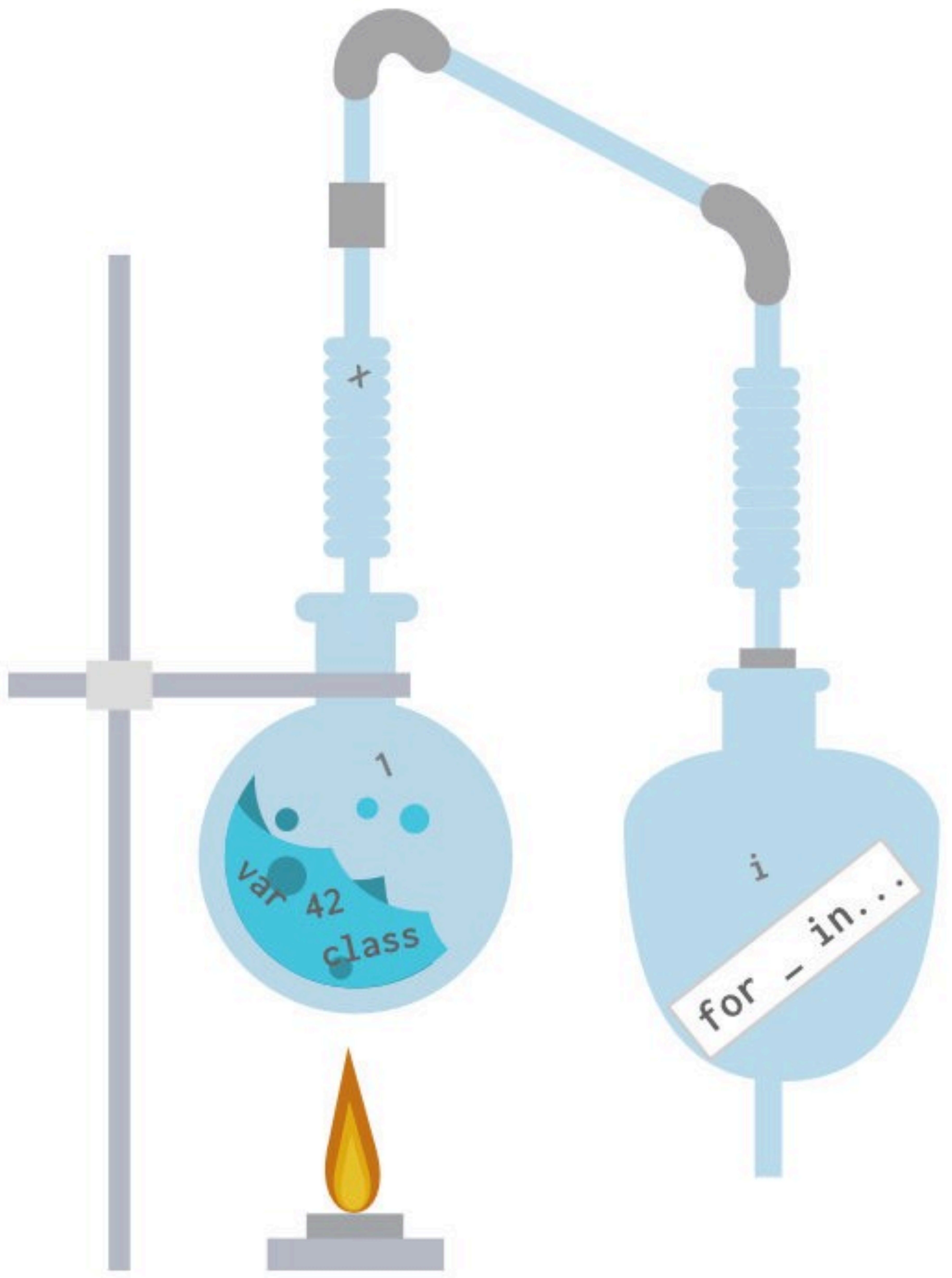
Interactive Program Distillation

Andrew Head
UC Berkeley

This talk will begin at 10:03am PST.



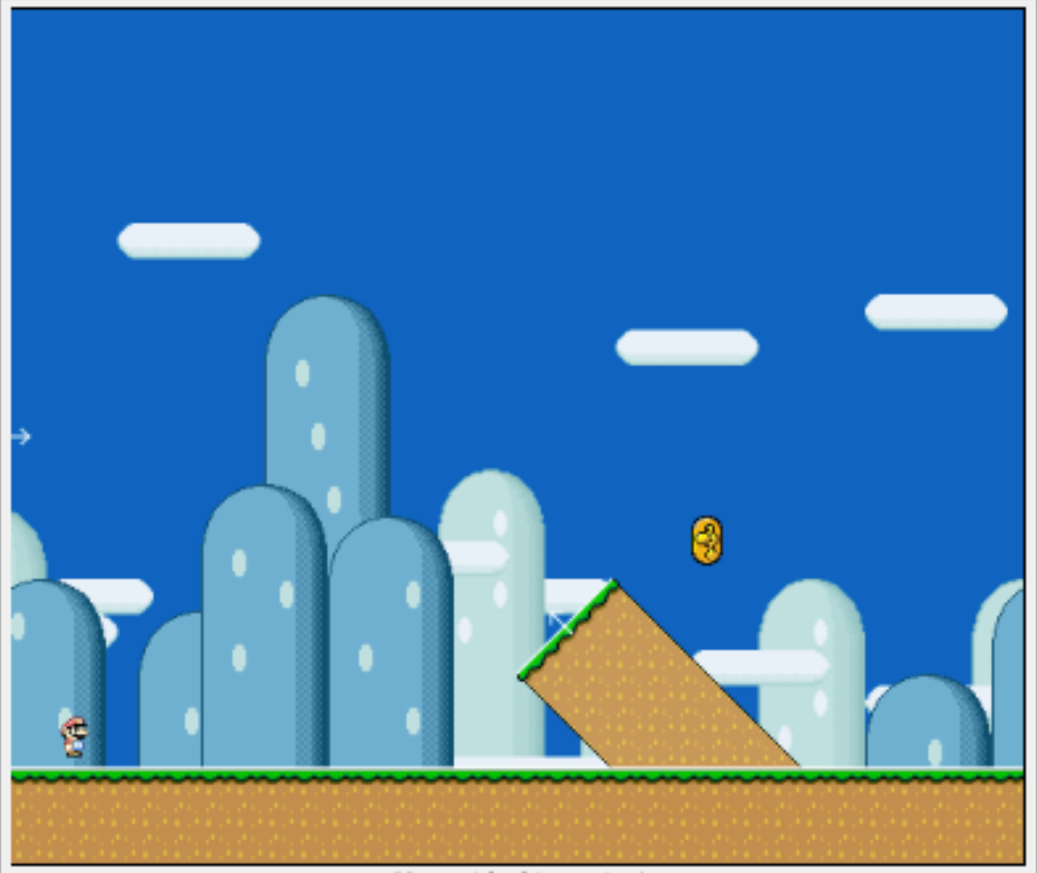
A sample program

```
// the dragon coin pickup
class DragonCoin extends MarioPickup {
  DragonCoin(float x, float y) {

super("Dragon coin", "graphics/assorted/Dragon-coin.gif", 1, 10, x, y, t
  }
}
```

That's it, that's all we have to define. Now if we want to use them in our game, we simply make `[n` `Coin(...,...)]` or `[new DragonCoin(...,...)]` and then add them to our list of pickups, fc the player only:

```
class MarioLayer extends LevelLayer {
  [...]
  MarioLayer(Level owner) {
    [...]

    // add coints 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 handy function for placing lots of coins
  void addCoins(float x, float y, float w) {
    float step = 16, i = 0, last = w/step;
    for(i=0; i
      addForPlayerOnly(new Coin(x+8+i*step,y));
    }
  }
}
```

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, usir 16 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 `[addForPlayerOnly(...)]` function name. The result? Why, let's play our updated game and see for ourselves:



Now with shiny coins!

