List comprehensions and generators

GC3: Grid Computing Competence Center,
University of Zurich

Sep. 28, 2012

An easy exercise

A *dotfile* is a file whose name starts with a dot character "`.`".

How can you list all dotfiles in a given directory?

(Recall that the Python library call for listing the entries in a directory is `os.listdir()`)

A very basic solution

Use a **`for`** loop to accumulate the results into a list:

```
dotfiles = [ ]
for entry in os.listdir(path):
  if entry.startswith('.'):
    dotfiles.append(entry)
```

List comprehensions, I

Python has a better and more compact syntax for *filtering* elements of a list and/or *applying* a function to them:

```
dotfiles = [ entry for entry in dotfiles
             if entry.startswith('.') ]
```

This is called a *list comprehension*.

List comprehensions, II

The general syntax of a list comprehension is:

[*expr* **for** *var* **in** *iterable* **if** *condition* **]**

where:

expr	is any Python expression;
iterable	is a (generalized) sequence;
condition	is a boolean expression, depending on *var*;
var	is a variable that will be bound in turn to each item in *iterable* which satisfies *condition*.

The '**if** *condition*' part is optional.

Generator expressions

List comprehensions are a special case of *generator expressions*:

```
( expr for var in iterable if condition )
```

A generator expression is a valid iterable and can be used to initialize tuples, sets, dicts, etc.:

```
# the set of square numbers < 100
squares = set(n*n for n in range(10))
```

Generator expressions are valid *expression*, so they can be nested:

```
# cartesian product of sets A and B
C = set( (a,b) for a in A for b in B )
```

Generators

Generator expressions are a special case of *generators*.

A generator is like a function, except it uses **`yield`** instead of **`return`**:

```
def squares():
  n = 0
  while True:
    yield n*n
    n += 1
```

At each iteration, execution resumes with the statement logically following **`yield`** in the generator's execution flow.

There can be multiple **`yield`** statements in a generator.

Reference: http://wiki.python.org/moin/Generators