Sequences, Generators

Fall 2016, Lecture 9, 10/21/16
Sequence definition

A sequence has a finite length. An empty sequence has length 0. A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0 for the first element.

So, a sequence is any data type that we can index into and get the length of. Examples?
Sequences we’ve seen

We’ve been working with sequences all semester!

Examples:
  Lists
  Tuples
  Strings

Note: a dictionary is NOT a sequence (can’t retrieve elements by index, is unordered)
Iterables

Any object that you can use a for loop over

All sequences are iterable, but not at all iterables are sequences

Examples of iterables:

- Lists
- Strings
- Tuples
- Dictionaries
Iterables

Functions that return special data types
  Range
  Zip
  Map

Are these data types sequences? Are they iterables?
Range

```python
>>> x = range(10)
>>> x
range(0, 10)
>>> len(x)  # We can get the length
10
>>> x[5]  # We can index
5
```
Map

```python
>>> y = map(lambda x: x**2, [1, 2, 3])
```

```python
>>> y
<map object at 0x101a3cb38>
```

```python
>>> len(y)  # We can’t get length
Error!
```

```python
>>> y[0]  # We can’t index
```
>>> z = zip([1, 2, 3], ["hello", "world", "cs88"])
>>> z
<zip object at 0x101c022c8>
>>> len(z)  # We can’t get length
Error!
>>> z[0]  # We can’t index
Error!
Functions that return iterables

Map, zip, and range return some sort of iterable

Other functions that return similar iterable data types:
  filter()
  reversed()

Try calling len() on and indexing into values returned by these/other functions on your own time
Viewing the value in the iterable

How do we view the values inside these iterables?

For range, we were able to index, but could not see all values at once

For map and zip we couldn’t even index!

We could iterate over the iterable using a for loop

We can call list() or tuple() on the iterable
What’s the point?

All of the values are never stored in memory at once.

**Each value is computed on demand - a concept called lazy evaluation**

After we are done using a value, it’s no longer stored on the computer.

If we want to save a value, we need to either bind it to a variable or loop through iterable again.

Big implications for big data! Allows us to work with huge amounts of data than can’t fit on the computer at once.
Data types of infinite length

We defined a sequence to be finite length

Can we have data types of infinite length? Or would space be an issue?

We can use lazy evaluation

How exactly would we do this? → Using a generator function

What if we tried to create an infinite length list? Computer won’t be happy.
Generator functions

Generator functions use iteration (for loops, while loops) and the `yield` keyword.

Generators functions have no return statement, but they don’t return None.

They implicitly return a generator data type.
Generator functions

Generators (the data type returned from a generator function) are iterables that compute values lazily.

Generators can also be of infinite length.
Generator function: all_ones()

>>> def all_ones():
...     for i in range(3):
...         yield 1
...
>>> all_ones
<function all_ones at 0x101c00158>
>>> x = all_ones()
>>> x
<generator object all_ones at 0x101a37ea0>
Infinite generator function: `naturals()`

```python
def naturals():
    i = 1
    while True:
        yield i
        i += 1

>>> for elem in naturals():
    ...     print(elem)
```

(keeps going, never ends)
Generator function: `all_pairs()`

We want to enumerate pairs of elements in a sequence, `s`.

How would you implement a generator function that outputs these pairs?

```python
>>> list(all_pairs([1, 2, 3]))
[(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3)]
```
Generator function: all_pairs()

We can do this vary naturally using a generator function

```python
>>> def all_pairs(s):
    for item1 in s:
        for item2 in s:
            yield (item1, item2)

>>> list(all_pairs([1, 2, 3]))
[(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3)]
```
Space and time complexity

For a list of $n$ values

- Time do we need to iterate over all values? $O(n)$ Linear
- Space do we need to iterate over all values? $O(n)$

For an iterable of $n$ values (assuming lazy evaluation)

- Time to iterate over all values? $O(1)$ Constant!