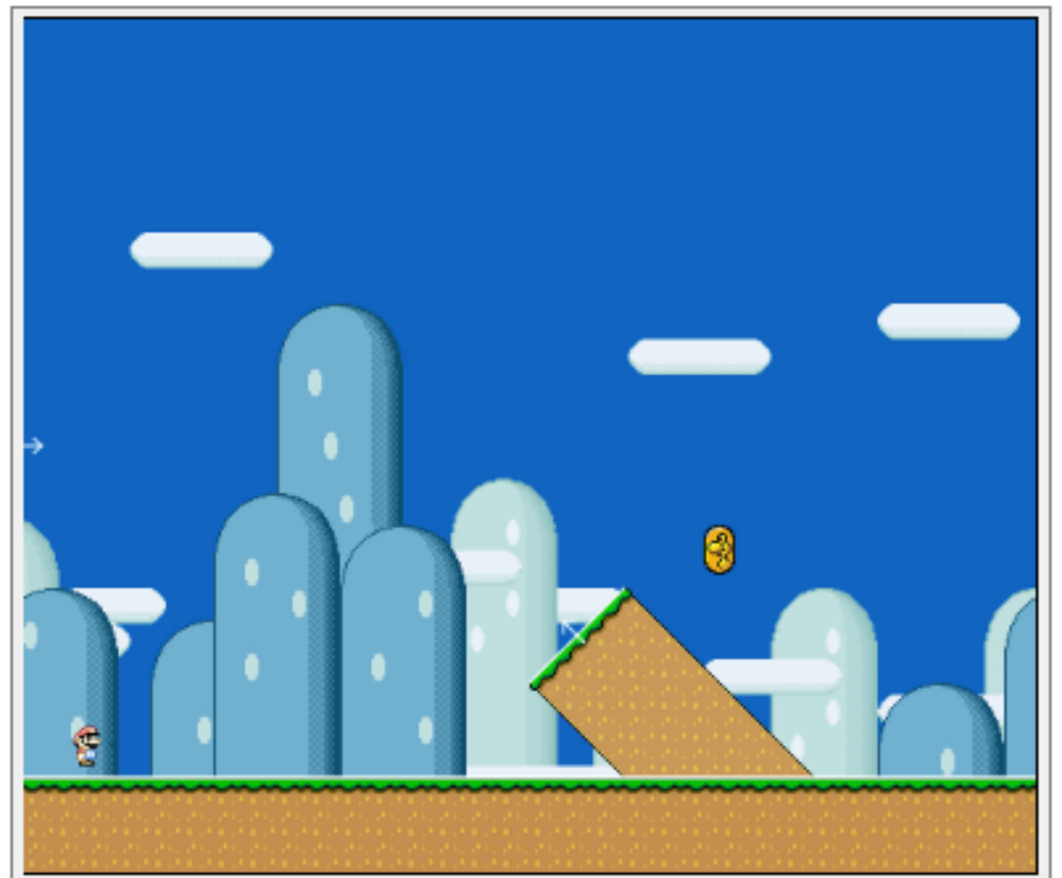
A sample program

```
// the dragon coin pickup
class DragonCoin extends MarioPickup {
  DragonCoin(float x, float y) {
  super("Dragon coin", "graphics/assorted/Dragon-coin.gif", 1, 10, x, y, t
  }
}
```

That's it, that's all we have to define. Now if we want to use them in our game, we simply make [n Coin(...,...)] or [new DragonCoin(...,...)] and then add them to our list of pickups, fo the player only:

```
class MarioLayer extends LevelLayer {
  [...]
  MarioLayer(Level owner) {
    [...]

    // add coints 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 handy function for placing lots of coins
  void addCoins(float x, float y, float w) {
    float step = 16, i = 0, last = w/step;
    for(i=0; i
      addForPlayerOnly(new Coin(x+8+i*step,y));
    }
  }
}
```

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, usir 16 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 [addForPlayerOnly(...)] function name. The result? Why, let's play our updated game and see for ourselves:



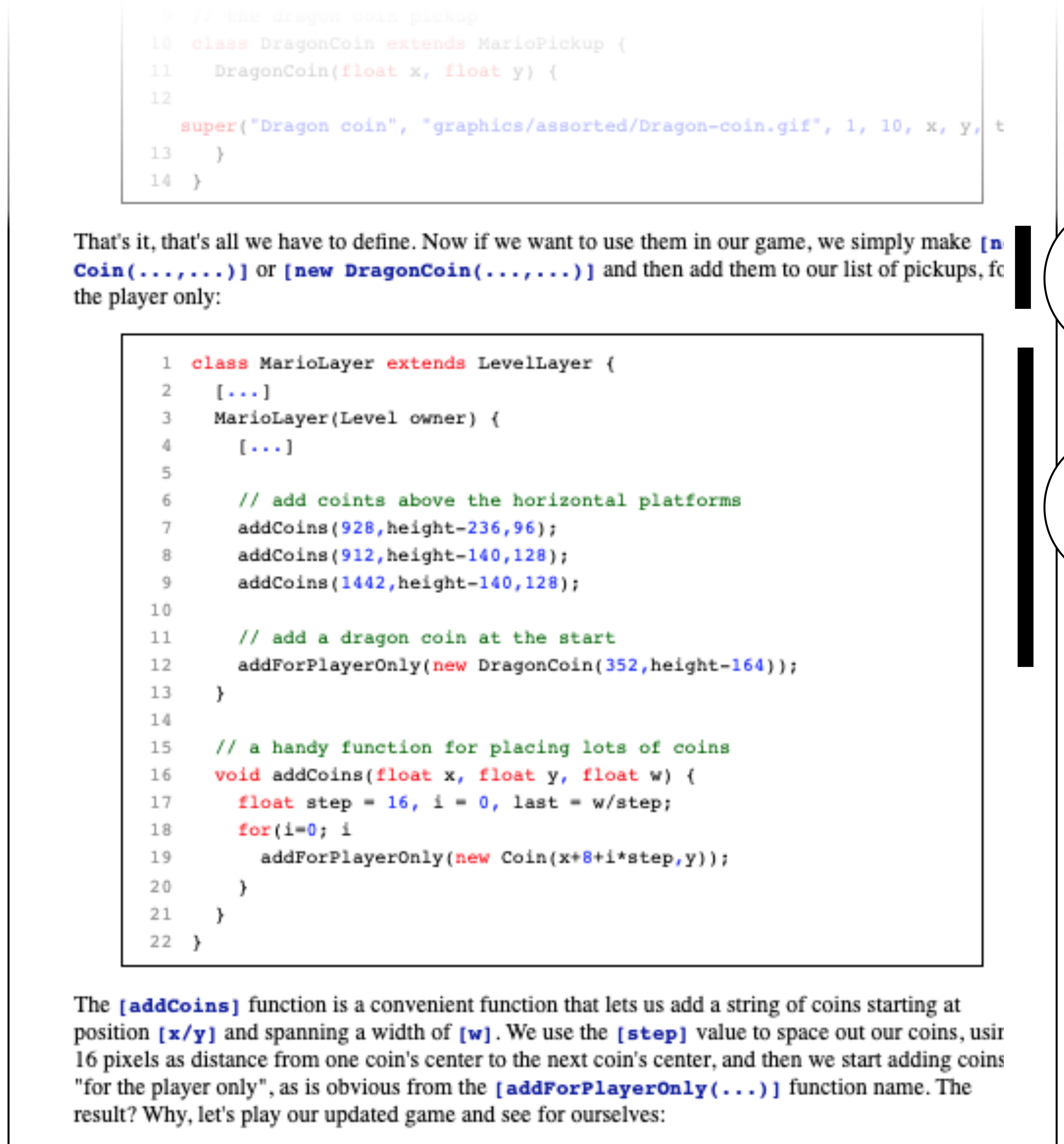
Now with shiny coins!

