Interactive Program Distillation

Andrew Head
UC Berkeley

This talk will begin at 10:03am PST.
That's it, that's all we have to do. Now if we want to use them in our game, we simply make a `Coin(x, y)` or `DragonCoin(x, y)` and then add them to our list of pickups, not the player only:

```java
1. class MultiLayer extends LevelLayer {
   2.    // ...}
3.   MultiLayer(level, room) {
4.     // ...}
5.     // add coins above the horizontal platforms
6.     addCoin(320, height - 220, 36);
7.     addCoin(731, height - 190, 36);
8.     addCoin(444, height - 140, 36);
9.     // add a dragon coin at the start
10.    addMultiLayerOnly(new DragonCoin(323, height - 154));
11.    // a handy function for placing lots of coins
12.    void addCoins(int x, float y, float w) {
13.       float step = 81; i = 0, last = w/step;
14.       for (i = 0; i < w; i += step),
15.       addMultiLayerOnly(new Coin(x + step * i, y));
16.    }

The `addCoins` function is a convenient function that lets us add a string of coins starting at position `x, y` and spanning a width of `w`. We use the `step` value to space out our coins, add 18 pixels as distance from one coin's center to the next coin's center, and then we start adding coins "for the player only", as is obvious from the `addMultiLayerOnly(x, y)` function name. The result? Why, let's play our updated game and see for ourselves.
Now if we want to use them in our game, we simply make \[\text{new Coin}(\ldots,\ldots)\] or \[\text{new DragonCoin}(\ldots,\ldots)\] and then add them to our list of pickups, only for the player:
A sample program

1. Written instructions

... Now if we want to use them in our game, we simply make `[new Coin(...,...)]` or `[new DragonCoin(...,...)]` and then add them to our list of pickups, for the player only:

2. Code snippets

```java
class MarioLayer extends LevelLayer {
    [...]
    MarioLayer(Level owner) {
        [...]
        // add coins above the horizontal platforms
        addCoins(928,height-236,96);
        addCoins(912,height-140,128);
        addCoins(1442,height-140,128);
        [...]
        // add a dragon coin at the start
        addForPlayerOnly(new DragonCoin(352,height-164));
    }
}
```
A sample program

1. Written instructions
2. Code snippets
3. Expected results

```java
class MultiLayer extends LevelLayer {
    [...] // class definition...
    @Override
    protected void update(float delta) {
        [...] // update logic...
    }
}
```

The `addCoins()` function is a convenient function that lets us add a string of coins starting at position `x: y` and spanning a width of `w`. We use the `t` value to space out our coins, add 10 pixels or distance from one coin’s center to the next coin’s center, and then we start adding coins “for the player only,” as is obvious from the `addPlayerOnly()` function name. The result? Why, let’s play our updated game and see for ourselves.

Now with shiny coins!
How do sample programs get written?

41 snippets, 22 outputs
How do sample programs get written?

source program

output

41 snippets, 22 outputs
How do sample programs get written?

Source program → select → output

Author

41 snippets, 22 outputs
How do sample programs get written?

source program \[\xrightarrow{\text{select}}\] simplify

output

Author

41 snippets, 22 outputs
How do sample programs get written?

source program  select  simplify  sequence

output

41 snippets, 22 outputs
How do sample programs get written?

- Source program
- Select
- Simplify
- Sequence
- Supplement

41 snippets, 22 outputs

Author

Output
How do sample programs get written?

source program

select

simplify

sequence

supplement

output

Author

41 snippets, 22 outputs
How do sample programs get written?

This is an iterative (not linear) process.
Program distillation

Existing Program

Sample Program

Snippet

Tutorial
Program distillation

Existing Program

Sample Program
Snippet
Tutorial

Author
Program distillation

Existing Program

Sample Program

Snippet

Author

Tool

Tutorial
Authors can transform existing programs into sample programs more efficiently and flexibly when aided by interactive tools for selecting, simplifying, supplementing, and sequencing code.
The Essence of Distillation Tools
The Essence of Distillation Tools

1. Interactions
   - select
   - simplify
   - supplement
   - sequence

Author

Existing Program

Snippet

Tutorial

Sample Program
The Essence of Distillation Tools

1. Interactions
   - select
   - simplify
   - supplement
   - sequence

2. Program Analysis

   ![Diagram showing interactions and program analysis steps]

   - Author
   - Existing Program
   - Tutorial
   - Sample Program
   - Snippet
The Essence of Distillation Tools

1. Interactions
   - select
   - simplify
   - supplement
   - sequence

2. Program Analysis

3. Efficient, Flexible Authoring

Author

Existing Program

Snippet

Tutorial

Sample Program
This Talk

**CodeScoop [CHI '18]**
- code editor
- snippets
- select
- simplify

**Torii [CHI '20]**
- code editor
- tutorials
- supplement
- sequence

**Gathering tools [CHI '19]**
- notebook
- snippets
- tutorials
- select
- supplement
This Talk

**CodeScoop [CHI '18]**
- **code editor**
- snippets
- select
- simplify

**Torii [CHI '20]**
- **code editor**
- tutorials
- supplement
- sequence

**Gathering tools [CHI '19]**
- notebook
- snippets
- tutorials
- select
- supplement
How can tools make it easier for programmers to share snippets from their own code?

Detailed, personal code

Concise, self-contained snippet

Post online, share locally, ....
Formative Study

12 programmers creating samples from their own programs.
Existing Program
Transcription errors

Forgotten code

...and time-consuming removal of dead code
Transcription errors
Forgotten code
Edit errors
...and time-consuming removal of dead code
Q. How can tools make it easier for programmers to share snippets from their own code?

A distillation tool should:

- Help authors *select* subsets of code quickly and completely
- Help make *simplifications* without introducing errors
```java
int rowNumber = 0;
while (finished == false) {

    int rowCount = cursor.rowCount();

    for (int i = 0; i < Math.min(rowCount, maxBooks); ++i) {

        cursor.fetchone();
        int id = cursor.getInt(COLUMN_INDEX_ID);
        String title = cursor.getString(COLUMN_INDEX_TITLE);
        int year = cursor.getInt(COLUMN_INDEX_YEAR);
        int num_pages = cursor.getInt(COLUMN_INDEX_NUM_PAGES);
        Book book = new Book(id, title, year, num_pages);

        if (title != null) {
            titles.add(title);
        }

        if (id != -1) {
```

int rowNumber = 0;
while (finished == false) {
    int rowCount = cursor.rowCount();

    for (int i = 0; i < Math.min(rowCount, maxBooks); ++i) {
        cursor.fetchone();
        int id = cursor.getInt(COLUMN_INDEX_ID);
        String title = cursor.getString(COLUMN_INDEX_TITLE);
        int year = cursor.getInt(COLUMN_INDEX_YEAR);
        int numPages = cursor.getInt(COLUMN_INDEX_NUM_PAGES);

        if (id != -1) {
            // Do something with id, title, year, numPages
        }
    }
}
(1) Author **selects** pattern
(2) Editor creates snippet,
(1) Author **selects** pattern
(2) Editor creates snippet,
(3) Flags errors,
(1) Author **selects** pattern
(2) Editor creates snippet,
(3) Flags errors,
(4) Suggests code fixes,
(1) Author **selects** pattern
(2) Editor creates snippet,
(3) Flags errors,
(4) Suggests code fixes,
(5) Suggests simplifications,
Author selects pattern
Editor creates snippet,
Flags errors,
Suggests code fixes,
Suggests simplifications,
Author selects pattern
Editor creates snippet,
Flags errors,
Suggests code fixes,
Suggests simplifications,
And makes automatic fixes.
for (____ _ = _; _ < _____·___(_______, ___________); ++_) {
cursor.fetchone();  // A row of data is fetched from the database.

int id = cursor.getInt(______________________);

_______ ______ = ______·______·________(______________________________);
______ _____ = ______·______·____________(______________________________);
____ _______ = ______·______·____________(______________________________);
Book book = new Book(id, ______, _____, ____, ____, ______________);
A row of data is fetched from the database.

This is data for a book.
for (int i = 0; i < Math.min(rowCount, maxBooks); ++i) {
  cursor.fetchone();
  int id = cursor.getInt(COLUMN_INDEX_ID);
  String title = cursor.getString(COLUMN_INDEX_TITLE);
  int year = cursor.getInt(COLUMN_INDEX_YEAR);
  int num_pages = cursor.getInt(COLUMN_INDEX_NUM_PAGES);
  Book book = new Book(id, title, year, num_pages);
Program Slicing
Program Slicing
Program **Slicing**

Program **Scooping**
Program Slicing

Program Scooping
Program **Slicing**

Program **Scooping**
```java
int rowCount = cursor.rowCount();

