

Part 16 - Generators

Part 16 - Generators

Generator Expressions

i Definition: Generator Expression

A phrase that creates a generator based on the syntax:

```
<expression> for <declarations> [as <type>] in <iterator> [if|unless  
<condition>]
```

Generator Expressions have similar syntax to the `for` loops that we have covered, and serve a similar purpose.

The best way to learn how to use Generator Expressions is by example, so here we load up a `booish` prompt.

```
$ booish
>>> List(x for x in range(5)) // simplest
Generator Expression
[0, 1, 2, 3, 4]
>>> List(x * 2 for x in range(5)) // get
double of values
[0, 2, 4, 6, 8]
>>> List(x**2 for x in range(5)) // get
square of values
[0, 1, 4, 9, 16]
>>> List(x for x in range(5) if x % 2 == 0)
// check if values are even
[0, 2, 4]
>>> List(x for x in range(10) if x % 2 == 0)
// check if values are even
[0, 2, 4, 6, 8]
>>> List(y for y in (x**2 for x in
range(10)) if y % 3 != 0) // Generator
Expression inside another
[1, 4, 16, 25, 49, 64]
```

```
>>> List(cat.Weight for cat in myKitties if
cat.Age >= 1.0).Sort()
[6.0, 6.5, 8.0, 8.5, 10.5]
>>> genex = x ** 2 for x in range(5)
generator(System.Int32)
>>> for i in genex:
...     print i
...
0
1
4
```

9

16

The cat-weight example is probably what Generator Expressions are most useful for.

You don't have to create `Lists` from them either, that's mostly for show.

generators are derived from `IEnumerable`, so you get all the niceties of the `for` loop as well.

✓ **Recommendation**

Don't overdo it with Generator Expressions. If they are causing your code to be less readable, then spread them out a little.

Generator Methods

📘 **Definition: Generator Expression**

A method that creates a generator based on stating the `yield` keyword within the method.

A Generator Method is like a regular method that can return multiple times.

Here's a Generator Method that will return exponents of 2.

Generator Method Example

```
def TestGenerator():
    i = 1
    yield i
    for x in range(10):
        i *= 2
        yield i

print List(TestGenerator())
```

Output

```
[1, 2, 4, 8, 16, 32, 64, 128, 512, 1024]
```

Generator Methods are very powerful because they keep all their local variables in memory after a yield. This can allow for certain programming techniques not found in some other languages.

Generators are very powerful and useful.

Exercises

1. Create a Generator that will destroy mankind.

Go on to [Part 17 - Macros](#)