Announcements
Announcements

• Homework 12 due Tuesday 12/10 @ 11:59pm.
Announcements

- Homework 12 due Tuesday 12/10 @ 11:59pm.
  - All you have to do is vote on your favorite recursive art.
Announcements

• Homework 12 due Tuesday 12/10 @ 11:59pm.
  • All you have to do is vote on your favorite recursive art.
• 29 review sessions next week! Come learn about the topics that interest you the most.
Announcements

• Homework 12 due Tuesday 12/10 @ 11:59pm.
  ▪ All you have to do is vote on your favorite recursive art.
• 29 review sessions next week! Come learn about the topics that interest you the most.
  ▪ See http://inst.eecs.berkeley.edu/~cs61a/fa13/exams/final.html for the schedule.
Announcements

• Homework 12 due Tuesday 12/10 @ 11:59pm.
  • All you have to do is vote on your favorite recursive art.
• 29 review sessions next week! Come learn about the topics that interest you the most.
  • See http://inst.eecs.berkeley.edu/~cs61a/fa13/exams/final.html for the schedule.
• The final exam is on Friday 12/20 @ 11:30am in the RSF gym, emphasizing:
Announcements

• Homework 12 due Tuesday 12/10 @ 11:59pm.
  ▪ All you have to do is vote on your favorite recursive art.

• 29 review sessions next week! Come learn about the topics that interest you the most.
  ▪ See http://inst.eecs.berkeley.edu/~cs61a/fa13/exams/final.html for the schedule.

• The final exam is on Friday 12/20 @ 11:30am in the RSF gym, emphasizing:
  ▪ Higher-order functions
Announcements

• Homework 12 due Tuesday 12/10 @ 11:59pm.
  ▪ All you have to do is vote on your favorite recursive art.

• 29 review sessions next week! Come learn about the topics that interest you the most.
  ▪ See http://inst.eecs.berkeley.edu/~cs61a/fa13/exams/final.html for the schedule.

• The final exam is on Friday 12/20 @ 11:30am in the RSF gym, emphasizing:
  ▪ Higher-order functions
  ▪ Sequences (tuples, lists, recursive lists, Scheme lists)
Announcements

• Homework 12 due Tuesday 12/10 @ 11:59pm.
  ▪ All you have to do is vote on your favorite recursive art.

• 29 review sessions next week! Come learn about the topics that interest you the most.
  ▪ See http://inst.eecs.berkeley.edu/~cs61a/fa13/exams/final.html for the schedule.

• The final exam is on Friday 12/20 @ 11:30am in the RSF gym, emphasizing:
  ▪ Higher-order functions
  ▪ Sequences (tuples, lists, recursive lists, Scheme lists)
  ▪ Non-local assignment and mutation
Announcements

• Homework 12 due Tuesday 12/10 @ 11:59pm.
  ▪ All you have to do is vote on your favorite recursive art.

• 29 review sessions next week! Come learn about the topics that interest you the most.
  ▪ See [http://inst.eecs.berkeley.edu/~cs61a/fa13/exams/final.html](http://inst.eecs.berkeley.edu/~cs61a/fa13/exams/final.html) for the schedule.

• The final exam is on Friday 12/20 @ 11:30am in the RSF gym, emphasizing:
  ▪ Higher-order functions
  ▪ Sequences (tuples, lists, recursive lists, Scheme lists)
  ▪ Non-local assignment and mutation
  ▪ Object-oriented programming
Announcements

• Homework 12 due Tuesday 12/10 @ 11:59pm.
  All you have to do is vote on your favorite recursive art.

• 29 review sessions next week! Come learn about the topics that interest you the most.
  See http://inst.eecs.berkeley.edu/~cs61a/fa13/exams/final.html for the schedule.

• The final exam is on Friday 12/20 @ 11:30am in the RSF gym, emphasizing:
  Higher-order functions
  Sequences (tuples, lists, recursive lists, Scheme lists)
  Non-local assignment and mutation
  Object-oriented programming
  Recursion and recursive data
Announcements

• Homework 12 due Tuesday 12/10 @ 11:59pm.
  ▪ All you have to do is vote on your favorite recursive art.
• 29 review sessions next week! Come learn about the topics that interest you the most.
  ▪ See http://inst.eecs.berkeley.edu/~cs61a/fa13/exams/final.html for the schedule.
• The final exam is on Friday 12/20 @ 11:30am in the RSF gym, emphasizing:
  ▪ Higher-order functions
  ▪ Sequences (tuples, lists, recursive lists, Scheme lists)
  ▪ Non-local assignment and mutation
  ▪ Object-oriented programming
  ▪ Recursion and recursive data
  ▪ Iterators, generators, and streams
Implicit Sequences Example
Example: Numerical Approximations

Is \( \sqrt{51} - 4 \) < \( \pi \) ?
Example: Numerical Approximations

Is $\sqrt{51} - 4 < \pi$ ?

