Week 11 - Lecture 21

István Albert
BMB, Bioinformatics
Infinite datasets

• Large dataset → the meaning of the word LARGE changes in time

• What is the limiting factor? The data processing or data representation?

• Infinite datasets → stream that gets processed
Iterators

• Iterators are objects that have a method called `next()` that returns the next element

• We don’t usually need to call the `next()` method ourselves. It gets automatically called while iterating
Get the iterator with the `iter()` function

```python
letters = "ABC"

# obtain the iterator from the string
it = iter(letters)
print(type(it))

# every iterator has next() method
print(it.next())
print(it.next())
```

Command Output

```
<type 'iterator'>
A
B
```
Loops use the iterators

```python
# explicit loops, map, filter
# actually iterates behind the scenes
letters = "AB"
for c in letters:
    print c

# is equivalent to
for c in iter(letters):
    print c
```
Iterators can be moved forward

```python
# we can advance the iterator ourselves

letters = "ABCDEF"G"
it = iter(letters)

for c in it:
    if c == 'C':
        it.next()
        it.next()
    else:
        print c

```

```
A
B
F
G
```
File streams are iterators!

```python
stream = open( 'DRR03.fastq' )
print stream.next()
```
Advancing the file stream

```python
stream = open('DRR03.fastq')

for record in stream:
    print(record)
    print(stream.next())
    break
```

```
@DRR000003.1 312RGAAAX:1:1:0:831

NGGCGAGGTGCACAAGCTCCTACCCCTGCTGCCTACC
```
Iterators can run out

```python
stream = open('DRR03.fastq')

print 'Start'

for rec in stream:
    pass  # do nothing

for rec in stream:
    print rec

print 'End'
```

```
Start
End
```
Multiple iterators on the same data

```python
s1 = open('DRR03.fastq')
s2 = open('DRR03.fastq')
print s1.next()
print s2.next()
```

Command Output:
```
@DRR000003.1 312RGAAAXX:1:1:0:831

@DRR000003.1 312RGAAAXX:1:1:0:831
```
Unwinding the iterator

If the data easily fits into memory it is convenient to unwind.

```python
# open stream iterator
stream = open('DRR03.fastq')

# go over each element of the stream
# and add it to a list
data = list(stream)

print len(data)
```
Operations that unwind an iterator

Operations that unwind the iterator:

• List, map, filter, dictionary creation, sort

Operations that do not unwind the iterator:

• for loop, reversed, iterator versions of map and filter (next lecture)
Infinite iterators

• Can be merged with finite data to produce certain types of behaviors → counting, cycling, repeating, chaining results

• Don’t unwind a an infinite generator → it has no end

• Many of the solutions are nicer and simpler with an iterator
```python
from itertools import count

vals = count(100)

print(type(vals))
print(vals.next())
print(vals.next())
```

Output:
```
<type 'itertools.count'>
100
101
```
```python
from itertools import repeat

# if you don't pass the last parameter (5)
# the repeat will become infinite!
text = repeat('Yes!', 5)

for word in text:
    print(word)
```

```
Yes!
Yes!
Yes!
Yes!
Yes!
Yes!
```
from itertools import cycle

# cycle is infinite! so a limiter is required
status = cycle(['On', 'Off'])
data = 'ABCDE'

for elem in zip(data, status):
    print(elem)
“silent killer” – implicit infinite loop

```python
from itertools import count, cycle, repeat

# this only repeats twice
print(list(zip(count(), repeat('HAHA', 2), cycle('ABC'))))

# this is infinite
#print(list(zip(count(), repeat('HAHA'), cycle('ABC'))))
```

```
[(0, 'HAHA', 'A'), (1, 'HAHA', 'B')]
```
```python
from itertools import count, cycle, repeat, chain

gimme_an_A = repeat('A', 2)

# chain concatenate iterators
for row in chain(gimme_an_A, "XY", range(3)):
    print(row)
```
Iterator recap

• Use minimal memory regardless of the data size
• Can be run forward only, can’t go back.
• Getting a value moves the iterator to next element
• Single use only at the end they run out

BUT

• You may usually run two iterators on the same underlying data structure
Homework 21

• Solve the problems on the next pages with iterators

• Provide either the code or the output

• If it is output then you should try to determine what the output is **without running** the code!
Problem 1

```python
from itertools import count, cycle, repeat, chain

numbers = count()
values = range(1, 100, 25)

for row in zip(numbers, values):
    print(row)
```
Problem 2

```python
from itertools import count, cycle, repeat, chain

numbers = count()
values = range(1, 100, 25)

for row in zip(numbers, numbers, values):
    print(row)
```
from itertools import count, cycle, repeat, chain

for message in zip(question, answer):
    print(message)
```python
index = count(100)
data = range(4)
color = cycle([ 'Red', 'Green', 'Blue' ])

for row in zip(index, data, color, color, color):
    print row
```
Problem 5

```python
from itertools import count, cycle, repeat, chain

a = repeat('A', 1)
b = repeat('B', 2)
c = repeat('C', 1)

abc = chain(a, b, c)

for row in zip(count(100), abc):
    print(row)
```
Problem 6

```python
from itertools import count, cycle, repeat, chain

a = repeat('A', 1)
b = repeat('B', 2)
c = chain(a, b)
d = cycle(c)

for row in zip("abcde", d):
    print(row)
```
which iterator stops the loop?
Week 11 - Lecture 22