for (int i = 0; i < Math.min(rowCount, maxBooks); ++i) {

cursor.fetchone();
int id = cursor.getInt(COLUMN_INDEX_ID);
String title = cursor.getString(COLUMN_INDEX_TITLE);
int year = cursor.getInt(COLUMN_INDEX_YEAR);
int num_pages = cursor.getInt(COLUMN_INDEX_NUM_PAGES);
Book book = new Book(id, title, year, num_pages);

if (title != null) {
    titles.add(title);
}

if (id != -1) {
    boolean bestseller = isBestseller(book.getId());
    if (bestseller) {
        booklist.hasBestseller = bestseller;
    }
}

if (DEBUG == true) {
}
```
public class ExtractedExample {

  public static void main(String[] args) {

    String title = cursor.getString(COLUMN_INDEX_TITLE);

  }

}
public class ExtractedExample {

    Cursor cursor = database.cursor();
    Booklist booklist = new Booklist();
    List titles = new ArrayList();

    try {

        cursor.execute("SELECT * FROM books");
        boolean finished = false;

        if (cursor.rowCount() > 0) {

            int rowNumber = 0;
            while (finished == false) {

                int rowCount = cursor.rowCount();

                for (int i = 0; i < Math.min(rowNumber, rowCount); i++) {

                    cursor.fetchone();
                    int id = cursor.getInt("id");
                    int year = cursor.getInt("year");
                    int num_pages = cursor.getInt("num_pages");

                    Book book = new Book(id, year, num_pages);
                    booklist.add(book);

                    if (i == rowNumber) {  // Change the condition to check for rowNumber instead of rowCount -1
                        String title = cursor.getString("title");
                        titles.add(title);
                    } else {
                        String title = cursor.getString("title");
                        titles.add(title);
                    }
                }
            }
        }
    }
}
```java
public class ExtractedExample {

    public static void main(String[] args) {

        Cursor cursor = database.cursor();
        String title = cursor.getString(COLUMN_INDEX_TITLE);
        int id = cursor.getInt(COLUMN_INDEX_ID);
        int year = cursor.getInt(COLUMN_INDEX_YEAR);
        int num_pages = cursor.getInt(COLUMN_INDEX_PAGES);

        Book book = new Book(id, title, year, num_pages);
    }

    try {
        cursor.execute(QUERY);
        boolean finished = false;

        if (cursor.rowCount() > 0) {
            int rowNumber = 0;
            while (finished == false) {
                int rowCount = cursor.rowCount();
                for (int i = 0; i < Math.min(rowNumber, rowCount); i++) {
                    cursor.fetchone();
                    int id = cursor.getInt(COLUMN_INDEX_ID);
                    String title = cursor.getString(COLUMN_INDEX_TITLE);
                    int year = cursor.getInt(COLUMN_INDEX_YEAR);
                    int num_pages = cursor.getInt(COLUMN_INDEX_PAGES);

                    Book book = new Book(id, title, year, num_pages);
                }
                finished = true;
            }
        }
    }
}
```
```java
public class ExtractedExample {

    public static void main(String[] args) {
        Cursor cursor = database.cursor();
        try {
            cursor.execute(QUERY);
            String title = cursor.getString(COLUMN_INDEX_TITLE);
            int id = cursor.getInt(COLUMN_INDEX_ID);
            int year = cursor.getInt(COLUMN_INDEX_YEAR);
            int num_pages = cursor.getInt(COLUMN_INDEX_NUM_PAGES);
            Book book = new Book(id, title, year, num_pages);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```
```java
String QUERY = "SELECT id, title, year FROM booklist;"
int COLUMN_INDEX_ID = 0;
int COLUMN_INDEX_TITLE = 1;
int COLUMN_INDEX_YEAR = 2;
int COLUMN_INDEX_NUM_PAGES = 3;
boolean DEBUG = true;

Database database = new Database("local".
Cursor cursor = database.cursor();
Booklist booklist = new Booklist();
List titles = new ArrayList();

try {
    cursor.execute(QUERY);
    boolean finished = false;
    if (cursor.rowCount() > 0) {
        int rowCount = cursor.rowCount();
        while (finished == false) {
            int rowNumber = 0;
            while (rowNumber < rowCount) {
                int i = 0;
                for (int i = 0; i < Math.max(10, rowCount); ++i) {
                    // Code implementation...
                }
            }
        }
    }
}
```
```java
public class ExtractedExample {

    public static void main(String[] args) {
        Cursor cursor = database.cursor();
        try {
            cursor.execute("SELECT id, title, year, num_pages FROM table_name");
            String title = cursor.getString(1);  
        } catch (Exception exception) {
        }
    }

    public class Booklist {
        List titles = new ArrayList();
    }
}
```
Scooping Summary

First selections
Scooping Summary

Code fixups

First selections
Scooping Summary

- Optional control
- Code fixups
- First selections
Scooping Summary

Variable substitutions
Optional control
Code fixups
First selections
Program Analysis

- Selecting Code
- Adding in Forgotten Lines
- Simplifying the Code
- Automatic Fixes

Existing Program

Working Snippet
Program Analysis

- Selecting Code
- Adding in Forgotten Lines
- Simplifying the Code
- Automatic Fixes

Existing Program → Static Dataflow Analysis → Working Snippet
Program Analysis

- Selecting Code
- Adding in Forgotten Lines
- Simplifying the Code
- Automatic Fixes

Existing Program → Static Dataflow Analysis → Execution Trace → Working Snippet
Program Analysis

- Selecting Code
- Adding in Forgotten Lines
- Simplifying the Code
- Automatic Fixes

Existing Program → Working Snippet

- Static Dataflow Analysis
- Execution Trace
- Reflections

Parse Tree Walkers
Evaluating CodeScoop

Q1. Does CodeScoop support efficient extraction of snippets?

Q2. Does CodeScoop provide flexible authoring choices?
A Pilot Study about Snippet Extraction

Participants: \( N = 19 \) undergraduate student programmers

Main Task: Create snippets from existing code
Measurements: Usability of CodeScoop vs. baseline text editor, Preference for their scoop vs. an automatic slice, time to extract a snippet, ...

Qualitative Feedback: Survey and Interview
<table>
<thead>
<tr>
<th>Question</th>
<th>Yes/No</th>
<th>Difference (7-pt scale)</th>
<th>p Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faster to use?</td>
<td>Yes</td>
<td>5.8 min vs 9.6 min</td>
<td>&lt; .001</td>
</tr>
<tr>
<td>Easier to use?</td>
<td>Yes</td>
<td>Δ = 3</td>
<td>&lt; .01</td>
</tr>
<tr>
<td>More enjoyable?</td>
<td>Yes</td>
<td>Δ = 3</td>
<td>&lt; .01</td>
</tr>
<tr>
<td>Producing more satisfying samples?</td>
<td>Yes</td>
<td>Δ = 2</td>
<td>&lt; .01</td>
</tr>
</tbody>
</table>
"[CodeScoop's features] made creating an example a lot easier because I just had to look at the relevant code and see if I needed it or not instead of having to manually add them in."