*No calculators/interpreters allowed!*
Example: Numerical Approximations

Is $\sqrt{51} - 4 < \pi$?

No calculators/interpreters allowed!

Let's say we have a computer that can $+, -, *, /$. How do we answer this question?
Example: Numerical Approximations

Is \( \sqrt{51} - 4 \) < \( \pi \) ?

*No calculators/interpreters allowed!*

Let's say we have a computer that can +, -, *, /. How do we answer this question?
Example: Numerical Approximations

Is $\sqrt{51} - 4 < \pi$?  

*No calculators/interpreters allowed!*

Let's say we have a computer that can $+, -, \times, /$. How do we answer this question?

Approximations of $\sqrt{51} - 4$

Approximations of $\pi$
Approximating Square Roots

Let's say we have a computer that can $+$, $-$, $\times$, $\div$. How do we answer this question?

(A) A sequence of approximations (SoA) to $y$ is an infinite sequence that converges to $y$. Implicitly define a SoA to $\sqrt{a}$.

Is $\sqrt{51} - 4 < \pi$ ?

No calculators/interpreters allowed!
Approximating Square Roots

Is $\sqrt{51} - 4 < \pi$ ?

No calculators/interpreters allowed!

Let's say we have a computer that can $+, -, *, /$. How do we answer this question?

(A) A sequence of approximations (SoA) to $y$ is an infinite sequence that converges to $y$. Implicitly define a SoA to $\sqrt{a}$.

How to compute $\text{square\_root}(a)$:

Idea: Iteratively refine a guess $x$ about the square root of $a$.

$$x = \frac{x + \frac{a}{x}}{2}$$

From lecture 6
Approximating Square Roots

Is \( \sqrt{51} - 4 < \pi \) ?

No calculators/interpreters allowed!

Let's say we have a computer that can +, -, *, /.

How do we answer this question?

(A) A sequence of approximations (SoA) to \( y \) is an infinite sequence that converges to \( y \). Implicitly define a SoA to \( \sqrt{a} \).

```python
def sqrt(a):
    x = 1
    while _______________________________
        yield __________________________
        x = ____________________________
```

How to compute square_root(a):

Idea: Iteratively refine a guess \( x \) about the square root of \( a \).

\[
x = \frac{x + \frac{a}{x}}{2}
\]

From lecture 6
Approximating Square Roots

Is $\sqrt{51} - 4 < \pi$?

No calculators/interpreters allowed!

Let's say we have a computer that can $+, -, *, /$. How do we answer this question?

(A) A sequence of approximations (SoA) to $y$ is an infinite sequence that converges to $y$. Implicitly define a SoA to $\sqrt{a}$.

```python
def sqrt(a):
    x = 1
    while ____________________________:
        yield _________________________
        x = _________________________
```

How to compute square_root(a):

Idea: Iteratively refine a guess $x$ about the square root of $a$.

$\sqrt{a} = \frac{x + \frac{a}{x}}{2}$

From lecture 6

```python
>>> for x in sqrt(2):
...    print(x)
1
1.5
1.4166666666666665
1.4142156862745097
...
Approximating Square Roots

Is \( \sqrt{51} - 4 \) < \( \pi \)?

No calculators/interpreters allowed!

Let's say we have a computer that can +, -, *, /.

(A) A sequence of approximations (SoA) to \( y \) is an infinite sequence that converges to \( y \).

Implicitly define a SoA to \( \sqrt{a} \).

