CS 61A Discussion 5

Iterators/Generators and Midterm Review

Albert Xu

Kaavya Shah

Slides: albertxu.xyz/teaching/cs61a/

Announcements

- Project 2, Typing Test.
  - Entire project is due Tuesday, July 23rd at 11:59 PM PDT. You may complete all phases with *one* partner.
  - Can get 1 point of extra credit by submitting by Monday, July 22nd at 11:59 PM PDT and filling out a feedback survey
  - *Recommendation: do not work on phase 3 until after the midterm*
- Midterm on Thursday, July 18th at 6-9 PM PDT
  - Practice Midterm worth 1 EC point. See Piazza @646
  - Midterm Study Guide. See Piazza @640
  - **Important Post on Piazza** See @673
- **W61A Students need to fill out @711 by 7/17 9:59 PM PDT or you will not be allowed to take the midterm**
  - This is not the same form as we previously posted

Iterators, Iterables

oh my

iterable

iterator

Iterators, Iterables

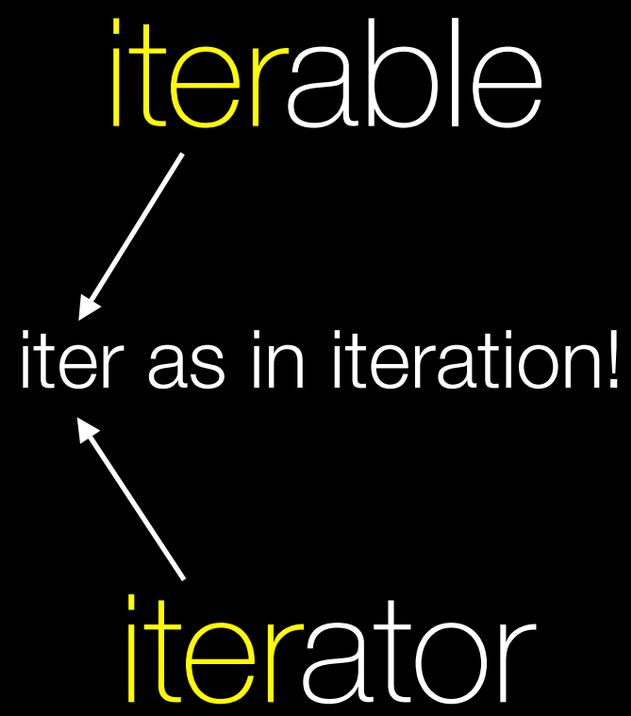
oh my

iterable

iterator

Iterators, Iterables

oh my



Iterators, Iterables

oh my

iterable

iter as in iteration!

iterator

both iterators and iterables implement the `__iter__` method, meaning that given an instance **`x`** of an iterator or iterable, calling **`iter(x)`** will give you a new iterator over **`x`**!

Iterators, Iterables

oh my

iterable

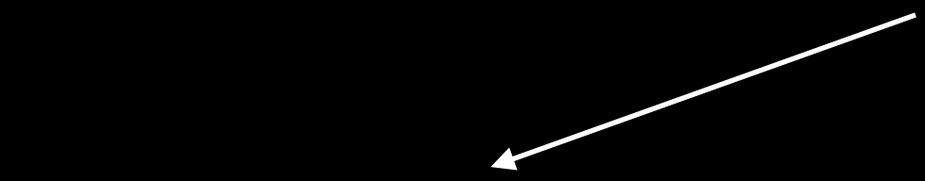
implying that it is ABLE to be iterated over!

iterator

Iterators, Iterables

oh my

iterable



implying that it is ABLE to be iterated over!

an iterable is any object, sequence or not, that can be iterated over! Examples include lists, tuples, sets, strings, and dictionaries!

iterator

Iterators, Iterables

oh my

iterable

implying that it is ABLE to be iterated over!

an iterable is any object, sequence or not, that can be iterated over! Examples include lists, tuples, sets, strings, and dictionaries!

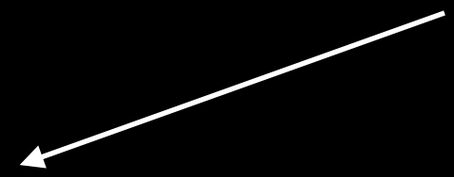
iterator

this is the thing DOING the iteration!

Iterators, Iterables

oh my

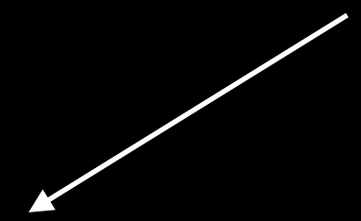
iterable



implying that it is ABLE to be iterated over!

an iterable is any object, sequence or not, that can be iterated over! Examples include lists, tuples, sets, strings, and dictionaries!

iterator



this is the thing DOING the iteration!

an iterator is an object that lets you get the next element of a sequence repeatedly until no more elements exist.

A Good Comparison

Iterable

An object whose values you can go through one at a time.