**CodeScoop provided a median of...**

12 automatic corrections

5 suggestions of optional code

2 suggestions of error fixes
A simplification that shortens the snippet

For Task 3, participants made an important simplification:

Slice (101 lines)  Scoop (median = 36 lines)
Flexibility: Extracting snippets in different ways

<table>
<thead>
<tr>
<th>Variable</th>
<th>Add Code</th>
<th>Insert Literal</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLUMN_INDEX_ID</td>
<td>18</td>
<td>5 6 15</td>
</tr>
<tr>
<td>COLUMN_INDEX_NUM_PAGES</td>
<td>18</td>
<td>5 12 15</td>
</tr>
<tr>
<td>COLUMN_INDEX_TITLE</td>
<td>18</td>
<td>5 6 12 15</td>
</tr>
<tr>
<td>COLUMN_INDEX_YEAR</td>
<td>18</td>
<td>5 12 15</td>
</tr>
<tr>
<td>num_pages</td>
<td>5 18</td>
<td></td>
</tr>
<tr>
<td>QUERY</td>
<td>5 17</td>
<td>6 18</td>
</tr>
</tbody>
</table>

**Task 1**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Add Code</th>
<th>Insert Literal</th>
</tr>
</thead>
<tbody>
<tr>
<td>arg0</td>
<td>3 4 8 19</td>
<td>16</td>
</tr>
<tr>
<td>priceInt</td>
<td>3 4</td>
<td></td>
</tr>
<tr>
<td>query</td>
<td>3 4 8 16 19</td>
<td>7 9</td>
</tr>
</tbody>
</table>

**Task 2**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Add Code</th>
<th>Insert Literal</th>
</tr>
</thead>
<tbody>
<tr>
<td>arg1</td>
<td>2 10</td>
<td>14</td>
</tr>
<tr>
<td>destination</td>
<td>1 2 10 13 14</td>
<td>11</td>
</tr>
<tr>
<td>messageHtml</td>
<td>1 2 10 14</td>
<td>11 13</td>
</tr>
<tr>
<td>password</td>
<td>1 2 13</td>
<td>11 14</td>
</tr>
<tr>
<td>sslFactoryClass</td>
<td>1 2 10 11 13</td>
<td>14</td>
</tr>
<tr>
<td>username</td>
<td>1 2 13</td>
<td>11 14</td>
</tr>
</tbody>
</table>

**Task 3**
Flexibility: Extracting snippets in different ways

```
try {
    if (cursor.rowCount() > 0) {

    }
} catch (ConnectionException exception) {
}
```

Choice: **Error checking** through exceptions and postconditions.

```
int COLUMN_INDEX_TITLE = 1;
String title = cursor.getString(COLUMN_INDEX_TITLE);
Book book = new Book(____ title, _____ ___________);
```

Choice: Column **variable names**, saving results to **Book** object.
Takeaways from Study

Q1. Scooping was efficient compared to using the baseline text editor.

Q2. Scooping provided flexibility in influencing the appearance of snippets.
A Content Analysis of the contents, structure, and format of 200 programming tutorials

Written in many languages, about many topics.

Each tutorial was analyzed for the presence of 23 properties.
Structure and dependencies in tutorials

- Fragments: 83%
- Duplicated Code: 59%
- Rewritten Code: 48%
### Structure and dependencies in tutorials

<table>
<thead>
<tr>
<th>Component</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fragments</td>
<td>83%</td>
</tr>
<tr>
<td>Duplicated Code</td>
<td>59%</td>
</tr>
<tr>
<td>Rewritten Code</td>
<td>48%</td>
</tr>
<tr>
<td>Generated Output</td>
<td>67%</td>
</tr>
<tr>
<td>Console Output</td>
<td>33%</td>
</tr>
<tr>
<td>Images of Output</td>
<td>32%</td>
</tr>
<tr>
<td>Live Demo...</td>
<td>15%</td>
</tr>
<tr>
<td>and Editable Code</td>
<td>5%</td>
</tr>
</tbody>
</table>
Interviews with 12 accomplished authors of programming tutorials.

Each author had written between a few and hundreds of tutorials.
Sequencing code $\rightarrow$ parallel views of the code

source program
Sequencing code → parallel views of the code

copied

tutorial

source program
Sequencing code → parallel views of the code

source program

copied

duplicated

tutorial
Sequencing code → parallel views of the code

copied

duplicated

generates output

source program

tutorial
Sequencing code → parallel views of the code

depends on languages and APIs (which change)

duplicated

generates output

source program

tutorial

copied
How to keep parallel views of code consistent

Proactive approaches: Starting with a reference implementation; Using version control.

Corrective approaches: Manually following one's own tutorial and checking the end result; Regenerating outputs as code changes.
Simultaneous support of **sequencing** and **supplementing**

**source program**

**tutorial**
Simultaneous support of **sequencing** and **supplementing**
Simultaneous support of sequencing and supplementing
Simultaneous support of **sequencing** and **supplementing**
Simultaneous support of **sequencing** and **supplementing**
Simultaneous support of **sequencing** and **supplementing**
Simultaneous support of sequencing and supplementing
Simultaneous support of **sequencing** and **supplementing**
Simultaneous support of sequencing and supplementing
# An example of a class

class Rectangle:

    def __init__(self, w, h):
        self.w = w
        self.h = h
        self.description = "This shape has not been described yet"
        self.author = "Nobody has claimed to make this shape yet"

    def area(self):
        return self.w * self.h

    def set_description(self, text):
        self.description = text

    def set_scale(self, scale):
        self.w = self.w * scale
        self.h = self.h * scale

class Square(Rectangle):
    def __init__(self, w):
        self.w = w
        self.h = w

rectangle = Rectangle(90, 45)
long_rectangle = Rectangle(120, 10)
fat_rectangle = Rectangle(130, 120)

print("The width of the rectangle is: " + str(rectangle.w))
print("And its description is: " + str(rectangle.description))

# finding the area of your rectangle:
print("The area of the rectangle is: " + str(rectangle.area()))

# describing the rectangle
rectangle.set_description("A wide rectangle, more than twice as wide as it is tall")
# An example of a class

class Rectangle:
    def __init__(self, w, h):
        self.w = w
        self.h = h
        self.description = "This shape has not been described yet"
        self.author = "Nobody has claimed to make this shape yet"

    def area(self):
        return self.w * self.h

    def set_description(self, text):
        self.description = text

    def set_scale(self, scale):
        self.w = self.w * scale
        self.h = self.h * scale

class Square(Rectangle):
    def __init__(self, w):
        self.w = w
        self.h = w

rectangle = Rectangle(90, 45)
long_rectangle = Rectangle(120, 10)
fat_rectangle = Rectangle(130, 120)

print("The width of the rectangle is: " + str(rectangle.w))
print("And its description is: " + str(rectangle.description))

# finding the area of your rectangle:
print("The area of the rectangle is: " + str(rectangle.area()))

# describing the rectangle
rectangle.set_description("A wide rectangle, more than twice as wide as it is tall")
# An example of a class

class Rectangle:
    
def __init__(self, w, h):
        self.w = w
        self.h = h
        self.description = "This shape has not been described yet"
        self.author = "Nobody has claimed to make this shape yet"

    def area(self):
        return self.w * self.h

    def set_description(self, text):
        self.description = text

    def set_scale(self, scale):
        self.w = self.w * scale
        self.h = self.h * scale

class Square(Rectangle):
    def __init__(self, w):
        self.w = w
        self.h = w

rectangle = Rectangle(90, 45)
long_rectangle = Rectangle(120, 10)
fat_rectangle = Rectangle(130, 120)

print("The width of the rectangle is: " + str(rectangle.w))
print("And its description is: " + str(rectangle.description))

# finding the area of your rectangle:
print("The area of the rectangle is: " + str(rectangle.area()))

# describing the rectangle
rectangle.set_description("A wide rectangle, more than twice as wide as it is high")
# An example of a class

class Rectangle:

def __init__(self, w, h):
    self.w = w
    self.h = h
    self.description = "This shape has not been described yet"
    self.author = "Nobody has claimed to make this shape yet"

def area(self):
    return self.w * self.h

def set_description(self, text):
    self.description = text

def set_scale(self, scale):
    self.w = self.w * scale
    self.h = self.h * scale

class Square(Rectangle):
    def __init__(self, w):
        self.w = w
        self.h = w

rectangle = Rectangle(90, 45)
long_rectangle = Rectangle(120, 10)
fat_rectangle = Rectangle(130, 120)

print("The width of the rectangle is: " + str(rectangle.w))
print("And its description is: " + str(rectangle.description))

# finding the area of your rectangle:
print("The area of the rectangle is: " + str(rectangle.area()))

# describing the rectangle
rectangle.set_description("A wide rectangle, more than twice as wide as it is tall")

So far in Python, we've used a couple of types of organized data structures: lists, sets, and dictionaries. Each of these data structures have their own properties—like length, or keys. They also have their own methods, like the ability to "append" to a list or, or get the index of an element from the list.