```python
def sqrt(a):
    x = 1
    while __________________________:
        yield __________________________
        x = ___________________________
```

```plaintext
>>> for x in sqrt(2):
...     print(x)
1
1.5
1.4166666666666665
1.4142156862745097
...
```

How to compute square_root(a):

**Idea:** Iteratively refine a guess \( x \) about the square root of \( a \).

\[
\text{\( x = \frac{x + \frac{a}{x}}{2} \)}
\]

*From lecture 6*
Approximating Square Roots

Is $\sqrt{51} - 4 < \pi$?

No calculators/interpreters allowed!

Let's say we have a computer that can $+, -, *, /$. How do we answer this question?

(A) A sequence of approximations (SoA) to $y$ is an infinite sequence that converges to $y$.

Implicitly define a SoA to $\sqrt{a}$.

```python
def sqrt(a):
    x = 1
    while ___________________ True ______________:  # your code here
        yield ____________________
        x = ____________________

>>> for x in sqrt(2):
    ...    print(x)
1
1.5
1.4166666666666665
1.4142156862745097
...```

How to compute square_root(a):

**Idea:** Iteratively refine a guess $x$ about the square root of $a$.

\[
x = \frac{x + \frac{a}{x}}{2}
\]

*From lecture 6*
Approximating Square Roots

Is $\sqrt{51} - 4 < \pi$?

No calculators/interpreters allowed!

Let's say we have a computer that can $+, -, *, /$. How do we answer this question?

(A) A sequence of approximations (SoA) to $y$ is an infinite sequence that converges to $y$. Implicitly define a SoA to $\sqrt{a}$.

```
def sqrt(a):
    x = 1
    while _______________ True ________________:
        yield _________________ x _________________
        x = ______________________
```

How to compute square_root(a):

Idea: Iteratively refine a guess $x$ about the square root of $a$.

$$x = \frac{x + \frac{a}{x}}{2}$$

From lecture 6

```python
>>> for x in sqrt(2):
...     print(x)
1
1.5
1.4166666666666665
1.4142156862745097
...
```
Approximating Square Roots

\[
\sqrt{51} - 4 < \pi \quad ?
\]

No calculators/interpreters allowed!

Let's say we have a computer that can \(+\), \(-\), \(*\), \(/\). How do we answer this question?

(A) A sequence of approximations (SoA) to \(y\) is an infinite sequence that converges to \(y\).

Implicitly define a SoA to \(\sqrt{a}\).

```python
def sqrt(a):
    x = 1
    while True:
        yield x
        x = (x + a/x)/2
```

How to compute square_root(a):

**Idea:** Iteratively refine a guess \(x\) about the square root of \(a\).

\[
x = \frac{x + \frac{a}{x}}{2}
\]

*From lecture 6*

```python
>>> for x in sqrt(2):
...    print(x)
1
1.5
1.4166666666666665
1.4142156862745097
...
Approximating Pi

Is \(\sqrt{51} - 4\) < \(\pi\) ?

```python
def sqrt(a):
    x = 1
    while True:
        yield x
        x = (x + a/x)/2
```
Approximating Pi

Is $\sqrt{51} - 4 < \pi$?

```python
def sqrt(a):
    x = 1
    while True:
        yield x
        x = (x + a/x)/2
```

(B) Define a sequence of approximations to $\pi$. 
Approximating Pi

Is $\sqrt{51} - 4 < \pi$?

```python
def sqrt(a):
x = 1
while True:
yield x
    x = (x + a/x)/2
```

(B) Define a sequence of approximations to $\pi$.

\[
\sum_{k=1}^{\infty} \frac{8}{(4k - 3) \cdot (4k - 1)} = \pi
\]

*From lecture 4*
Approximating Pi

Is \( \sqrt{51} - 4 \) < \( \pi \) ?

(B) Define a sequence of approximations to \( \pi \).

```python
def sqrt(a):
    x = 1
    while True:
        yield x
        x = (x + a/x)/2
```

From lecture 4

\[
\sum_{k=1}^{\infty} \frac{8}{(4k - 3) \cdot (4k - 1)} = \pi
\]

>>>
```
for x in pi():
    ... print(x)
0
2.6666666666666665
2.895238095238095
2.976046176046176
3.017071817071817
3.041839618929402
3.0584027659273314
...
```
Approximating Pi

Is $\sqrt{51} - 4 < \pi$ ?