1

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:

A sample program

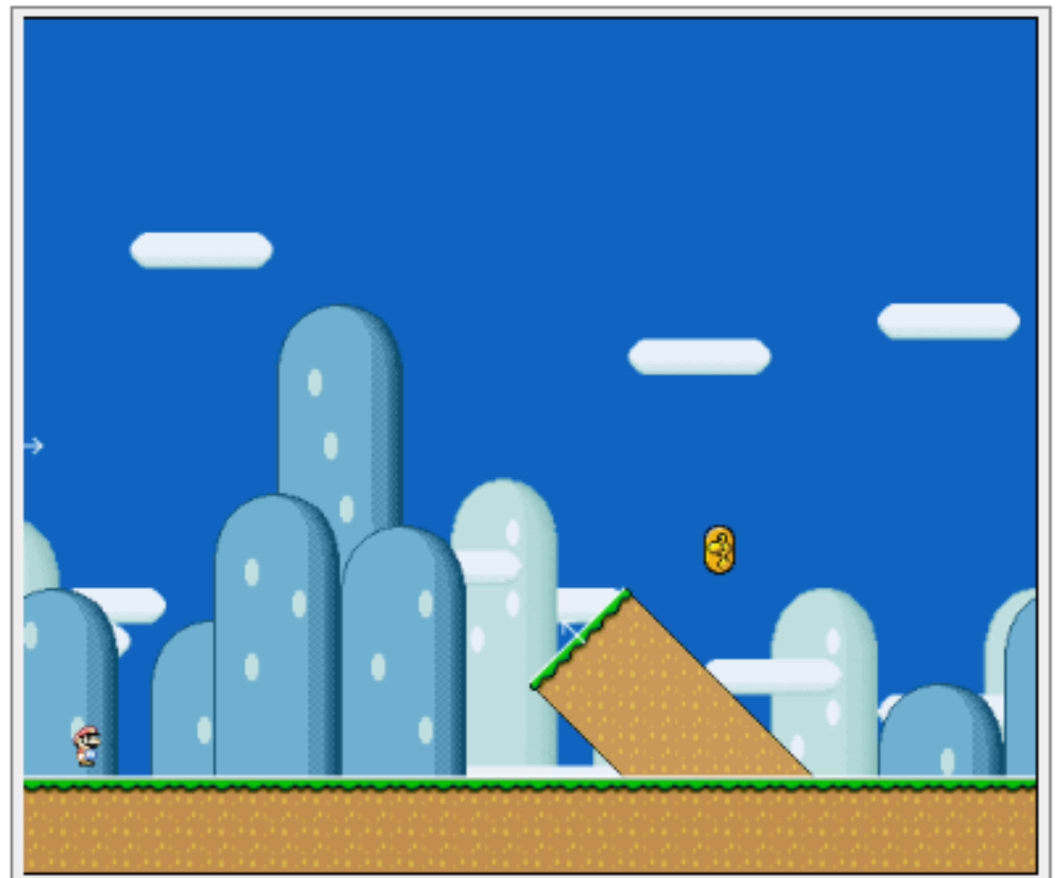


```
9   // the dragon coin pickup
10  class DragonCoin extends MarioPickup {
11    DragonCoin(float x, float y) {
12
  super("Dragon coin", "graphics/assorted/Dragon-coin.gif", 1, 10, x, y, t
13    }
14  }
```

That's it, that's all we have to define. Now if we want to use them in our game, we simply make [n Coin(...,...)] or [new DragonCoin(...,...)] and then add them to our list of pickups, fc the player only:

```
class MarioLayer extends LevelLayer {
  [...]
  MarioLayer(Level owner) {
    [...]

    // add coints 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 handy function for placing lots of coins
  void addCoins(float x, float y, float w) {
    float step = 16, i = 0, last = w/step;
    for(i=0; i
      addForPlayerOnly(new Coin(x+8+i*step,y));
    }
  }
}
```

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, usir 16 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 [addForPlayerOnly(...)] function name. The result? Why, let's play our updated game and see for ourselves:



Now with shiny coins!