What if we want to define our own data structures, with useful, common properties and data structures? Like, for instance, maybe we wanted to refer to a "shape" object that has an area and a perimeter in a program where we draw shapes on a screen.

Wouldn't it be cool if we could just create an object with all of these built-in properties and methods with a call that looked like this?
# An example of a class

class Rectangle:

def __init__(self, w, h):
    self.w = w
    self.h = h
    self.description = "This shape has not been described yet"
    self.author = "Nobody has claimed to make this shape yet"

def area(self):
    return self.w * self.h

def set_description(self, text):
    self.description = text

def set_scale(self, scale):
    self.w = self.w * scale
    self.h = self.h * scale

class Square(Rectangle):
    def __init__(self, w):
        self.w = w
        self.h = w

rectangle = Rectangle(90, 45)
long_rectangle = Rectangle(120, 10)
fat_rectangle = Rectangle(130, 120)

print("The width of the rectangle is: " + str(rectangle.w))
print("And its description is: " + str(rectangle.description))

# finding the area of your rectangle:
print("The area of the rectangle is: " + str(rectangle.area()))

# describing the rectangle
rectangle.set_description("A wide rectangle, more than twice as wide as it is long")

Programming with Objects in Python

So far in Python, we've used a couple of types of organized data structures: lists, sets, and dictionaries. Each of these data structures have their own properties—like length, or keys. They also have their own methods, like the ability to "append" to a list or, or get the index of an element from the list.

What if we want to define our own data structures, with useful, common properties and data structures? Like, for instance, maybe we wanted to refer to a "shape" object that has an area and a perimeter in a program where we draw shapes on a screen.

Wouldn't it be cool if we could just create an object with all of these built-in properties and methods with a call that looked like this?

```python
rectangle = Rectangle(90, 45)
```
# An example of a class

class Rectangle:
    def __init__(self, w, h):
        self.w = w
        self.h = h
        self.description = "This shape has not been described yet"
        self.author = "Nobody has claimed to make this shape yet"

    def area(self):
        return self.w * self.h

    def set_description(self, text):
        self.description = text

    def set_scale(self, scale):
        self.w = self.w * scale
        self.h = self.h * scale


class Square(Rectangle):
    def __init__(self, w):
        self.w = w
        self.h = w

rectangle = Rectangle(90, 45)
long_rectangle = Rectangle(120, 10)
fat_rectangle = Rectangle(130, 120)

print("The width of the rectangle is: " + str(rectangle.w))
print("And its description is: " + str(rectangle.description))

# finding the area of your rectangle:
print("The area of the rectangle is: " + str(rectangle.area()))

# describing the rectangle
rectangle.set_description("A wide rectangle, more than twice as wide as it is tall")

---

Programming with Objects in Python

So far in Python, we've used a couple of types of organized data structures: lists, sets, and dictionaries. Each of these data structures have their own properties---like length, or keys. They also have their own methods, like the ability to "append" to a list or, or get the index of an element from the list.

What if we want to define our own data structures, with useful, common properties and data structures? Like, for instance, maybe we wanted to refer to a "shape" object that has an area and a perimeter in a program where we draw shapes on a screen.

Wouldn't it be cool if we could just create an object with all of these built-in properties and methods with a call that looked like this?

```
rectangle = Rectangle(90, 45)
```
# An example of a class

class Rectangle:
    def __init__(self, w, h):
        self.w = w
        self.h = h
        self.description = "This shape has not been described yet"
        self.author = "Nobody has claimed to make this shape yet"

    def area(self):
        return self.w * self.h

    def set_description(self, text):
        self.description = text

    def set_scale(self, scale):
        self.w = self.w * scale
        self.h = self.h * scale


class Square(Rectangle):
    def __init__(self, w):
        self.w = w
        self.h = w

rectangle = Rectangle(90, 45)
long_rectangle = Rectangle(120, 10)
fat_rectangle = Rectangle(130, 120)

print("The width of the rectangle is: ", str(rectangle.w))
print("And its description is: ", str(rectangle.description))
print("The area of the rectangle is: ", str(rectangle.area()))
print("A wide rectangle, more than twice as wide as it is tall")

# Programming with Objects in Python

So far in Python, we’ve used a couple of types of organized data structures: lists, sets, and dictionaries. Each of these data structures have their own properties—like length, or keys. They also have their own methods, like the ability to "append" to a list or, or get the index of an element from the list.

What if we want to define our own data structures, with useful, common properties and data structures? Like, for instance, maybe we wanted to refer to a “shape” object that has an area and a perimeter in a program where we draw shapes on a screen.

Wouldn’t it be cool if we could just create an object with all of these built-in properties and methods with a call that looked like this?

rectangle = Rectangle(90, 45)

In Python, such data structures are called "objects". And to create objects, first we have to define "templates" for each new type of object. These templates would list all of the properties that could be defined for the object, and the methods that you can call on the object to get its data or to transform that data.

Those templates are called classes, and here’s how we define them.

class Rectangle:
    def __init__(self, w, h):
        self.w = w
        self.h = h
        self.description = "This shape has not been described yet"
        self.author = "Nobody has claimed to make this shape yet"
# An example of a class

class Rectangle:
    def __init__(self, w, h):
        self.w = w
        self.h = h
        self.description = "This shape has not been described yet"
        self.author = "Nobody has claimed to make this shape yet"

    def area(self):
        return self.w * self.h

    def set_description(self, text):
        self.description = text

    def set_scale(self, scale):
        self.w = self.w * scale
        self.h = self.h * scale

class Square(Rectangle):
    def __init__(self, w):
        self.w = w
        self.h = w

rectangle = Rectangle(90, 45)
long_rectangle = Rectangle(120, 10)
fat_rectangle = Rectangle(130, 120)

print("The width of the rectangle is: " + str(rectangle.w))

print("And its description is: " + str(rectangle.description))

print("The area of your rectangle: " + str(rectangle.area()))

# describing the rectangle
rectangle.set_description("A wide rectangle, more than twice as wide as it is high")

This class is a template for creating "shape" objects. Each class has a name---here, it's `Shape`. They also have a "constructor" method---which is always called `__init__`, which is a function that creates a new instance of the object from a list of properties the object is supposed to have. This init function gets called to create the class whenever you call the name of the class. The body of the `__init__` function sets up the properties of the object by setting values on "self". These properties will be accessible using the dot operator on the object that gets created by the constructor.

So, paste the class code at the top of your file, and then initialize a shape using the command we show at the top of the tutorial. Then, let's see what data the rectangle has:
An example of a class

class Rectangle:

    def __init__(self, w, h):
        self.w = w
        self.h = h
        self.description = "This shape has not been described yet"
        self.author = "Nobody has claimed to make this shape yet"

    def area(self):
        return self.w * self.h

    def set_description(self, text):
        self.description = text

    def set_scale(self, scale):
        self.w = self.w * scale
        self.h = self.h * scale


class Square(Rectangle):

    def __init__(self, w):
        self.w = w
        self.h = w

rectangle = Rectangle(90, 45)

long_rectangle = Rectangle(120, 10)
fat_rectangle = Rectangle(130, 120)

print("The width of the rectangle is: " + str(rectangle.w))

print("And its description is: " + str(rectangle.description))

# finding the area of your rectangle:
print("The area of the rectangle is: " + str(rectangle.area()))

# describing the rectangle
rectangle.set_description("A wide rectangle, more than twice as wide as it is tall")

In Python, such data structures are called "objects". And to create objects, first we have to define "templates" for each new type of object. These templates would list all of the properties that could be defined for the object, and the methods that you can call on the object to get its data or to transform that data.