Examples

```
[1, 5, 2, 3]
{“name”: “John”, “age”: 5}
range(2, 10)
```

Things you can do:

```
for item in <iterable>:
    ...

iter(<iterable>)
```

Iterator

An object that gives you values one at a time, when you ask for them.

Things you can do:

```
next(<iterator>)
list(<iterator>)
anything with <iterable>
```

Generator

A type of iterator, created using a generator function.

*borrowed from Kevin T. Li

WWPD, Iter*

```
>>> a = [1, 2, 3]
>>> fun_iter = iter(a)
```

WWPD, Iter*

```
>>> a = [1, 2, 3]
>>> fun_iter = iter(a)
>>> next(fun_iter)
```

WWPD, Iter*

```
>>> a = [1, 2, 3]
>>> fun_iter = iter(a)
>>> next(fun_iter)
1
```

WWPD, Iter*

```
>>> a = [1, 2, 3]
>>> fun_iter = iter(a)
>>> next(fun_iter)
1
>>> next(fun_iter)
```

WWPD, Iter*

```
>>> a = [1, 2, 3]
>>> fun_iter = iter(a)
>>> next(fun_iter)
1
>>> next(fun_iter)
2
```

WWPD, Iter*

```
>>> a = [1, 2, 3]
>>> fun_iter = iter(a)
>>> next(fun_iter)
1
>>> next(fun_iter)
2
>>> unfun_iter = iter(fun_iter)
```

WWPD, Iter*

```
>>> a = [1, 2, 3]
>>> fun_iter = iter(a)
>>> next(fun_iter)
1
>>> next(fun_iter)
2
>>> unfun_iter = iter(fun_iter)
>>> funner_iter = iter(a)
```

WWPD, Iter*

```
>>> a = [1, 2, 3]
>>> fun_iter = iter(a)
>>> next(fun_iter)
1
>>> next(fun_iter)
2
>>> unfun_iter = iter(fun_iter)
>>> funner_iter = iter(a)
>>> next(unfun_iter)
```

WWPD, Iter*

```
>>> a = [1, 2, 3]
>>> fun_iter = iter(a)
>>> next(fun_iter)
1
>>> next(fun_iter)
2
>>> unfun_iter = iter(fun_iter)
>>> funner_iter = iter(a)
>>> next(unfun_iter)
3
```

WWPD, Iter*

```
>>> a = [1, 2, 3]
>>> fun_iter = iter(a)
>>> next(fun_iter)
1
>>> next(fun_iter)
2
>>> unfun_iter = iter(fun_iter)
>>> funner_iter = iter(a)
>>> next(unfun_iter)
3
>>> next(fun_iter)
```

WWPD, Iter*

```
>>> a = [1, 2, 3]
>>> fun_iter = iter(a)
>>> next(fun_iter)
1
>>> next(fun_iter)
2
>>> unfun_iter = iter(fun_iter)
>>> funner_iter = iter(a)
>>> next(unfun_iter)
3
>>> next(fun_iter)
StopIteration
```

WWPD, Iter*

```
>>> a = [1, 2, 3]
>>> fun_iter = iter(a)
>>> next(fun_iter)
1
>>> next(fun_iter)
2
>>> unfun_iter = iter(fun_iter)
>>> funner_iter = iter(a)
>>> next(unfun_iter)
3
>>> next(fun_iter)
StopIteration
>>> next(unfun_iter)
```

WWPD, Iter*

```
>>> a = [1, 2, 3]
>>> fun_iter = iter(a)
>>> next(fun_iter)
1
>>> next(fun_iter)
2
>>> unfun_iter = iter(fun_iter)
>>> funner_iter = iter(a)
>>> next(unfun_iter)
3
>>> next(fun_iter)
StopIteration
>>> next(unfun_iter)
StopIteration
```

WWPD, Iter*

```
>>> a = [1, 2, 3]
>>> fun_iter = iter(a)
>>> next(fun_iter)
1
>>> next(fun_iter)
2
>>> unfun_iter = iter(fun_iter)
>>> funner_iter = iter(a)
>>> next(unfun_iter)
3
>>> next(fun_iter)
StopIteration
>>> next(unfun_iter)
StopIteration
>>> next(funner_iter)
```

WWPD, Iter*

```
>>> a = [1, 2, 3]
>>> fun_iter = iter(a)
>>> next(fun_iter)
1
>>> next(fun_iter)
2
>>> unfun_iter = iter(fun_iter)
>>> funner_iter = iter(a)
>>> next(unfun_iter)
3
>>> next(fun_iter)
StopIteration
>>> next(unfun_iter)
StopIteration
>>> next(funner_iter)
1
```

WWPD, Iter*

```
>>> a = [1, 2, 3]
>>> fun_iter = iter(a)
>>> next(fun_iter)
1
>>> next(fun_iter)
2
>>> unfun_iter = iter(fun_iter)
>>> funner_iter = iter(a)
>>> next(unfun_iter)
3
>>> next(fun_iter)
StopIteration
>>> next(unfun_iter)
StopIteration
>>> next(funner_iter)
1
>>> a
```

WWPD, Iter*

```
>>> a = [1, 2, 3]
>>> fun_iter = iter(a)
>>> next(fun_iter)
1
>>> next(fun_iter)
2
>>> unfun_iter = iter(fun_iter)
>>> funner_iter = iter(a)
>>> next(unfun_iter)
3
>>> next(fun_iter)
StopIteration
>>> next(unfun_iter)
StopIteration
>>> next(funner_iter)
1
>>> a
[1, 2, 3]
```

Generators

how do you write your own iterators?