István Albert

BMB, Bioinformatics
Compute the sum of N numbers

```python
import time

# set a smaller value first and see
# how much memory it uses
# the list below will contain 50
# million elements
N = 5*10**7
data = range(N)

print "Type", type(data)
print "Sum", sum(data)
print "Size", len(data)

time.sleep(15)
```

Uses almost 1 Gb
Same result with `xrange`

```python
import time

# set a smaller value first and see
# how much memory it uses
# the list below will contain 50 million elements
N = 5*10**7
data = xrange(N)

print "Type", type(data)
print "Sum", sum(data)
print "Size", len(data)
time.sleep(15)
```

Memory use is not noticeable

```
Type <type 'xrange'>
Sum 12499999975000000
Size 50000000
```
range vs xrange

range(10) → list( xrange(10) )

The earliest iterators that was created before the iterator formalism was developed

Unlike most iterators xrange knows how long it is → len(xrange(10))
Naming conventions for iterator tools

• Usually they are labeled with * \(\Rightarrow\) `imap`, `ifilter`, `izip` etc

• All iterators are imported from the `itertools` module

• One exception: `xrange`

• In Python 3.0 `everything` is an iterator!
Reminder on map and filter

```python
def condition(x):
    print 'Condition for %s' % x
    return True

def format(x):
    print 'Formatting %s' % x
    return x

data = [1, 2, 3]
step1 = filter(condition, data)
step2 = map(format, step1)
step2 = list(step2)
```

Command Output:

```
Condition for 1
Condition for 2
Condition for 3
Formatting 1
Formatting 2
```

Iterator imap and ifilter

```python
from itertools import *

def condition(x):
    print 'Condition for %s' % x
    return True

def format(x):
    print 'Formatting %s' % x
    return x

data = [1, 2, 3]
step1 = ifilter(condition, data)
step2 = imap(format, step1)
step2 = list(step2)
```

Compare order to previous slide
As soon as a data satisfies the condition it will passed to the next function.

```python
from itertools import *
def condition(x):
    print 'Condition for %s' % x
    return x > 2

def format(x):
    print 'Formatting %s' % x
    return x

data = [1, 2, 3, 4]
step1 = ifilter(condition, data)
step2 = imap(format, step1)
step2 = list(step2)
```

```
Condition for 1
Condition for 2
Condition for 3
Formatting 3
Condition for 4
```
Slicing iterators

```python
from itertools import count, islice

N = 10**2
data1 = range(N)

# default method of slicing
print(data1[10:15])

# this is infinite iterator
# that we will slice to limit
counter = count()
result = islice(counter, 10, 15)

# unwind the iterator
print(list(result))
```

```
[10, 11, 12, 13, 14]
[10, 11, 12, 13, 14]
```
from itertools import *

stream = open('DRR03.fastq')

# take first three lines
stream = islice(stream, 4)

for line in stream:
    print line.strip()
Merge generators with streams

```python
from itertools import *

# how many lines does my file have?
stream = open('DRR03.fastq')

lc = count()
for row in izip(lc, stream):
    pass  # do nothing

print lc.next()
```

9997
Finding consecutive objects in an iterator

```python
from itertools import *

data = "AAABCCCA"

# finding consecutive objects
# in an iterable
for key, value in groupby(data):
    print(key, list(value))
```

```
A ['A', 'A', 'A']
B ['B']
C ['C', 'C', 'C']
A ['A']
```
Sort if you want all identical objects found

```python
from itertools import *

data = "AAABCCCA"
data = sorted(data)

# finding consecutive objects!
for key, value in groupby(data):
    print(key, list(value))
```

```
A ['A', 'A', 'A', 'A']
B ['B']
C ['C', 'C', 'C']
```
Equivalence

map and filter are iterators that are already unwound

map(something) \rightarrow list(imap(something))

filter(something) \rightarrow list(ifilter(something))
Homework 22

• Run the examples from the class
• Create an iterator version file reader that reads off only the sequences from the DRR03.fastq file → 2\textsuperscript{nd}, 6\textsuperscript{th}, 10\textsuperscript{th} ... lines and writes them to a file

Possible solution:

1. Merge another iterator with your stream that labels every 4\textsuperscript{th} line with True (starting with the second) → for example \texttt{cycle}!
2. Filter on the cycled element to keep the valid lines
3. Create a filter that extracts the sequence and \texttt{imap}-it onto the valid element
4. Write the element to a file