Those templates are called classes, and here's how we define them.

class Rectangle:

    def __init__(self, w, h):
        self.w = w
        self.h = h
        self.description = "This shape has not been described yet"
        self.author = "Nobody has claimed to make this shape yet"

This class is a template for creating "shape" objects. Each class has a name—here, it's Shape. They also have a "constructor" method—which is always called __init__, which is a function that creates a new instance of the object from a list of properties the object is supposed to have. This init function gets called to create the class whenever you call the name of the class. The body of the __init__ function sets up the properties of the object by setting values on "self". These properties will be accessible using the dot operator on the object that gets created by the constructor.

So, paste the class code at the top of your file, and then initialize a shape using the command we show at the top of the tutorial. Then, let's see what data the rectangle has:

print("The width of the rectangle is: " + str(rectangle.w))

The width of the rectangle is: 90
# An example of a class

class Rectangle:
    def __init__(self, w, h):
        self.w = w
        self.h = h
        self.description = "This shape has not been described yet"
        self.author = "Nobody has claimed to make this shape yet"

    def area(self):
        return self.w * self.h

    def set_description(self, text):
        self.description = text

    def set_scale(self, scale):
        self.w = self.w * scale
        self.h = self.h * scale

class Square(Rectangle):
    def __init__(self, w):
        self.w = w
        self.h = w

rectangle = Rectangle(90, 45)
long_rectangle = Rectangle(120, 10)
fat_rectangle = Rectangle(130, 120)

print("The width of the rectangle is: " + str(rectangle.w))
print("And its description is: " + str(rectangle.description))

# finding the area of your rectangle:
print("The area of the rectangle is: " + str(rectangle.area()))

# describing the rectangle
rectangle.set_description("A wide rectangle, more than twice as wide as it is tall")

In Python, such data structures are called "objects". And to create objects, first we have to define "templates" for each new type of object. These templates would list all of the properties that could be defined for the object, and the methods that you can call on the object to get its data or to transform that data.

Those templates are called classes, and here’s how we define them.

class Rectangle:
    def __init__(self, w, h):
        self.w = w
        self.h = h
        self.description = "This shape has not been described yet"
        self.author = "Nobody has claimed to make this shape yet"

This class is a template for creating "shape" objects. Each class has a name---here, it’s Shape. They also have a "constructor" method---which is always called __init__, which is a function that creates a new instance of the object from a list of properties the object is supposed to have. This init function gets called to create the class whenever you call the name of the class. The body of the __init__ function sets up the properties of the object by setting values on "self". These properties will be accessible using the dot operator on the object that gets created by the constructor.

So, paste the class code at the top of your file, and then initialize a shape using the command we show at the top of the tutorial. Then, let’s see what data the rectangle has:

```python
print("The width of the rectangle is: " + str(rectangle.w))
The width of the rectangle is: 90
```
# An example of a class

class Rectangle:
    
def __init__(self, w, h):
        self.w = w
        self.h = h
        self.description = "This shape has not been described yet"
        self.author = "Nobody has claimed to make this shape yet"
    
def area(self):
        return self.w * self.h
    
def set_description(self, text):
        self.description = text
    
def set_scale(self, scale):
        self.w = self.w * scale
        self.h = self.h * scale

class Square(Rectangle):
    
def __init__(self, w):
        self.w = w
        self.h = w
    
rectangle = Rectangle(90, 45)
long_rectangle = Rectangle(120, 10)
fat_rectangle = Rectangle(130, 120)

print("The width of the rectangle is: " + str(rectangle.w))
print("And its description is: " + str(rectangle.description))
print("The area of the rectangle is: " + str(rectangle.area()))
rectangle.set_description("A wide rectangle, more than twice as wide as it is tall")
rectangle.set_scale(2)

print("The width of the rectangle is: " + str(rectangle.w))

What if we want to define other properties of an object when we create it? Well in that case...

# An example of a class

class Rectangle:
    def __init__(self, w, h):
        self.w = w
        self.h = h
        self.description = "This shape has not been described yet"
        self.author = "Nobody has claimed to make this shape yet"

    def area(self):
        return self.w * self.h

    def set_description(self, text):
        self.description = text

    def set_scale(self, scale):
        self.w = self.w * scale
        self.h = self.h * scale

class Square(Rectangle):
    def __init__(self, w, h):
        super().__init__(w, h)

rectangle = Rectangle(90, 45)
long_rectangle = Rectangle(120, 10)
fat_rectangle = Rectangle(130, 120)

print("The width of the rectangle is: " + str(rectangle.w))
print("And its description is: " + str(rectangle.description))
print("The area of the rectangle is: " + str(rectangle.area()))

print("And its description is: This is a rectangle")
print("The width of the rectangle is: 90")
print("And its description is: This is a rectangle")
print("The area of the rectangle is: 9050")

Is it possible to make new classes of objects that build on other types of objects? Yes!
So let's say, for instance, you want to be able to create squares. A square needs only
requires us to define one dimension, as its width and height are the same:
# An example of a class

class Rectangle:

def __init__(self, w, h):
    self.w = w
    self.h = h
    self.description = "This shape has not been described yet"
    self.author = "Nobody has claimed to make this shape yet"

def area(self):
    return self.w * self.h

def set_description(self, text):
    self.description = text

def set_scale(self, scale):
    self.w = self.w * scale
    self.h = self.h * scale

class Square(Rectangle):
    def __init__(self, w):
        self.w = w
        self.h = w

rectangle = Rectangle(90, 45)
long_rectangle = Rectangle(120, 10)
fat_rectangle = Rectangle(130, 120)

print("The width of the rectangle is: " + str(rectangle.w))
print("And its description is: " + str(rectangle.description))
# finding the area of your rectangle:
print("The area of the rectangle is: " + str(rectangle.area()))
# describing the rectangle
rectangle.set_description("A wide rectangle, more than twice as wide as it is tall")

print("The area of the rectangle is: " + str(rectangle.area()))

Then the `area` function gets invoked. The object is passed in as the `self` parameter. The properties on the object---like width and height---are accessible through `self`. This method then computes area by multiplying the dimensions. Add the print statement above to your program, and you should see this area:

The width of the rectangle is: 90
The area of the rectangle is: 4050

Is it possible to make new classes of objects that build on other types of objects? Yes! Let's say, for instance, you want to be able to create squares. A square needs only requires us to define one dimension, as its width and height are the same.
```python
# An example of a class

class Rectangle:
    def __init__(self, w, h):
        self.w = w
        self.h = h
        self.description = "This shape has not been described yet"
        self.author = "Nobody has claimed to make this shape yet"

    def area(self):
        return self.w * self.h

    def set_description(self, text):
        self.description = text

    def set_scale(self, scale):
        self.w = self.w * scale
        self.h = self.h * scale

class Square(Rectangle):
    def __init__(self, w):
        self.w = w
        self.h = w

rectangle = Rectangle(90, 45)
large_rectangle = Rectangle(120, 10)
fat_rectangle = Rectangle(130, 20)

print("The width of the rectangle is: " + str(rectangle.w))
print("And its description is: " + str(rectangle.description))
# finding the area of your rectangle:
print("The area of the rectangle is: " + str(rectangle.area()))
# describing the rectangle
rectangle.set_description("A wide rectangle, more than twice as wide as it is tall")
```

Then the `area()` method gets invoked. The object is passed in as the `self` parameter. The properties on the object---like width and height---are accessible through `self`. This method then computes area by multiplying the dimensions. Add the print statement above to your program, and you should see this area:

```
The width of the rectangle is: 90
The area of the rectangle is: 4050
```

Is it possible to make new classes of objects that build on other types of objects? Yes! So let's say, for instance, you want to be able to create squares. A square needs only requires us to define one dimension, as its width and height are the same:
class Square(Rectangle):
    def __init__(self, w):
        self.w = w
        self.h = w

rectangle = Rectangle(90, 45)
long_rectangle = Rectangle(120, 10)
fat_rectangle = Rectangle(130, 120)

print("The width of the rectangle is: " + str(rectangle.w))
print("And its description is: " + str(rectangle.description))

# finding the area of your rectangle:
print("The area of the rectangle is: " + str(rectangle.area()))

# describing the rectangle
rectangle.set_description("A wide rectangle, more than twice as wide as it is tall")

# making the rectangle 50% smaller
rectangle.set_scale(0.5)

# re-printing the new area of the rectangle
print(rectangle.area())

square = Square(100)
print("The square's width: " + str(square.w))
print("The square's height: " + str(square.h))