Generators

how do you write your own iterators?

Using a list

Generators

how do you write your own iterators?

Using a list

```
>>> fibs = [1, 1, 2, 3, 5, 8]
>>> fib_iter = iter(fibs)
```

Generators

how do you write your own iterators?

Using a list

```
>>> fibs = [1, 1, 2, 3, 5, 8]
>>> fib_iter = iter(fibs)
```

Two issues:

1) I have to calculate out values of fib myself
2) I can only make a finite iterator

Generators

how do you write your own iterators?

Using a list

```
>>> fibs = [1, 1, 2, 3, 5, 8]
>>> fib_iter = iter(fibs)
```

Two issues:

1) I have to calculate out values of fib myself
2) I can only make a finite iterator

Using a generator

Generators

how do you write your own iterators?

Using a list

```
>>> fibs = [1, 1, 2, 3, 5, 8]
>>> fib_iter = iter(fibs)
```

Two issues:

1) I have to calculate out values of fib myself
2) I can only make a finite iterator

Using a generator

```
>>> def fibs():
        prev = 0
        current = 1
        while True:
            yield current
            current = current + prev
            prev = current
```

Generators

how do you write your own iterators?

Using a list

```
>>> fibs = [1, 1, 2, 3, 5, 8]
>>> fib_iter = iter(fibs)
```

Two issues:

1) I have to calculate out values of fib myself
2) I can only make a finite iterator

Using a generator

```
>>> def fibs():
        prev = 0
        current = 1
        while True:
            yield current
            current = current + prev
            prev = current
>>> fib_iter = fibs()
```

Generators

how do you write your own iterators?

Using a list

```
>>> fibs = [1, 1, 2, 3, 5, 8]
>>> fib_iter = iter(fibs)
```

Two issues:

1) I have to calculate out values of fib myself
2) I can only make a finite iterator

Using a generator

generator function

```
>>> def fibs():
        prev = 0
        current = 1
        while True:
            yield current
            current = current + prev
            prev = current
>>> fib_iter = fibs()
```

Generators

how do you write your own iterators?

Using a list

```
>>> fibs = [1, 1, 2, 3, 5, 8]
>>> fib_iter = iter(fibs)
```

Two issues:

1) I have to calculate out values of fib myself
2) I can only make a finite iterator

Using a generator

```
>>> def fibs():
        prev = 0
        current = 1
        while True:
            yield current
            current = current + prev
            prev = current
>>> fib_iter = fibs()
```

generator function

generator object (generator)

a final note on equality

if the second part doesn't make sense, it's okay.

- a is b
  - Object equality
  - This tells you whether two names are pointing to the same thing (consult environment diagram)

a final note on equality

if the second part doesn't make sense, it's okay.

- a is b
  - Object equality
  - This tells you whether two names are pointing to the same thing (consult environment diagram)
- a == b
  - Value equality
  - This tells you whether two numbers or two lists contain identical values.

a final note on equality

if the second part doesn't make sense, it's okay.

- a is b
  - Object equality
  - This tells you whether two names are pointing to the same thing (consult environment diagram)
- a == b
  - Value equality
  - This tells you whether two numbers or two lists contain identical values.

But what happens if you use is for checking number equality?
Weird things happen.

a final note on equality

if the second part doesn't make sense, it's okay.

- a is b
  - Object equality
  - This tells you whether two names are pointing to the same thing (consult environment diagram)
- a == b
  - Value equality
  - This tells you whether two numbers or two lists contain identical values.

But what happens if you use is for checking number equality?
Weird things happen.

```
>>> a = 256
>>> b = 256
>>> a is b
True
```

a final note on equality

if the second part doesn't make sense, it's okay.

- a is b
  - Object equality
  - This tells you whether two names are pointing to the same thing (consult environment diagram)
- a == b
  - Value equality
  - This tells you whether two numbers or two lists contain identical values.

But what happens if you use is for checking number equality?
Weird things happen.

```
>>> a = 256
>>> b = 256
>>> a is b
True
```

```
>>> a = 257
>>> b = 257
>>> a is b
False
```

Thanks for coming.

Have a great rest of your week! :)

Attendance: links.cs61a.org/albert-disc
Slides: albertxu.xyz/teaching/cs61a/