```
def sqrt(a):
    x = 1
    while True:
        yield x
        x = (x + a/x)/2
```

```
(B) Define a sequence of approximations to $\pi$.

def pi():
    ______________________________
    while True:
        yield ______________________
                        ______________________
                        ______________________
                        ______________________

>>> for x in pi():
...    print(x)
...    0
... 2.6666666666666665
... 2.895238095238095
... 2.976046176046176
... 3.017071817071817
... 3.041839618929402
... 3.0584027659273314
... ...
```

$$\sum_{k=1}^{\infty} \frac{8}{(4k - 3) \cdot (4k - 1)} = \pi$$

*From lecture 4*
Approximating Pi

Is $\sqrt{51} - 4 < \pi$ ?

(B) Define a sequence of approximations to $\pi$.

```python
def sqrt(a):
    x = 1
    while True:
        yield x
        x = (x + a/x)/2

def pi():
    #\sum_{k=1}^{\infty} \frac{8}{(4k-3)(4k-1)} = \pi

>>> for x in pi():
...     print(x)
0
2.6666666666666665
2.895238095238095
2.976046176046176
3.017071817071817
3.041839618929402
3.0584027659273314
...```
Approximating Pi

Is $\sqrt{51} - 4 < \pi$?

**def sqrt(a):**
```
x = 1
while True:
    yield x
    x = (x + a/x)/2
```

(B) Define a sequence of approximations to $\pi$.

```python
def pi():
    total, k, 0, 1
    while True:
        yield
        total = total + (4k - 3) * (4k - 1)
        k = k + 1
```

```python
>>> for x in pi():
    ...
    print(x)
0
2.6666666666666665
2.895238095238095
2.976046176046176
3.017071817071817
3.041839618929402
3.0584027659273314
```

\[ \sum_{k=1}^{\infty} \frac{8}{(4k - 3) \cdot (4k - 1)} = \pi \]

From lecture 4
Approximating Pi

Is \( \sqrt{51} - 4 \) < \( \pi \)?

def sqrt(a):
    x = 1
    while True:
        yield x
        x = (x + a/x)/2

(B) Define a sequence of approximations to \( \pi \).

def pi():
    total, k = 0, 1
    while True:
        yield total
        total = total + 8/(4*k-3)/(4*k-1)

>>> for x in pi():
...     print(x)
0
2.6666666666666665
2.895238095238095
2.976046176046176
3.017071817071817
3.041839618929402
3.0584027659273314
...
Approximating Pi

Is \( \sqrt{51} - 4 \) < \( \pi \) ?

\[
\sum_{k=1}^{\infty} \frac{8}{(4k-3) \cdot (4k-1)} = \pi
\]

From lecture 4

(B) Define a sequence of approximations to \( \pi \).

```python
def sqrt(a):
    x = 1
    while True:
        yield x
        x = (x + a/x)/2

def pi():
    total, k = 0, 1
    while True:
        yield total
        total += 8 / (4*k-3) * (4*k-1)
```

```python
>>> for x in pi():
...     print(x)
0
2.6666666666666665
2.895238095238095
2.976046176046176
3.017071817071817
3.041839618929402
3.0584027659273314
...```

Approximating Pi

Is $\sqrt{51} - 4 < \pi$?

(B) Define a sequence of approximations to $\pi$.

```python
def sqrt(a):
    x = 1
    while True:
        yield x
        x = (x + a/x)/2

def pi():
    total, k = 0, 1
    while True:
        yield total
        total += 8/( (4*k-3) * (4*k-1) )
        k += 1
```

$\sum_{k=1}^{\infty} \frac{8}{(4k-3) \cdot (4k-1)} = \pi$

*From lecture 4*

```python
>>> for x in pi():
...     print(x)
0
2.6666666666666665
2.895238095238095
2.976046176046176
3.017071817071817
3.041839618929402
3.0584027659273314...
```
Sequences of Approximation

Is $\sqrt{51} - 4 < \pi$ ?

```python
def sqrt(a):
    x = 1
    while True:
        yield x
        x = (x + a/x)/2

def pi():
    total, k = 0, 1
    while True:
        yield total
        total += 8/((4*k-3)*(4*k-1))
        k += 1
```
Sequences of Approximation

Is \( \sqrt{51} - 4 \) < \( \pi \)?

```python
def sqrt(a):
    x = 1
    while True:
        yield x
        x = (x + a/x)/2

def pi():
    total, k = 0, 1
    while True:
        yield total
        total += 8/((4*k-3)*(4*k-1))
        k += 1

def four():
    while True:
        yield 4
```

\[ p_5^1 \] \( \pi \)
Sequences of Approximation

Is $\sqrt{51} - 4 < \pi$?