Then the `area` def gets invoked. The object is passed in as the `self` parameter. The properties of the object—like width and height—are accessible through `self`. This method then computes area by multiplying the dimensions. Add the print statement above to your program, and you should see this area:

The width of the rectangle is: 90
The area of the rectangle is: 4050

Is it possible to make new classes of objects that build on other types of objects? Yes! So let's say, for instance, you want to be able to create squares. A square needs only requires us to define one dimension, as its width and height are the same:
```python
class Square(Rectangle):
    def __init__(self, w):
        self.w = w
        self.h = w

rectangle = Rectangle(90, 45)
long_rectangle = Rectangle(120, 10)
fat_rectangle = Rectangle(130, 120)

print("The width of the rectangle is: " + str(rectangle.w))
print("And its description is: " + str(rectangle.description))

# finding the area of your rectangle:
print("The area of the rectangle is: " + str(rectangle.area()))

# describing the rectangle
rectangle.set_description("A wide rectangle, more than twice as wide as it is high")

# making the rectangle 50% smaller
rectangle.set_scale(0.5)

# re-printing the new area of the rectangle
print(rectangle.area())

square = Square(100)
print("The square's width: " + str(square.w))
print("The square's height: " + str(square.h))
```

Then the `area` method gets invoked. The object is passed in as the `self` parameter. The properties on the object---like width and height—are accessible through `self`. This method then computes area by multiplying the dimensions. Add the print statement above to your program, and you should see this area:

The width of the rectangle is: 90
The area of the rectangle is: 4050

Is it possible to make new classes of objects that build on other types of objects? Yes! So let’s say, for instance, you want to be able to create squares. A square needs only requires us to define one dimension, as its width and height are the same:

The width of the rectangle is: 90
The area of the rectangle is: 4050
The square's width: 100
The square's height: 100
Program Analysis

source order

tutorial order

snapshot

class Shape:
    def __init__(self, w, h):
        self.w = w
        self.h = h
        self.description = "This shape has not been described yet"
        self.author = "Nobody has claimed to make this shape"

    def area(self):
        return self.w * self.h

    def set_scale(self, scale):
        self.w = self.w * scale
        self.h = self.h * scale

rectangle = Shape(100, 45)

print("The width of the rectangle is", rectangle.w)
print("The height of the rectangle is", rectangle.h)
print("shape", rectangle.description)
print("It was created by", rectangle.author)
print("The area of the rectangle is", rectangle.area())

print("The width of the rectangle is: ", 2 * rect(w))

print("The width of the rectangle is: ", 2 * rect(w))

The width of the rectangle is: 
2 * rect(w)

execute

generated output
Evaluating Torii

Q1. Does Torii support efficient editing of tutorials?

Q2. Does Torii support flexible snippet organization?
An In-Lab Study of Torii

Participants: $N = 12$ tutorial authors

Maintenance Task $\times 2$: Update a tutorial, with Torii and with a baseline tool.

Exploration: Create a tutorial about object-oriented programming, based on an existing program.
Efficiency of editing tasks with Torii

Comparison Tool

Torii

subtask (a) linked edits

subtask (b) localized edits

subtask (c) revert edits
Efficiency of editing tasks with Torii

Comparison Tool

Torii

subtask (a) linked edits

Tim (min)

Time (min)

1 3 5

subtask (b) localized edits

Time (min)

1 3 5 7

subtask (c) revert edits

Time (min)

1 2 3

Enough evidence to show participants could use Torii to make these edits. Not enough to claim they're efficient with this design.
Presenting snippets flexibly

Often times when coding, one will want to create a kind of “Template” to create objects. These objects might have the same kinds of attributes, but different values from one another. For example, if we wanted to create multiple dog variable objects, you might create a template for a “Dog” with the different attributes of “name” and “color”. For each dog object you then create, you could assign those values. For example, when creating a Dog variable Mocha, you might assign its name to be “Mocha” and its color to be “brown”.

These kinds of “templates” are called “classes” in computer science and here we will learn how to create one.

The first thing we want to do when creating our template / class is to actually declare it. This is done simply with the keyword ‘class’ and the name of the object you’ll be creating. Here is the declaration of a Shape class that you will be seeing for the remainder of the tutorial.

```python
class Shape:
```

From here, we need to find a way to actually instantiate this new class. For a shape, some things we might care about are the “width” and “height”. In order to portray this, we would include these as instance variables, what we will call a constructor:

```python
def __init__(self, w, h):
```

Every time, we go to create a new ‘Shape’ type variable, we must provide the “init” of the shape class and know that and pass in a ‘w’ and an ‘h’ as its arguments. From there, it looks to its attributes using the body of the constructor:

```python
def __init__(self, w, h):
    self.w = w
    self.h = h
    self.description = "This shape has not been described yet"
    self.author = "Nobody has claimed to make this shape yet"
```

In order to tell the program that it actually needs to use this constructor to create a shape variable, we might do something like the following:

```python
rectangle = Shape(100, 45)
long_rectangle = Shape(120, 10)
fat_rectangle = Shape(30, 120)
```

Regardless of whatever reason you’re creating Shapes for, you will likely, at some point need to access those shapes’ height and width attributes. In Python, we can’t exactly say “get rectangle’s height”. Computers aren’t smart. However, we can use something called ‘dot notation’ that the computer will recognize as doing just that!

In dot notation, you specify the object you’re trying to access something from, follow it with a “.”, and then include the name of the attribute:

```python
name.attribute
```

A real example of this can be seen here:

```python
square = Square(100)
print(square.w)
```

Here (assuming we’ve created a square class and a Square type variable named “square”, we’re able to access its “w” attribute that was passed in as 100. This would print the following:

```
100
```

But where did this ‘Square’ come from? Our square is really a shape, so we can check its height and width.

Because a square is so similar to another one of our Shape class’
Presenting snippets flexibly

Repetition (6 of 12)
Show a class multiple times, adding new properties or methods each time

Out-of-order execution (1 of 12)
Present the use of a class before its declaration

Fragmentation (3 of 12)
Split up the declaration and implementation of a class / method
Takeaways from Study

Q1. Authors could perform linked edits more quicky with Torii than a baseline tool, though this is not statistically significant.

Q2. Authors used Torii's capabilities to flexibly organize snippets in tutorials.
This Talk

**CodeScoop [CHI '18]**
- code editor
- snippets
- select
- simplify

**Torii [CHI '20]**
- code editor
- tutorials
- supplement
- sequence

**Gathering tools [CHI '19]**
- notebook
- snippets
- tutorials
- select
- supplement
This Talk

CodeScoop [CHI '18]
- code editor
- snippets
- select
- simplify

Torii [CHI '20]
- code editor
- tutorials
- supplement
- sequence

Gathering tools [CHI '19]
- notebook
- snippets
- tutorials
- select
- supplement
This Talk

CodeScoop [CHI '18]
- code editor
- snippets
- select
- simplify

Torii [CHI '20]
- code editor
- tutorials
- supplement
- sequence

Gathering tools [CHI '19]
- notebook
- snippets
- tutorials
- select
- supplement
Notebook Model of Exploratory Programming

1. Incremental execution
Notebook Model of Exploratory Programming

1. Incremental execution
2. In-situ output
Notebook Model of Exploratory Programming

1. Incremental execution
2. In-situ output
3. Incremental changes
1. Incremental execution
2. In-situ output
3. Incremental changes
4. Control over layout
Notebook Model of Exploratory Programming

1. Incremental execution
2. In-situ output
3. Incremental changes
4. Control over layout
1. Incremental execution
2. In-situ output
3. Incremental changes
4. Control over layout
Notebook Model of Exploratory Programming

1. Incremental execution
2. In-situ output
3. Incremental changes
4. Control over layout

1 WEEK LATER

1. How did I produce this result?
Notebook Model of Exploratory Programming

1. Incremental execution
2. In-situ output
3. Incremental changes
4. Control over layout

How did I produce this?

