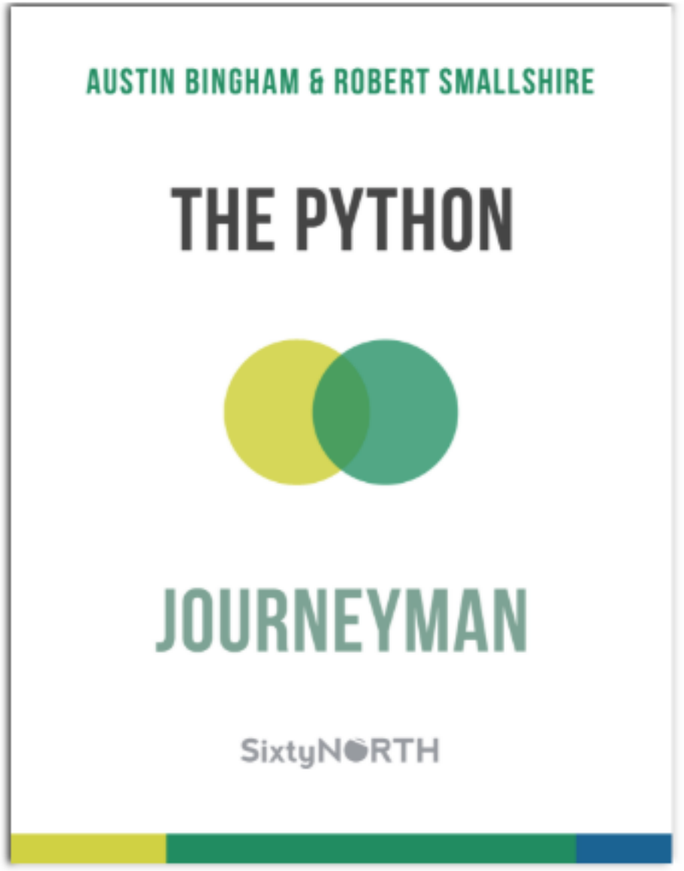
Well done!

Core Python

on

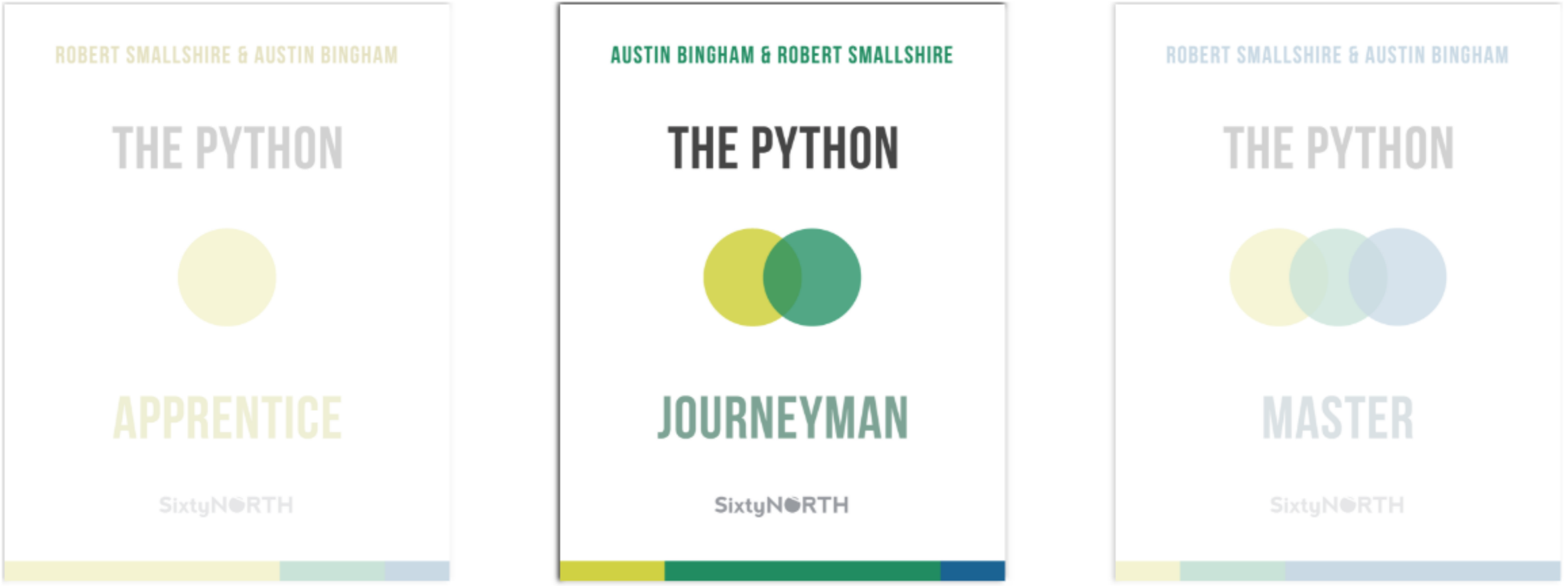
PLURALSIGHT

The Python Craftsman



leanpub.com/b/python-craftsman

The Python Journeyman



leanpub.com/python-journeyman

Happy Programming!