```python
def sqrt(a):
    x = 1
    while True:
        yield x
        x = (x + a/x)/2

def pi():
    total, k = 0, 1
    while True:
        yield total
        total += 8/((4*k-3)*(4*k-1))
        k += 1

def four():
    while True:
        yield 4

def subtract(x, y):
    while True:
        yield next(x) - next(y)
```
Sequences of Approximation

Is \( \sqrt{51} - 4 < \pi \)?

```
def sqrt(a):
    x = 1
    while True:
        yield x
        x = (x + a/x)/2

def pi():
    total, k = 0, 1
    while True:
        yield total
        total += 8/((4*k-3)*(4*k-1))
        k += 1

def four():
    while True:
        yield 4

def subtract(x, y):
    while True:
        yield next(x) - next(y)
```

(C) Assume that \( s \) is a SoA to \( y \) and each element of \( s \) is closer to \( y \) than the last. Define \( \text{less\_than\_0}(s) \) that returns True if it is certain that \( y < 0 \).
Sequences of Approximation

Is \( \sqrt{51} - 4 \) < \( \pi \)?

```python
def sqrt(a):
    x = 1
    while True:
        yield x
        x = (x + a/x)/2

def pi():
    total, k = 0, 1
    while True:
        yield total
        total += 8/((4*k-3)*(4*k-1))
        k += 1

def four():
    while True:
        yield 4

def subtract(x, y):
    while True:
        yield next(x) - next(y)
```

(C) Assume that \( s \) is a SoA to \( y \) and each element of \( s \) is closer to \( y \) than the last. Define \( \text{less\_than\_0}(s) \) that returns True if it is certain that \( y < 0 \).
Sequences of Approximation

Is $\sqrt{51} - 4$ < $\pi$ ?

```
def sqrt(a):
    x = 1
    while True:
        yield x
        x = (x + a/x)/2

def pi():
    total, k = 0, 1
    while True:
        yield total
        total += 8/((4*k-3)*(4*k-1))
        k += 1

def four():
    while True:
        yield 4

def subtract(x, y):
    while True:
        yield next(x) - next(y)
```

(C) Assume that $s$ is a SoA to $y$ and each element of $s$ is closer to $y$ than the last. Define $\text{less}_{-}0(s)$ that returns True if it is certain that $y < 0$.

```
def less_than_0(s):
    current = next(s)
    while True:
        last, current = current, next(s)
        if ____________________________________________:
            return True
```

```
Sequences of Approximation

Is $\sqrt{51} - 4 < \pi$?

```python
def sqrt(a):
    x = 1
    while True:
        yield x
        x = (x + a/x)/2

def pi():
    total, k = 0, 1
    while True:
        yield total
        total += 8/((4*k-3)*(4*k-1))
        k += 1

def four():
    while True:
        yield 4

def subtract(x, y):
    while True:
        yield next(x) - next(y)
```

(C) Assume that $s$ is a SoA to $y$ and each element of $s$ is closer to $y$ than the last. Define `less_than_0(s)` that returns True if it is certain that $y < 0$.

```python
def less_than_0(s):
    current = next(s)
    while True:
        last, current = current, next(s)
        if ____________________________:
            return True
```

```python
1 2 3 4 5 6 7
-5 -10 -15 -20 -25
```

```
```

```python
```

```python
```
Sequences of Approximation

Is $\sqrt{51} - 4 < \pi$? Is $\sqrt{51} - 4 - \pi < 0$?

```python
def sqrt(a):
    x = 1
    while True:
        yield x
        x = (x + a/x)/2

def pi):
    total, k = 0, 1
    while True:
        yield total
        total += 8/((4*k-3)*(4*k-1))
        k += 1

def four():
    while True:
        yield 4

def subtract(x, y):
    while True:
        yield next(x) - next(y)
```

(C) Assume that $s$ is a SoA to $y$ and each element of $s$ is closer to $y$ than the last. Define `less_than_0(s)` that returns True if it is certain that $y < 0$.

```python
def less_than_0(s):
    current = next(s)
    while True:
        last, current = current, next(s)
        if _____________________________________________:
            return True
```
Sequences of Approximation

Is $\sqrt{51} - 4 < \pi$? Is $\sqrt{51} - 4 - \pi < 0$?

```python
>>> a = subtract(sqrt(51), four())
>>> less_than_0(subtract(a, pi()))
```

(C) Assume that $s$ is a SoA to $y$ and each element of $s$ is closer to $y$ than the last.