1 WEEK LATER

1. How did I produce this result?
1. Incremental execution
2. In-situ output
3. Incremental changes
4. Control over layout

1 WEEK LATER
1. How did I produce this result?
2. Didn't I have a better version of this?
Notebook Model of Exploratory Programming

1. Incremental execution
2. In-situ output
3. Incremental changes
4. Control over layout

What can I get rid of?

1 WEEK LATER

1. How did I produce this result?
2. Didn't I have a better version of this?
3. What can I get rid of?
Disappearance
Deleted / overwritten code

Disorder
Out-of-order execution
1/2 of notebooks on GitHub [Rule et al. 2018]

Dispersion
Too many cells

Notebooks contain ugly code and dirty tricks [Rule et al. 2018]

31 / 41 surveyed participants had trouble finding prior analyses [Kery et al. 2018]
CODE GATHERING TOOLS Demo

In [10]: clusters = KMeans(n_clusters=2).fit(data).labels_

In [11]: plt.scatter(petal_length, petal_width, c=clusters)

Out[11]: <matplotlib.collections.PathCollection at 0x100beacba8>

In [1]: import matplotlib.pyplot as plt
   ...: from sklearn import datasets

In [2]: data = datasets.load_iris().data[:,2:4]
   ...: petal_length, petal_width = data[:,0], data[:,1]

In [12]: petal_length, petal_width = data[:,1], data[:,0]

In [3]: print("Average petal length: %.3f " % (sum(petal_length) / len(petal_length)))
   ...: Average petal length: 3.758
Task 1: Recovering Code

How did I produce this?
Task 1: Recovering Code

How did I produce this?

Variables

Outputs
Task 1: Recovering Code

How did I produce this?
CODE GATHERING TOOLS Demo

Task 1: Recovering Code

How did I produce this?

Request cell subset that produced the result.
**CODE GATHERING TOOLS Demo**

Task 1: Recovering Code

**How did I produce this?**

Request cell subset that produced the result.
The gathered code is...

- reduced
- ordered
- complete

Task 1: Recovering Code

How did I produce this?

Request cell subset that produced the result.
Task 1: Recovering Code

Request cell subset that produced the result.

Task 2: Comparing Versions

Didn't I have a better version of this?
CODE GATHERING TOOLS Demo

Task 1: Recovering Code
Request cell subset that produced the result.

Task 2: Comparing Versions
Didn't I have a better version of this?
Open a version browser for a result.
**Code Gathering Tools Demo**

**Task 1: Recovering Code**
Request cell subset that produced the result.

**Task 2: Comparing Versions**

Didn't I have a better version of this?

Open a version browser for a result.
**Task 1: Recovering Code**

Request cell subset that produced the result.

**Task 2: Comparing Versions**

*Didn't I have a better version of this?*

Open a version browser for a result.
**CODE GATHERING TOOLS Demo**

**Task 1: Recovering Code**
Request cell subset that produced the result.

**Task 2: Comparing Versions**

Didn't I have a better version of this?

Open a version browser for a result.
**Code Gathering Tools Demo**

**Task 1: Recovering Code**
Request cell subset that produced the result.

**Task 2: Comparing Versions**

Didn't I have a better version of this?

Open a version browser for a result.
Task 1: Recovering Code
Request cell subset that produced the result.

Task 2: Comparing Versions
Open a version browser for a result.

Task 3: Cleaning Notebook
What code can I get rid of?

CODE GATHERING TOOLS Demo
Code Gathering Tools Demo

Task 1: Recovering Code
Request cell subset that produced the result.

Task 2: Comparing Versions
Open a version browser for a result.

Task 3: Cleaning Notebook
What code can I get rid of?
... Request cell subset that produced the result.
Program Analysis: Slicing Notebooks

1 Notebook
some cells missing,
some cells out-of-order

[10]
[11]
[1]
[2]
[12]
[3]

cleaned, ordered notebooks

versioned results
Program Analysis: Slicing Notebooks

1 Notebook
some cells missing,
some cells out-of-order

2 Execution Log
all cells present, in-order

 execution time
Program Analysis: Slicing Notebooks

① Notebook
some cells missing,
some cells out-of-order

② Execution Log
all cells present, in-order

[10]
[11]
[1]
[2]
[12]
[3]

 executio n time
Program Analysis: Slicing Notebooks

1. Notebook
   some cells missing,
some cells out-of-order

2. Execution Log
   all cells present, in-order

3. Program Slices [Weiser '81]
Program Analysis: Slicing Notebooks

1 Notebook

- some cells missing,
- some cells out-of-order

2 Execution Log

- all cells present, in-order

3 Program Slices [Weiser '81]

- cleaned, ordered notebooks
  (preserve cell boundaries and outputs)

- versioned results
  (slice all cell versions)

Some cells missing, some cells out-of-order

Notebook Execution Log

· · ·

execution time

1 2 3
Evaluating Code Gathering Tools

Q. How do data analysts *distill code* during the process of exploratory data analysis?
A Qualitative Study of Gathering

Participants: \(N = 12\) professional data analysts

Cleaning Task \(\times 2\): Clean a computational notebook, with and without code gathering tools.

Exploration: Rank movies in from a movies dataset. Use code gathering tools as you wish.
Q. How do data analysts distill code during the process of exploratory data analysis?

Participants described gathering to a notebook as "beautiful" and "amazing": it "hits the nail on the head."
Some Observed Uses of Gathering Tools

"Finishing moves"

Lightweight branching

Gathering for multiple audiences

Creating personal references
Needs for distilling notebooks

Picking a subset of cells [P1-P12]...
and removing the rest [P8, P10-12].

"I picked a plot that looked interesting and, if you think of a dependency tree of cells, walked backwards and removed everything that wasn’t necessary."

... And many additional stages:

polishing visualizations  [P1, P6]  restructuring code  [P3, P4, P6, P12]
integrating with version control  [P7]
Takeaways from Study

Code gathering tools can be picked up quickly and readily applied to new use cases in notebooks.

Gathering covers an important yet incomplete set of tasks for distilling notebooks.
$ jupyter labextension install nbgather

```python
from sklearn.cluster import KMeans
from sklearn.datasets import datasets

data = datasets.load_iris().data[:,2:4]
petal_length, petal_width = data[:,0], data[:,1]

clusters = KMeans(n_clusters=2).fit(data).labels_
scatter(petal_length, petal_width, c=clusters)
```

![Scatter plot of petal length vs petal width with clusters colored](image)
Also implemented as an experimental feature in VS Code

Python > Data Science: Enable Gather

Enable code gathering for single cells in the interactive window (experimental).
Thesis

Authors can transform existing programs into sample programs more efficiently and flexibly when aided by interactive tools for selecting, simplifying, supplementing, and sequencing code.
The Essence of Distillation Tools
The Essence of Distillation Tools

Interactions:
- select
- simplify
- supplement
- sequence

Author → Existing Program → Interactions

Snippet
Tutorial
Sample Program
The Essence of Distillation Tools
The Essence of Distillation Tools

1. Interactions
   - select
   - simplify
   - supplement
   - sequence

2. Program Analysis

3. Efficient, Flexible Authoring

Author

Existing Program

Snippet

Tutorial

Sample Program
How do we help authors create more effective instructions?
A design space of distillation tools

From Chapter 3 of Andrew's dissertation. Coming soon!
A design space of distillation tools

<table>
<thead>
<tr>
<th>Output</th>
<th>Snippet</th>
<th>Notebook</th>
<th>Tutorial</th>
<th>Sample project</th>
<th>Screencast</th>
<th>Live demo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program qualities</td>
<td>executable</td>
<td>minimal</td>
<td>copyable</td>
<td>readable</td>
<td>robust</td>
<td>...</td>
</tr>
<tr>
<td>Length (lines)</td>
<td>1</td>
<td>10</td>
<td>100</td>
<td>1,000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time spent</td>
<td>1 min</td>
<td>10 min</td>
<td>1 hour</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Programming environment</th>
<th>code editor</th>
<th>notebook</th>
<th>dedicated editor</th>
<th>non-interactive</th>
</tr>
</thead>
</table>

