Other Comprehensions

Dict Comprehensions

Dictionary comprehensions:

```
>>> blocks = { num: "x" * num for num in range(5) }
>>> print(blocks)
{0: '', 1: 'x', 2: 'xx', 3: 'xxx', 4: 'xxxx'}
```

Set comprehensions

Imagine a Student class, which includes a "major" attribute.

```
>>> # A list of student majors...
... [ student.major for student in students ]
['Computer Science', 'Economics', 'Computer Science', 'Economics', 'Basket Weaving']
>>> # And a set of majors:
... { student.major for student in students }
{'Economics', 'Computer Science', 'Basket Weaving'}
>>> # You can also use the set() built-in.
... set(student.major for student in students)
{'Economics', 'Computer Science', 'Basket Weaving'}
```

P.S. The official docs for Python sets:
https://docs.python.org/3/tutorial/datastructures.html#sets

Generator Comprehensions

If you use parentheses instead of square brackets, something really interesting happens.

```
>>> items = ( x*3 for x in range(5) )
>>> type(items)
<class 'generator'>
>>> next(items)
0
>>> next(items)
3
>>> next(items)
6
```

Watch Out: The official docs call these "generator expressions". But a lot of Pythonistas call them "generator comprehensions". Because they ARE comprehensions.

Another way to generate

It turns out this...

```
COUNT = 5
items = ( x*3 for x in range(COUNT) )
```

... is EXACTLY equivalent to this:

```
def gen_items(limit):
    for x in range(limit):
        yield x*3

COUNT = 5
items = gen_items(COUNT)
```

((Pro Tip))

When passing a generator expression inline to a function, you can omit the parenthesis:

```
>>> sorted( (student.name for student in students) )
['Jones, Tina', 'Shan, Geetha', 'Simmons, Russell', 'Smith, Joe']
>>> sorted( student.name for student in students )
['Jones, Tina', 'Shan, Geetha', 'Simmons, Russell', 'Smith, Joe']
```

But not always:

```
>>> sorted( student.name for student in students, reversed=True)
  File "<stdin>", line 1
SyntaxError: Generator expression must be parenthesized if not sole argument
```

Rule of thumb: `( ( ... ) )` can be replaced with `( ... )`