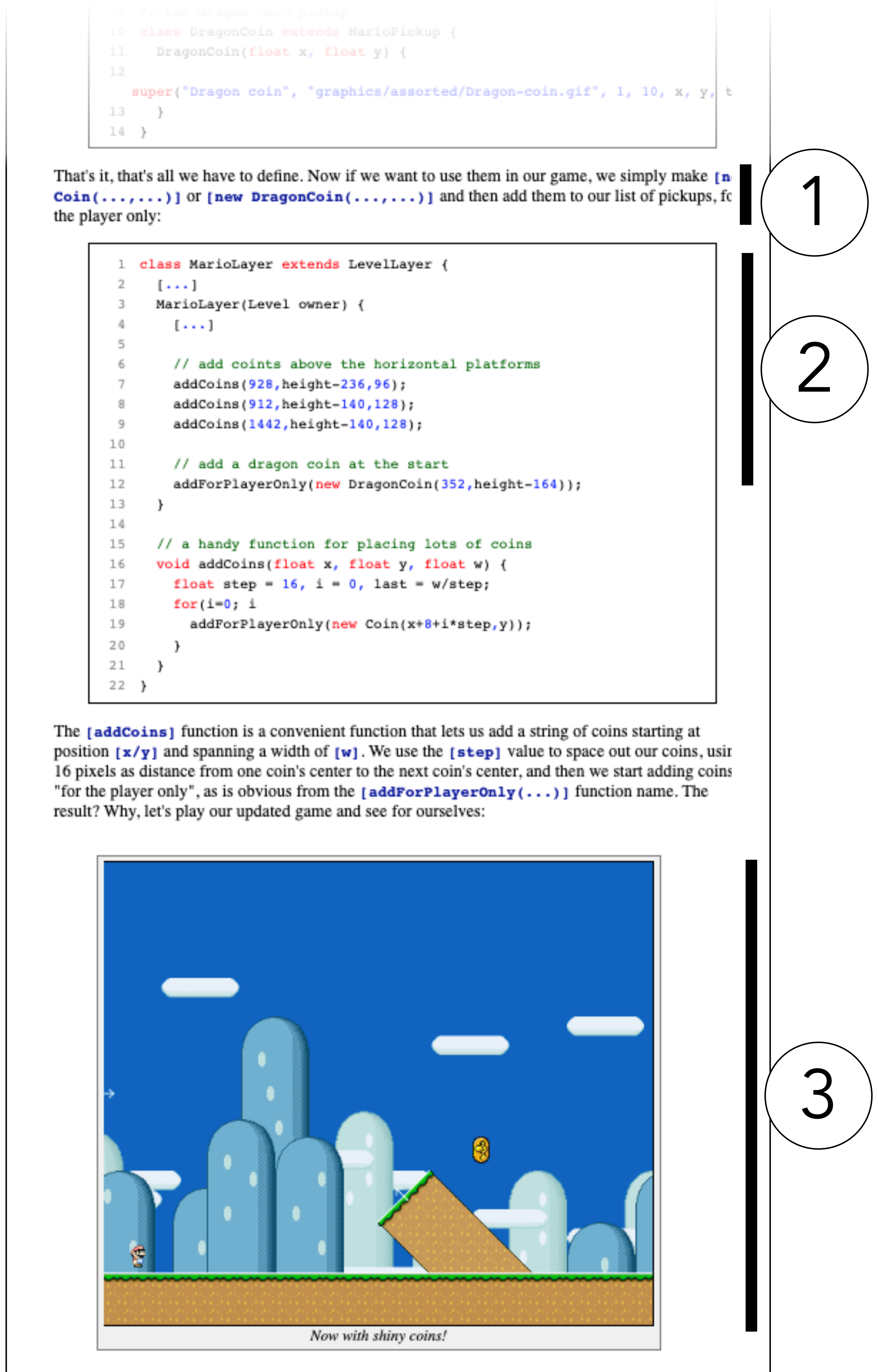
1. Written instructions

... 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

```
class MarioLayer extends LevelLayer {
  [...]
  MarioLayer(Level owner) {
    [...]

    // add coints 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



```
// the dragon coin pickup
class DragonCoin extends MarioPickup {
  DragonCoin(float x, float y) {

super("Dragon coin", "graphics/assorted/Dragon-coin.gif", 1, 10, x, y, t
  }
}
```

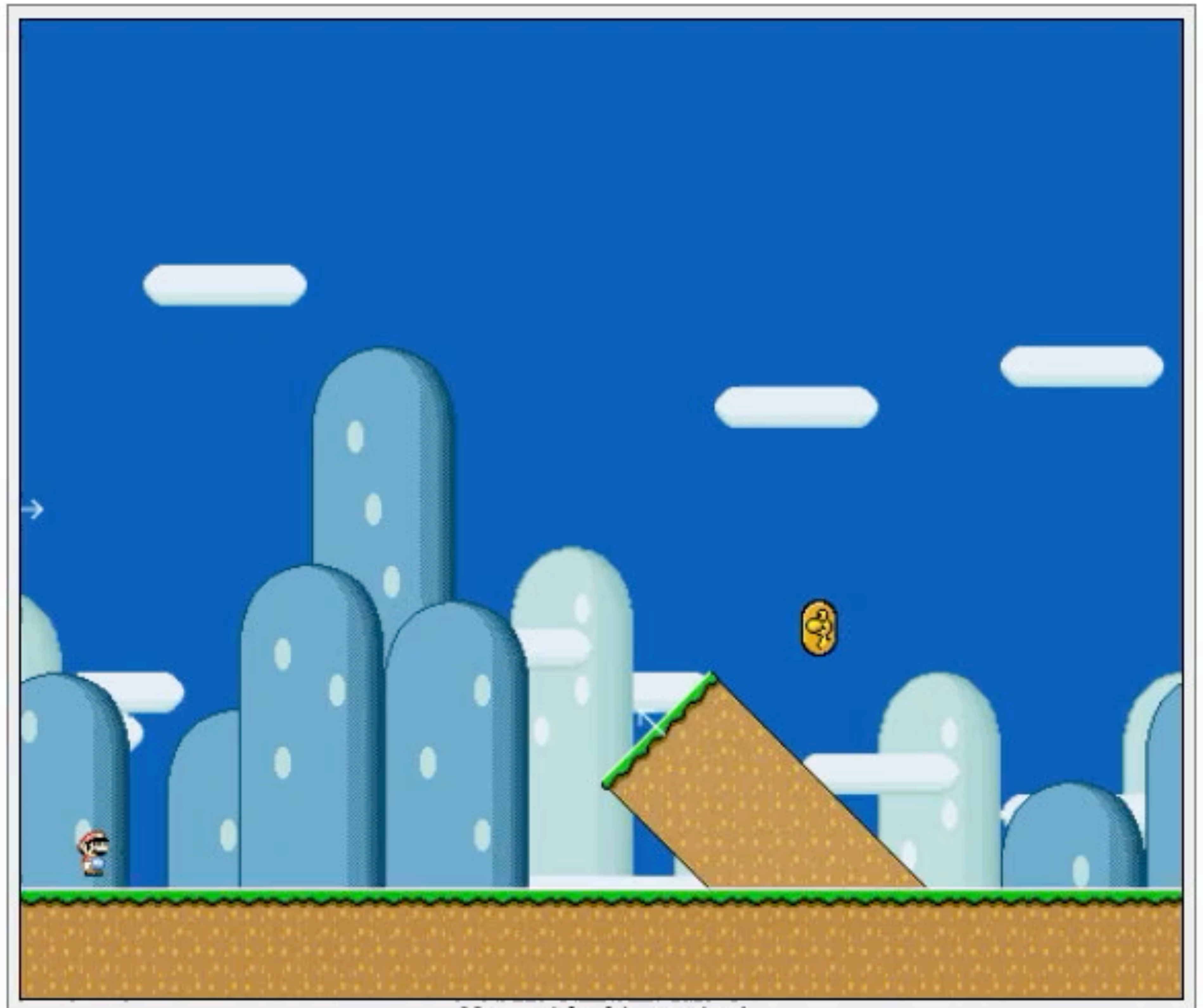
That's it, that's all we have to define. Now if we want to use them in our game, we simply make [n Coin(...,...)] or [new DragonCoin(...,...)] and then add them to our list of pickups, fc the player only:

```
class MarioLayer extends LevelLayer {
  [...]
  MarioLayer(Level owner) {
    [...]