### GOALS
- **TOOL BASICS**
- **GOALS**
- **SELECT**
- **SIMPLIFY**
- **SUPPLEMENT**
- **SEUENCE**

#### # versions
- 1
- 10 checkpoints
- 100 continuous recording

#### Snippet execution order
- Sequential
- Infer from source program
- Infer from dependencies
- Hard-coded order

#### Linked edits
- Source program
- Snippets
- Outputs
- Explanations

#### Program embellishments
- Guards
- Program alternatives
- Example data
- Exception handling
- Log statements

#### Assets
- Jupyter notebook
- PDF
- HTML
- CSS
- Images

#### Interactivity for readers
- Visualizations
- Live editing
- Exercises
- Version browser

#### # versions
- 1
- 10 checkpoints
- 100 continuous recording

#### Snippet execution order
- Sequential
- Infer from source program
- Infer from dependencies
- Hard-coded order

#### Linked edits
- Source program
- Snippets
- Outputs
- Explanations

**SEQUENCE**

from Chapter 3 of Andrew's dissertation. Coming soon!
## Tool Basics

<table>
<thead>
<tr>
<th>Output</th>
<th>Snippet</th>
<th>Notebook</th>
<th>Tutorial</th>
<th>Sample project</th>
<th>Screen cast</th>
<th>Live demo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program qualities</td>
<td>executable</td>
<td>minimal</td>
<td>copyable</td>
<td>readable</td>
<td>robust</td>
<td>...</td>
</tr>
<tr>
<td>Length (lines)</td>
<td>1 min</td>
<td>10 min</td>
<td>1 hour</td>
<td>1,000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time spent</td>
<td>1 min</td>
<td>10 min</td>
<td>1 hour</td>
<td>1,000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Goals

<table>
<thead>
<tr>
<th>Programming environment</th>
<th>code editor</th>
<th>notebook</th>
<th>dedicated editor</th>
<th>non-interactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authoring role</td>
<td>Autonomous</td>
<td>Unsolicited reporting</td>
<td>Subdialogue initiation</td>
<td>Fixed subtask initiative</td>
</tr>
<tr>
<td>Author selections</td>
<td>Select statements</td>
<td>Select outputs</td>
<td>Interact with running program</td>
<td>None</td>
</tr>
<tr>
<td>Slicing assistance</td>
<td>Slice</td>
<td>Interactive slice expansion</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Slicing domain</td>
<td>Interpreter history</td>
<td>One file, one language</td>
<td>Multiple files</td>
<td>Multiple languages</td>
</tr>
<tr>
<td>Help authors replace...</td>
<td>Statements</td>
<td>Classes</td>
<td>Entire programs</td>
<td>External services</td>
</tr>
<tr>
<td>Simplification techniques</td>
<td>Rename identifiers</td>
<td>Insert placeholders</td>
<td>Synthesize equivalent code</td>
<td>Generate stubs</td>
</tr>
<tr>
<td># simplification options</td>
<td>1</td>
<td>10</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Simulation options</td>
<td>1</td>
<td>10</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

## Select

<table>
<thead>
<tr>
<th>Program embellishments</th>
<th>Guards</th>
<th>Program alternatives</th>
<th>Example data</th>
<th>Exception handling</th>
<th>Log statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assets</td>
<td>Rich text</td>
<td>Diagrams</td>
<td>Console output</td>
<td>Charts / graphs</td>
<td>Screenshots</td>
</tr>
<tr>
<td>Interactivity for readers</td>
<td>Visualizations</td>
<td>Live editing</td>
<td>Exercises</td>
<td>Version browser</td>
<td></td>
</tr>
<tr>
<td># versions</td>
<td>1</td>
<td>10</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Snippet execution order</td>
<td>Sequential</td>
<td>Differ from source program</td>
<td>Differ from dependencies</td>
<td>Hard-coded order</td>
<td></td>
</tr>
<tr>
<td>Linked edits</td>
<td>Source program</td>
<td>Snippets</td>
<td>Outputs</td>
<td>Explanations</td>
<td></td>
</tr>
</tbody>
</table>

## Supplement

| Source program | Snippets | Outputs | Explanations |

## Sequence

### A design space of distillation tools

from Chapter 3 of Andrew's dissertation. Coming soon!
A design space of distillation tools

from Chapter 3 of Andrew's dissertation. Coming soon!
A design space of distillation tools

Mixed-initiative program synthesis

The design of explorable tutorials

Natural language generation

from Chapter 3 of Andrew's dissertation. Coming soon!
The distillation of explorables

In the next section, you will start combining these Core Components to learn about how React works. Have a play with them here now!

```javascript
import React from 'react';
import { View, Text, Image, ScrollView, TextInput } from 'react-native';

export default function App() {
  return (
    <ScrollView>
      <Text>Some text</Text>
      <Text>Some more text</Text>
      <Image source={"https://reactnative.dev/docs/assets/p_cat2.png"} style={{width: 100, height: 200}}/>
      <TextInput
        style={{
          height: 40,
          borderColor: 'black',
          borderWidth: 1
        }}
        defaultValue="You can type in me"
      />
    </ScrollView>
  );
}
```

Because React Native uses the same API structure as React components, you'll need to understand React component APIs to get started. The next section makes for a quick introduction or refresher on

Empirical Questions

What are effective patterns for creating explroables?

Technical Questions

How can tools help authors distill programs into instructive explroables?
The distillation of scientific discourse

In the distillation of scientific discourse, a common task is to predict the next token given the history $(t_1, ..., t_{k-1})$:

$$p(t_k | t_1, t_2, ..., t_{k-1}) = \prod_{k=1}^{N} p(t_k | t_1, t_2, ..., t_{k-1}).$$

Recent state-of-the-art neural language models (Józefowicz et al., 2016; Melis et al., 2017; Merity et al., 2017) compute a context-independent token representation $x_{k,j}^{LM}$ (via token embeddings or a CNN over characters) then pass it through $L$ layers of forward LSTMs. At each position $k$, each LSTM layer outputs a context-dependent representation $h_{k,j}^{LM}$ where $j = 1, ..., L$. The top layer LSTM output, $h_{k,L}^{LM}$, is used to predict the next token $t_{k+1}$ with a Softmax layer.

A backward LM is similar to a forward LM, except it runs over the sequence in reverse, predicting the previous token given the future context:

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_{k+1}, t_{k+2}, ..., t_N).$$

For each token $t_k$, a $L$-layer biLM computes a set of $2L+1$ representations:

$$R_k = \{x_k^{LM}, h_{k,j}^{LM}, h_{k,j}^{LM} | j = 1, ..., L\} = \{h_{k,j}^{LM} | j = 0, ..., L\},$$

where $h_{k,0}^{LM}$ is the token layer and $h_{k,j}^{LM} = [h_{k,j}^{LM}; h_{k,j}^{LM}]$ for each biLSTM layer.

For inclusion in a downstream model, ELMo collapses all layers in $R$ into a single vector, $\text{ELMo}_k = E(R_k; \Theta_k)$. In the simplest case, ELMo just selects the top layer, $E(R_k) = h_{k,L}^{LM}$, as in TagLM (Peters et al., 2017) and CoVe (McCann et al., 2017). More generally, we compute a task specific weighting of all biLM layers:

$$\text{ELMo}_{k}^{\text{task}} = E(R_k; \Theta^{\text{task}}) = \gamma^{\text{task}} \sum_{j=0}^{L} s_{j}^{\text{task}} h_{k,j}^{LM}.$$  \hspace{1cm} (1)

In (1), $s_{j}^{\text{task}}$ are softmax-normalized weights and the scalar parameter $\gamma^{\text{task}}$ allows the task model to

...
A plug for my dissertation

Chapter 2: How programmers read, use, write, and should write sample programs.

Chapter 3: A design space of tools for authoring and presenting programs, from the '80s until now.

Chapters 4-6 describe the projects from this talk.
Thank you to my mentors and collaborators!

From Berkeley to Microsoft Research, Google, AI2, and beyond.

Björn

Marti
Thank you to my mentors and collaborators!

From Berkeley to Microsoft Research, Google, AI2, and beyond.
Questions?

Andrew Head
UC Berkeley
"... in practice, experience shows that it is very unlikely that the output of a computer will ever be more readable than its input, except in such trivial but important aspects as improved indentation."