Define `less_than_0(s)` that returns True if it is certain that $y < 0$.

```python
def less_than_0(s):
    current = next(s)
    while True:
        last, current = current, next(s)
        if current < last:
            return True
```
Sequences of Approximation

Is $\sqrt{51} - 4 < \pi$? Is $\sqrt{51} - 4 - \pi < 0$?

```python
>>> a = subtract(sqrt(51), four())
>>> less_than_0(subtract(a, pi()))
```

(C) Assume that $s$ is a SoA to $y$ and each element of $s$ is closer to $y$ than the last. Define less_than_0($s$) that returns True if it is certain that $y < 0$.

```python
def less_than_0($s$):
    current = next($s$)
    while True:
        last, current = current, next($s$)
        if ____________________________________________:
            return True
```

```python
def sqrt(a):
    x = 1
    while True:
        yield x
        x = (x + a/x)/2
def pi():
    total, k = 0, 1
    while True:
        yield total
        total += 8/((4*k-3)*(4*k-1))
        k += 1
def four():
    while True:
        yield 4
def subtract(x, y):
    while True:
        yield next(x) - next(y)
```
Sequences of Approximation

Is $\sqrt{51} - 4 < \pi$? Is $\sqrt{51} - 4 - \pi < 0$?

```python
>>> a = subtract(sqrt(51), four())
>>> less_than_0(subtract(a, pi()))
```

(C) Assume that $s$ is a SoA to $y$ and each element of $s$ is closer to $y$ than the last. Define $\text{less\_than\_0}(s)$ that returns True if it is certain that $y < 0$.

```python
def less_than_0(s):
    current = next(s)
    while True:
        last, current = current, next(s)
        if last < 0 and current < last:
            return True
```
Sequences of Approximation

Is $\sqrt{51} - 4 < \pi$? Is $\sqrt{51} - 4 - \pi < 0$?

```
>>> a = subtract(sqrt(51), four())
>>> less_than_0(subtract(a, pi()))
```

(C) Assume that $s$ is a SoA to $y$ and each element of $s$ is closer to $y$ than the last.
Define $\text{less}\_\text{than}\_0(s)$ that returns True if it is certain that $y < 0$.

```
def less_than_0(s):
    current = next(s)
    while True:
        last, current = current, next(s)
        if last < 0 and current < last:
            return True
```

(Demo)
Computer Science
61A was Designed to Introduce the Big Ideas in Computer Science
61A was Designed to Introduce the Big Ideas in Computer Science

What are functions, data, sequences, trees, programs, languages, and interpreters.
61A was Designed to Introduce the Big Ideas in Computer Science

What are functions, data, sequences, trees, programs, languages, and interpreters.
How to write legible programs, use recursion, measure complexity, and solve problems.
61A was Designed to Introduce the Big Ideas in Computer Science

What are functions, data, sequences, trees, programs, languages, and interpreters. How to write legible programs, use recursion, measure complexity, and solve problems. Different programming paradigms: functional, object-oriented, and declarative.
61A was Designed to Introduce the Big Ideas in Computer Science

What are functions, data, sequences, trees, programs, languages, and interpreters. How to write legible programs, use recursion, measure complexity, and solve problems. Different programming paradigms: functional, object-oriented, and declarative.

What's left to learn in CS?
61A was Designed to Introduce the Big Ideas in Computer Science

What are functions, data, sequences, trees, programs, languages, and interpreters.

How to write legible programs, use recursion, measure complexity, and solve problems.

Different programming paradigms: functional, object-oriented, and declarative.

What's left to learn in CS?
• Designing and testing software
61A was Designed to Introduce the Big Ideas in Computer Science

What are functions, data, sequences, trees, programs, languages, and interpreters. How to write legible programs, use recursion, measure complexity, and solve problems. Different programming paradigms: functional, object-oriented, and declarative.

What's left to learn in CS?
• Designing and testing software
• Algorithms for solving known problems
61A was Designed to Introduce the Big Ideas in Computer Science

What are functions, data, sequences, trees, programs, languages, and interpreters. How to write legible programs, use recursion, measure complexity, and solve problems. Different programming paradigms: functional, object-oriented, and declarative.