    // add coints 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 handy function for placing lots of coins
  void addCoins(float x, float y, float w) {
    float step = 16, i = 0, last = w/step;
    for(i=0; i
      addForPlayerOnly(new Coin(x+8+i*step,y));
    }
  }
}
```

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, usir 16 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 [addForPlayerOnly(...)] function name. The result? Why, let's play our updated game and see for ourselves:

Now with shiny coins!

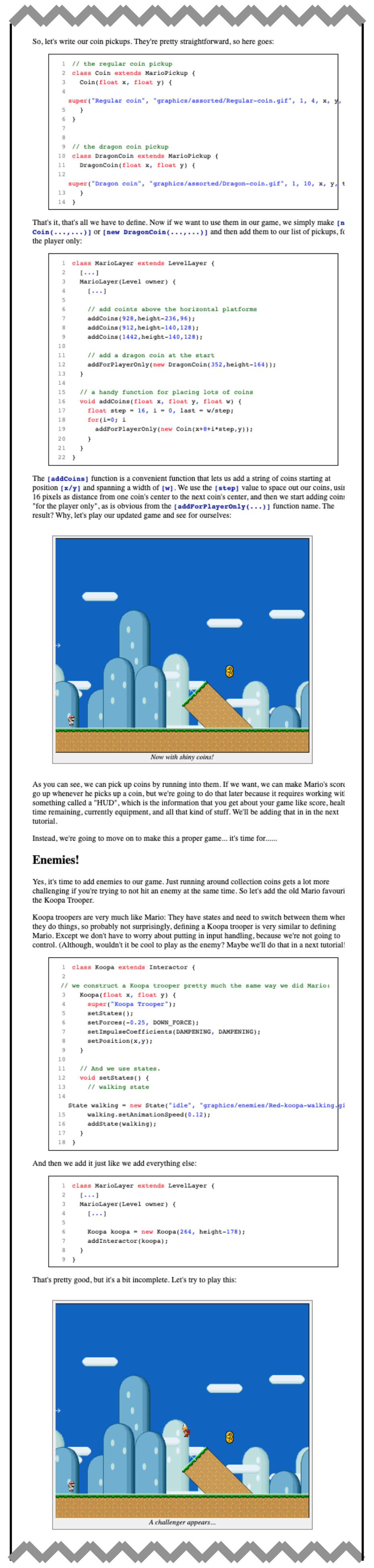
1. **Written instructions**
2. **Code snippets**
3. **Expected results**



Now with shiny coins!

