We learn how to generate values on demand.

1. Questions?

2. A generator is an object that constructs a (possibly infinite) stream of values on demand.

(a) Whenever we write a function that mentions the yield keyword, the result of the function, when called, is a generator.

```python
def countTo(n):
    i = 1
    while i <= n:
        yield i
        i += 1
```

(b) The generator object, `g`, can be asked to compute and return the next value in the sequence by calling `next(g)`. This causes the generator to execute the function until a value is returned with yield:

```python
>>> g = countTo(3)
>>> print(next(g))
1
>>> print(next(g))
2
>>> print(next(g))
3
>>> print(next(g))
Traceback (most recent call last):
  File ''<stdin>'', line 1, in <module>
StopIteration
```

If you call `next` to get a value from a generator that has run dry, it raises a `StopIteration` exception.

(c) This exception could be caught with a try-except statement, but a more efficient mechanism is to use a for loop:

```python
>>> for v in countTo(10):
>>>     print(v)
1
2
3
```

(d) Generates have the potential to generate an infinite number of values:
def count(start = 0, step = 1):
    i = start
    while True:
        yield i
        i += step

(e) How would you generate all the Fibonacci numbers? Assume the first two are 1.
(f) How would you generate all prime numbers?
(g) How would you generate a list of all the words that appear in a list of strings?
(h) How would you convert uniq (or uniqCount) from the faculty trivia lab so that it worked on infinite lists?

*