What's left to learn in CS?
- Designing and testing software
- Algorithms for solving known problems
- Low-level representations of data and programs
61A was Designed to Introduce the Big Ideas in Computer Science

What are functions, data, sequences, trees, programs, languages, and interpreters.
How to write legible programs, use recursion, measure complexity, and solve problems.
Different programming paradigms: functional, object-oriented, and declarative.

What's left to learn in CS?
• Designing and testing software  
• Algorithms for solving known problems  
• Low-level representations of data and programs  
• Discrete mathematics and analysis of programs
61A was Designed to Introduce the Big Ideas in Computer Science

What are functions, data, sequences, trees, programs, languages, and interpreters. How to write legible programs, use recursion, measure complexity, and solve problems. Different programming paradigms: functional, object-oriented, and declarative.

What's left to learn in CS?
• Designing and testing software
• Algorithms for solving known problems
• Low-level representations of data and programs
• Discrete mathematics and analysis of programs
• Programming languages
61A was Designed to Introduce the Big Ideas in Computer Science

What are functions, data, sequences, trees, programs, languages, and interpreters. How to write legible programs, use recursion, measure complexity, and solve problems. Different programming paradigms: functional, object-oriented, and declarative.

What's left to learn in CS?
• Designing and testing software
• Algorithms for solving known problems
• Low-level representations of data and programs
• Discrete mathematics and analysis of programs
• Programming languages
• User interface design
61A was Designed to Introduce the Big Ideas in Computer Science

What are functions, data, sequences, trees, programs, languages, and interpreters. How to write legible programs, use recursion, measure complexity, and solve problems. Different programming paradigms: functional, object-oriented, and declarative.

What's left to learn in CS?
• Designing and testing software
• Algorithms for solving known problems
• Low-level representations of data and programs
• Discrete mathematics and analysis of programs
• Programming languages
• User interface design
• Networking
61A was Designed to Introduce the Big Ideas in Computer Science

What are functions, data, sequences, trees, programs, languages, and interpreters.
How to write legible programs, use recursion, measure complexity, and solve problems.
Different programming paradigms: functional, object-oriented, and declarative.

What's left to learn in CS?
• Designing and testing software
• Algorithms for solving known problems
• Low-level representations of data and programs
• Discrete mathematics and analysis of programs
• Programming languages
• User interface design
• Networking
• Systems
61A was Designed to Introduce the Big Ideas in Computer Science

What are functions, data, sequences, trees, programs, languages, and interpreters. How to write legible programs, use recursion, measure complexity, and solve problems. Different programming paradigms: functional, object-oriented, and declarative.

What's left to learn in CS?
* Designing and testing software
* Algorithms for solving known problems
* Low-level representations of data and programs
* Discrete mathematics and analysis of programs
* Programming languages
* User interface design
* Networking
* Systems
* Artificial intelligence
61A was Designed to Introduce the Big Ideas in Computer Science

What are functions, data, sequences, trees, programs, languages, and interpreters. How to write legible programs, use recursion, measure complexity, and solve problems. Different programming paradigms: functional, object-oriented, and declarative.

What's left to learn in CS?
• Designing and testing software
• Algorithms for solving known problems
• Low-level representations of data and programs
• Discrete mathematics and analysis of programs
• Programming languages
• User interface design
• Networking
• Systems
• Artificial intelligence
• Lots of other subfields: graphics, theory, scientific computing, security, etc.
Life
Important Ideas Take a Long Time to Learn
Important Ideas Take a Long Time to Learn

- It's a good idea to study subjects other than computer science.
Important Ideas Take a Long Time to Learn

• It's a good idea to study subjects other than computer science.
• Who you spend your time with is important.
Important Ideas Take a Long Time to Learn

• It's a good idea to study subjects other than computer science.
• Who you spend your time with is important.
• Ideas come from people, and people think from experience.
Important Ideas Take a Long Time to Learn

• It's a good idea to study subjects other than computer science.
• Who you spend your time with is important.
• Ideas come from people, and people think from experience.
• Don't compare.
Important Ideas Take a Long Time to Learn

• It's a good idea to study subjects other than computer science.
• Who you spend your time with is important.
• Ideas come from people, and people think from experience.
• Don't compare.
• Contribute to the world.
Thanks for being amazing!

Please stay for the HKN survey.