Find Your Fun Magnets

- “The fact that True Fun is an emotional experience means that, while activities can help generate fun, activities themselves are not fun.”

- “With that said, each of us has certain activities – and, for that matter, people, and settings – that are much more likely than others to trigger or enhance our feelings of playfulness, connection, and flow, and thus more likely to attract True Fun.”

- “I call these “fun magnets,” and each of us has a collection that’s unique to us. If you want to up your chances of experiencing True Fun, you should seek out and prioritize your fun magnets as often as you can.”

  -- The Power of Fun: How to Feel Alive Again by Price

Topics

- Iterators
- Binary search
- How to resubmit work in this course

Your to-dos

1. Better not come to class on Friday!
2. Read before Mon: Bailey, Ch 12-12.5.
3. Lab 7 (partner lab), due Tuesday 11/1 by 10pm.
Announcements

• CS Colloquium, Fri @ 2:35 in Wege Auditorium
  Pre-registration info session + cookies

Announcements

Please consider being a TA next semester (especially for this class!)
Applications due Friday, Oct 28.
https://csci.williams.edu/tatutor-application/

Iterators

What do the following have in common?

```java
double[] a
// _ initialize a_
double sum = 0.0;
for (int i = 0; i < a.length; i++) {
    sum += a[i];
}
```

```java
List<Double> ls = new SinglyLinkedList<>()
// _ initialize ls_
double sum = 0.0;
for (int i = 0; i < ls.size(); i++)
    sum += ls.get(i);
```

```java
Stack<Double> s = new StackVector<>()
// _ initialize s_
double sum = 0.0;
while (!s.isEmpty())
    sum += s.pop();
```
Iteration is the repetition of a process in order to generate a (possibly unbounded) sequence of outcomes. Each repetition of the process is a single iteration, and the outcome of each iteration is then the starting point of the next iteration.

declare double[] a
initialize a = 0.0;
for (int i = 0; i < a.length; i++) {
    sum += a[i];
}

iterating
i
sum
0
1
2
3
100
101
102
103
0
1
2
3
100
101
102
103
0
1
2
3
100
101
102
103
0
1
2
3
100
101
102
103
0
1
2
3
100
101
102
103
0
1
2
3

Each program iterates

double[] a
// ... initialize a ...
double sum = 0.0;
for (int i = 0; i < a.length; i++) {
    sum += a[i];
}

100 101 102 103
0 1 2 3

i 1 sum 201

Each program iterates

double[] a
// ... initialize a ...
double sum = 0.0;
for (int i = 0; i < a.length; i++) {
    sum += a[i];
}

100 101 102 103
0 1 2 3

i 2 sum 201

Each program iterates

double[] a
// ... initialize a ...
double sum = 0.0;
for (int i = 0; i < a.length; i++) {
    sum += a[i];
}

100 101 102 103
0 1 2 3

i 2 sum 303

Each program iterates

double[] a
// ... initialize a ...
double sum = 0.0;
for (int i = 0; i < a.length; i++) {
    sum += a[i];
}

100 101 102 103
0 1 2 3

i 3 sum 303
Each program iterates

double[] a
// _ initialize a _
double sum = 0.0;
for (int i = 0; i < a.length; i++) {
    sum += a[i];
}

100 101 102 103
0 1 2 3

i 3 sum 406

Each program iterates

List<Double> ls = new SinglyLinkedList<>();
// _ initialize ls _
double sum = 0.0;
for (int i = 0; i < ls.size(); i++) {
    sum += ls.get(i);
}

100 101 102 Ø
0 1 2 3

i 0 sum 0

Each program iterates

double[] a
// _ initialize a _
double sum = 0.0;
for (int i = 0; i < a.length; i++) {
    sum += a[i];
}

100 101 102 103
0 1 2 3

i 3 sum 406

“Iteration is terminated!”

Each program iterates

List<Double> ls = new SinglyLinkedList<>();
// _ initialize ls _
double sum = 0.0;
for (int i = 0; i < ls.size(); i++) {
    sum += ls.get(i);
}

100 101 102 Ø
0 1 2 3

i 0 sum 100
Each program iterates

List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
double sum = 0.0;
for (int i = 0; i < ls.size(); i++) {
    sum += ls.get(i);
}

Each program iterates

List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
double sum = 0.0;
for (int i = 0; i < ls.size(); i++) {
    sum += ls.get(i);
}

Each program iterates

List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
double sum = 0.0;
for (int i = 0; i < ls.size(); i++) {
    sum += ls.get(i);
}

Each program iterates

List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
double sum = 0.0;
for (int i = 0; i < ls.size(); i++) {
    sum += ls.get(i);
}
Each program iterates