How do sample programs get written?

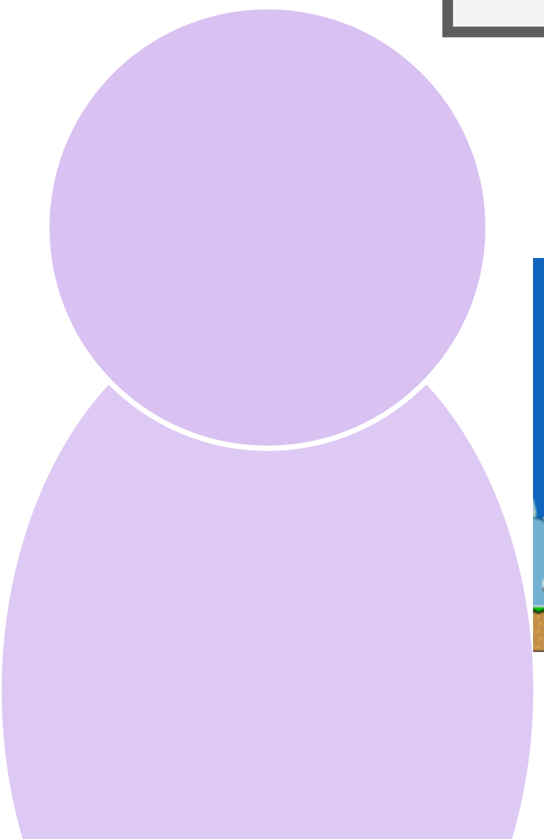
41 snippets,
22 outputs



How do sample programs get written?

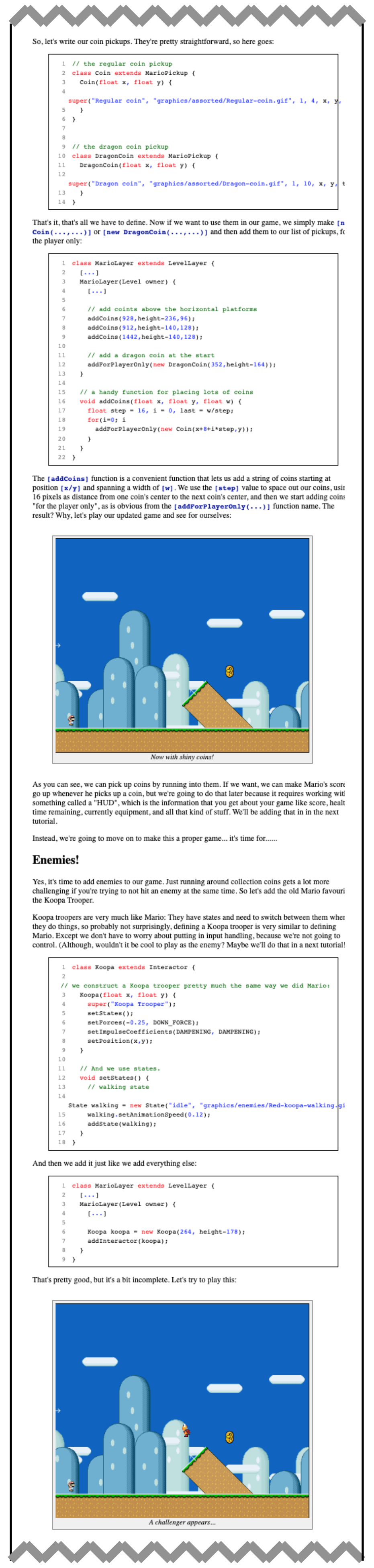
source program

output

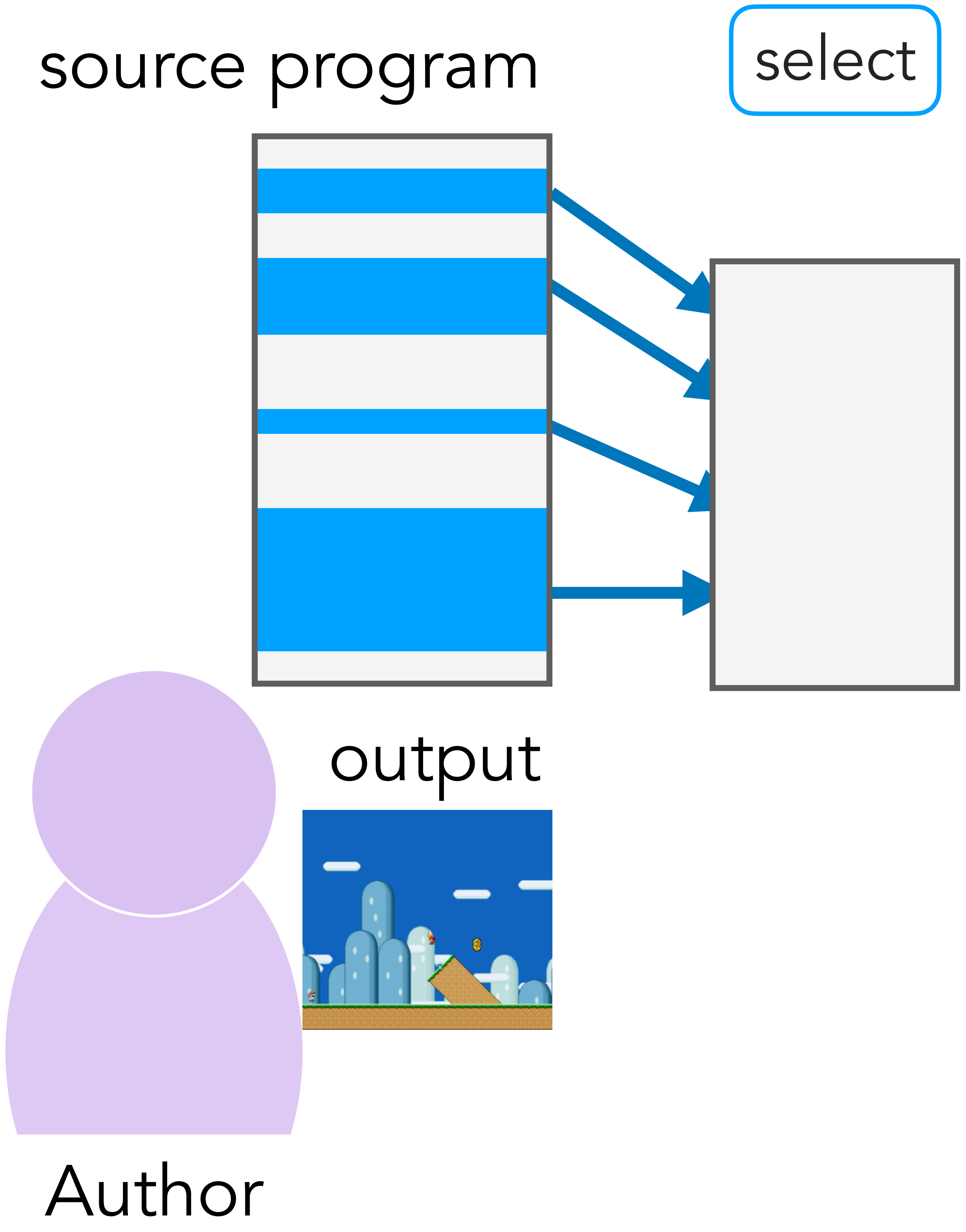


Author

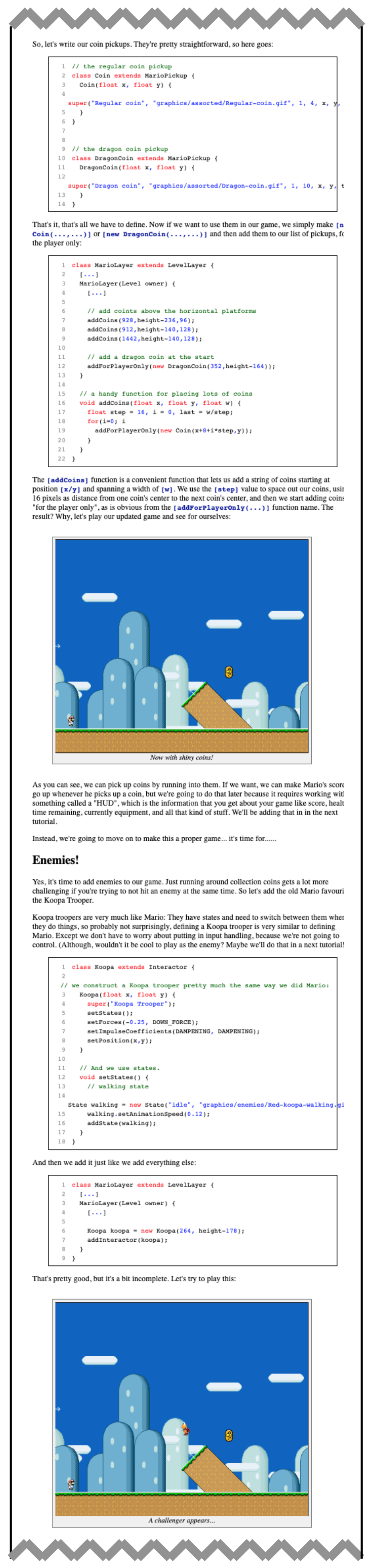
41 snippets,
22 outputs



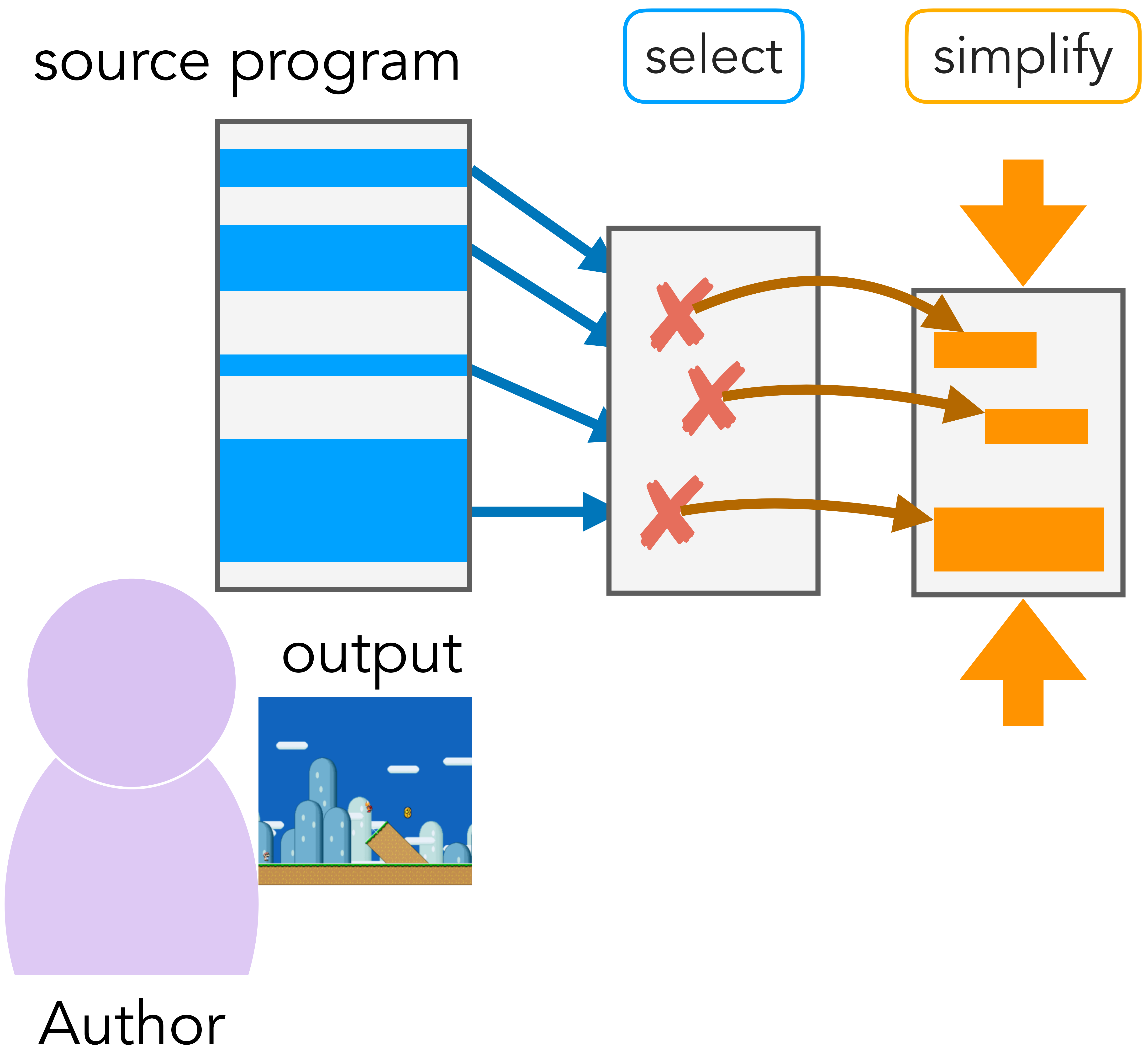
