Lecture #5: Generators, Comprehensions, and More!

Presented by Jamal Bouajjaj

2023–10–02

For University of New Haven's Fall 2023 CSCIxx51 Course

Itterable

Python Iterable

An iterable is an object that can be iterated over, either manully or with the for loop (and other methods like list compression).

For example, lists are iterable.

Iterable

On a technical level, an iterable must return an iterator object with `__iter__()` or `__getitem__()`

Iterator

An iterator is an object with the `__next__()` method, which returns the next item in the iterator.

An iterator raises the StopIteration exception when there is nothing next if the next function is called to it.

Generators

Generators

An generator is an iterator that YOU can create. Every generator is an iterator, but not every iterator is a generator.

See PEP 255 for more details.

Generator Function

To make a function an iterator, you need the yield keyword. It simply returns a value when the function gets called, and "resumes" when `next()` is called on the iterator. Yield rememeber's the function state unlike return. This happens until the end of a function

To force exit (i.e raise a StopIteration exception), simply exit out of the function.

```
# from https://wiki.python.org/moin/Generators
def firstn(n):
    num = 0
    while num < n:
        yield num
        num += 1
```

Usage

A generator is useful if you know the objects will be sequentially grabbed, and storing them all will be too much memory.

An example is a database query.

range?

Is *range* a generator or not?

range?

Is *range* a generator or not?

NO, it isn't (suprisinly). It's actually it's own type, and is considered a sequence.

Comprehensions

List Comprehensions

You can generate a Python list in one line with the following format

`[ <expression> for item in list if <conditional> ]`

```
a = [i for i in range(5)]
a = [i*5 for i in range(5) if i != 2]
a = [i+5 for i in a]
```

Set Comprehensions

Sets also have compression!

```
a = {i for i in range(5)}
```

Dictionary Comprehensions

Same applies to a dictionary

```
a = {f"F{i:d}": i*5 for i in range(5)}
```

And More!

Useful Itterable Functions

The next slides will go over two useful itterable functions

map

```
map(function, iterable, *iterables)
```

Return an iterator that applies function to every item of iterable, yielding the results. If additional iterables arguments are passed, function must take that many arguments and is applied to the items from all iterables in parallel. With multiple iterables, the iterator stops when the shortest iterable is exhausted. For cases where the function inputs are already arranged into argument tuples, see itertools.starmap().

map, example

```
def f(a):
  return 2**a

b = map(f, range(10))
for i in b:
  print(i)
```

zip

```
zip(*iterables, strict=False)
```

Iterate over several iterables in parallel, producing tuples with an item from each one.

zip, example

```
for item in zip([1, 2, 3], ['sugar', 'spice', 'everything
↪  nice']):
    print(item)
```

End

The end