List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
    for (int i = 0; i < ls.size(); i++) {
        sum += ls.get(i);
    }

100  101  102  Ø

i  2  sum  201

Each program iterates

List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
    for (int i = 0; i < ls.size(); i++) {
        sum += ls.get(i);
    }

100  101  102  Ø

i  2  sum  201

Each program iterates

List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
    for (int i = 0; i < ls.size(); i++) {
        sum += ls.get(i);
    }

100  101  102  Ø

i  2  sum  303

Each program iterates

List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
    for (int i = 0; i < ls.size(); i++) {
        sum += ls.get(i);
    }

100  101  102  Ø

i  2  sum  303

"Iteration is terminated!"
Each program iterates

```java
Stack<Double> s = new StackVector<>();
// ... initialize s ...
double sum = 0.0;
while (!s.isEmpty()) {
    sum += s.pop();
}
```
Each program iterates

```java
Stack<Double> s = new StackVector<>();
// _ initialize s_
double sum = 0.0;
while (!s.isEmpty()) {
    sum += s.pop();
}
```

```
0 1 2 3
_________
| 100 |
| 00  |
| 00  |
| 00  |
```

```
sum 203
```

```
“Iteration is terminated!”
```

Each program iterates

```java
Stack<Double> s = new StackVector<>();
// _ initialize s_
double sum = 0.0;
while (!s.isEmpty()) {
    sum += s.pop();
}
```

```
0 1 2 3
_________
| 00  |
| 00  |
| 00  |
| 303 |
```

```
sum 303
```

Essentially the same algorithm!

```java
double[] a
// _ initialize a_
// double sum = 0.0;
for (int i = 0; i < a.length; i++) {
    sum += a[i];
}
```

```java
List<Double> ls = new SinglyLinkedList<>();
// _ initialize ls_
double sum = 0.0;
for (int i = 0; i < ls.size(); i++) {
    sum += ls.get(i);
}
```

```java
Stack<Double> s = new StackVector<>();
// _ initialize s_
// double sum = 0.0;
while (!s.isEmpty()) {
    sum += s.pop();
}
```

```
sum 303
```

But the code looks different.
Problems

• **Different data structures** yield **different code for same algorithm**.

• **Data hiding** potentially causes **efficiency problems**.

• **Inspecting** data structure “from the outside” can change the state of a data structure (e.g., `pop()`'ing a `Stack`).

What if I told you that you could solve all of these problems with abstraction?

Iteration abstraction to the rescue.

```java
double[] a
// _ _ initialize a _
double sum = 0.0;
for (double d : a) {
    sum += d;
}
```

```java
List<Double> ls = new SinglyLinkedList<>();
// _ _ initialize ls _
double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```java
Stack<Double> s = new StackVector<>();
// _ _ initialize s _
double sum = 0.0;
for (double d : s) {
    sum += d;
}
```

Brought to you by Iterators.

Iterators are a really good idea.

• Invented by Barbara Liskov in 1974.

• Incidentally, **abstract data types** were also invented by Barbara Liskov in 1974.

• Both debuted in the influential PL called **CLU**.

• Barbara won the **Turing Award in 2008** for this work and more.
How does “for each” work?

```java
for (int num : nums) { ... }
```

All of these data structures must implement `Iterable<T>`

```
public interface Iterable<T>
{
    Iterator<T> iterator();
}
```

(array is a special case)

What is an `Iterable<T>`?

It’s a class that returns an `Iterator<T>`.

```
public interface Iterable<T>
{
    Iterator<T> iterator();
}
```

Let’s look at `SinglyLinkedList<T>`

What’s an `Iterator<T>`???

```
public interface Iterator<E>
{
    boolean hasNext();
    E next();
    ...
}
```

It’s an object that lets you iterate through a data structure.

Importantly, `Iterators` are stateful.

Why does statefulness matter? It can save work.
Naive iteration makes $O(n)$ operation $O(n^2)$!

List<Double> ls = new SinglyLinkedList<>();
// _initialize ls _
  double sum = 0.0;
  for (int i = 0; i < ls.size(); i++) {
    sum += ls.get(i);
  }

i 0  sum 0

Naive iteration makes $O(n)$ operation $O(n^2)$!

List<Double> ls = new SinglyLinkedList<>();
// _initialize ls _
  double sum = 0.0;
  for (int i = 0; i < ls.size(); i++) {
    sum += ls.get(i);
  }

i 0  sum 100

Naive iteration makes $O(n)$ operation $O(n^2)$!

List<Double> ls = new SinglyLinkedList<>();
// _initialize ls _
  double sum = 0.0;
  for (int i = 0; i < ls.size(); i++) {
    sum += ls.get(i);
  }

i 1  sum 100

Naive iteration makes $O(n)$ operation $O(n^2)$!

List<Double> ls = new SinglyLinkedList<>();
// _initialize ls _
  double sum = 0.0;
  for (int i = 0; i < ls.size(); i++) {
    sum += ls.get(i);
  }

i 1  sum 100
Naive iteration makes $O(n)$ operation $O(n^2)$!

```java
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
double sum = 0.0;
for (int i = 0; i < ls.size(); i++) {
    sum += ls.get(i);
}
```

```
100 101 102 ∅
```

```
i 1 sum 201
```

Naive iteration makes $O(n)$ operation $O(n^2)$!

```java
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
double sum = 0.0;
for (int i = 0; i < ls.size(); i++) {
    sum += ls.get(i);
}
```

```
100 101 102 ∅
```

```
i 2 sum 201
```

Naive iteration makes $O(n)$ operation $O(n^2)$!

```java
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
double sum = 0.0;
for (int i = 0; i < ls.size(); i++) {
    sum += ls.get(i);
}
```

```
100 101 102 ∅
```

```
i 2 sum 201
```
Naive iteration makes $O(n)$ operation $O(n^2)$!

```java
List<Double> ls = new SinglyLinkedList<>();
// … initialize ls …
double sum = 0.0;
for (int i = 0; i < ls.size(); i++) {
    sum += ls.get(i);
}
```

---

How does \texttt{for} use an \texttt{Iterator<T>}?

The following code

```java
List<Integer> ls = new SinglyLinkedList<>();
// …
for (int i : ls) {
    // … work …
}
```

is the moral equivalent to

```java
List<Integer> ls = new SinglyLinkedList<>();
// …
for (Iterator<Integer> i = ls.iterator(); i.hasNext();)
    int n = i.next();
    // … work …
```

1. Get \texttt{Iterator<T>}
2. Get next element.
3. If there is a next element, go to 2.

---

Example.

```java
List<Double> ls = new SinglyLinkedList<>();
// … initialize ls …
double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```
Example.

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head
```

```
100  101  102  Ø
```

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head
```

```
100  101  102  Ø
```

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head
```

```
100
```

```
100
```

```
d  0
```

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head
```

```
100  101  102  Ø
```

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head
```

```
100  101  102  Ø
```

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head
```

```
100  101  102  Ø
```

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head
```

```
100  101  102  Ø
```

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head
```

```
100  101  102  Ø
```

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head
```

```
100  101  102  Ø
```

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head
```

```
100  101  102  Ø
```

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head
```

```
100  101  102  Ø
```

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head
```

```
100  101  102  Ø
```

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head
```

```
100  101  102  Ø
```

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head
```

```
100  101  102  Ø
```

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head
```

```
100  101  102  Ø
```

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head
```

```
100  101  102  Ø
```

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head
```

```
100  101  102  Ø
```

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head
```

```
100  101  102  Ø
```

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head
```

```
100  101  102  Ø
```

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head
```

```
100  101  102  Ø
```

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head
```

```
100  101  102  Ø
```

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head
```

```
100  101  102  Ø
```

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
    double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head
```

```
100  101  102  Ø
```
Example.

```java
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
100    101    102    Ø
```

```
head
current
```

```
sum 201
d 101
```

```
Example.
```

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
100    101    102    Ø
```

```
head
current
```

```
sum 303
d 102
```

```
Example.
```

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
100    101    102    Ø
```

```
head
current
```

```
sum 303
d 102
```

```
“Iteration is terminated!”
```

```
Example.
```

```
List<Double> ls = new SinglyLinkedList<>();
// ... initialize ls ...
```

```
Efficient searching: binary search
```

```
100    101    102    Ø
```

```
head
current
```

```
sum 303
d 102
```

```
“Iteration is terminated!”
```
Want to know whether the array contains the value 322, and if so, what its index is.

Binary search is a divide-and-conquer algorithm that solves this problem.

Binary search is fast: in the worst case, it returns an answer in $O(\log_2 n)$ steps.

Important precondition: array must be sorted.

Looking for the value 322.
Looking for the value 322.

322 = 365? no
322 < 365? yes
Binary search

Looking for the value 322.

```
<table>
<thead>
<tr>
<th>100</th>
<th>101</th>
<th>322</th>
<th>365</th>
<th>423</th>
<th>478</th>
<th>499</th>
<th>504</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
```

322 = 101? no
322 < 101? no
322 > 101? yes

Recap & Next Class

**Today:**

Iteration
Binary search

**Next class:**

Ordered structures