How do sample programs get written?



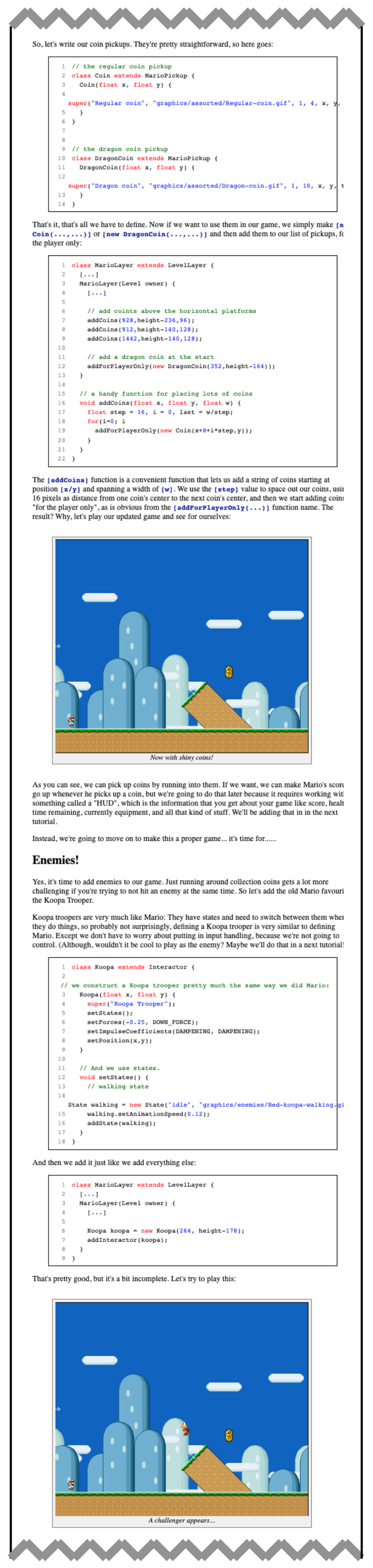
41 snippets,
22 outputs



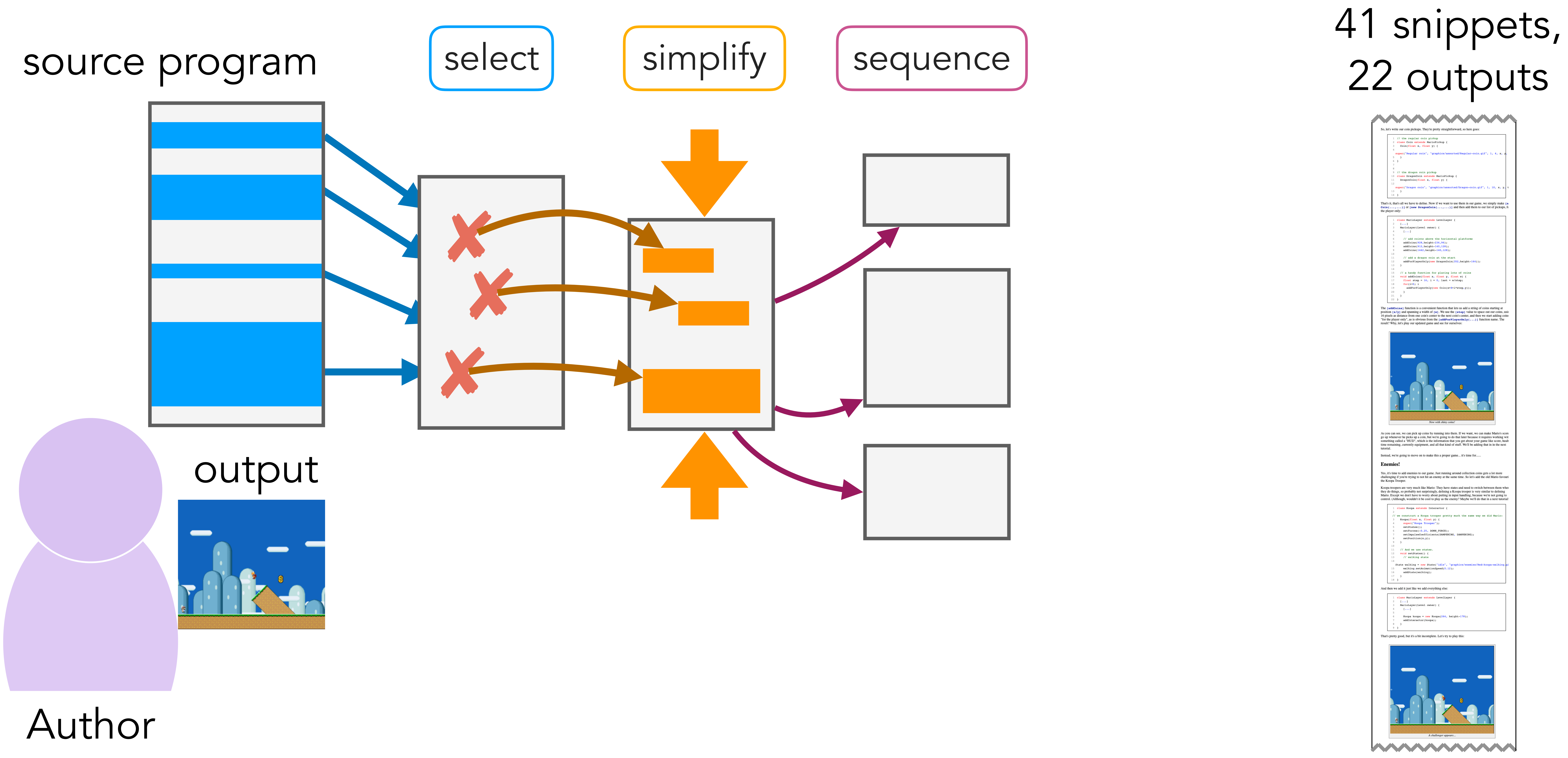
How do sample programs get written?



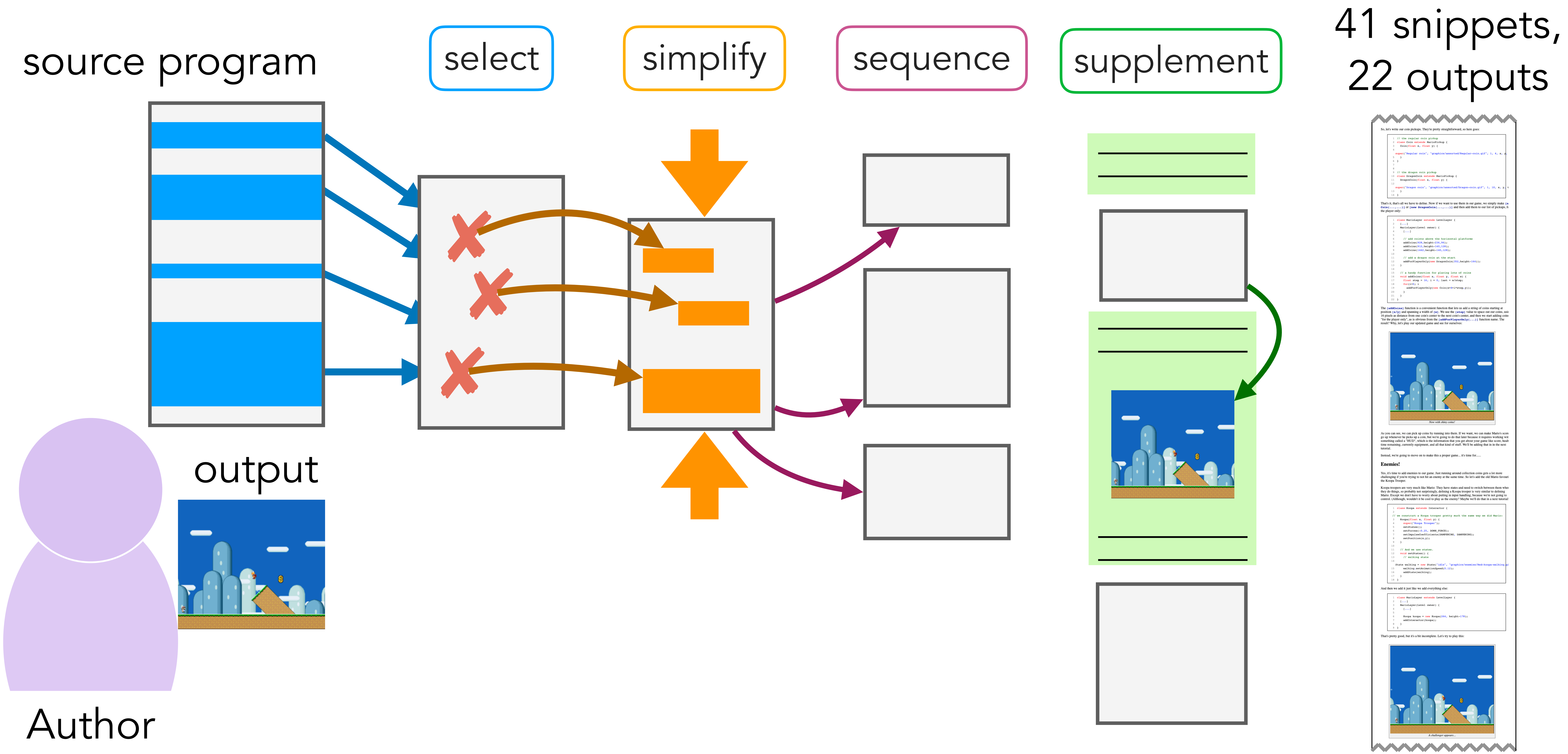
41 snippets,
22 outputs



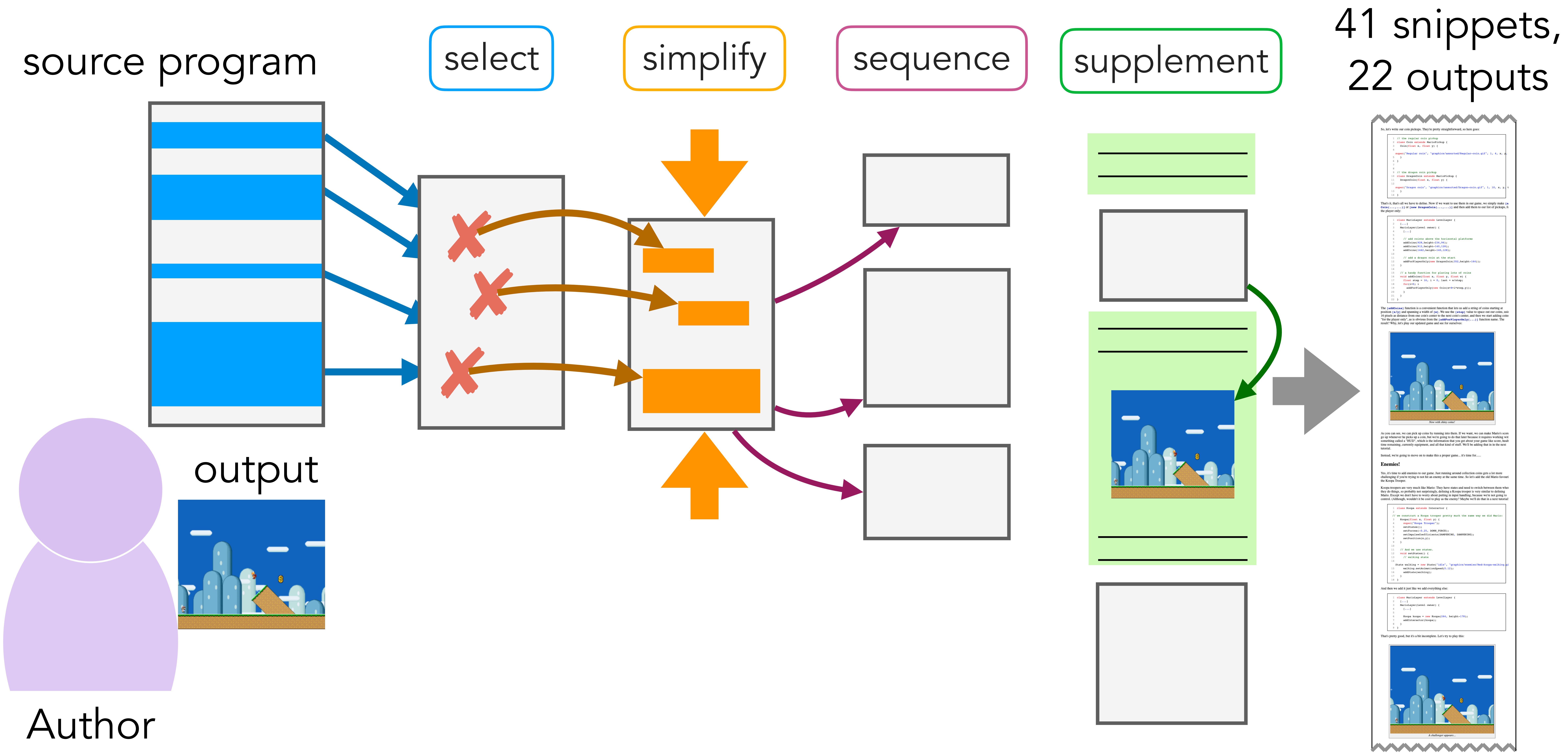
How do sample programs get written?



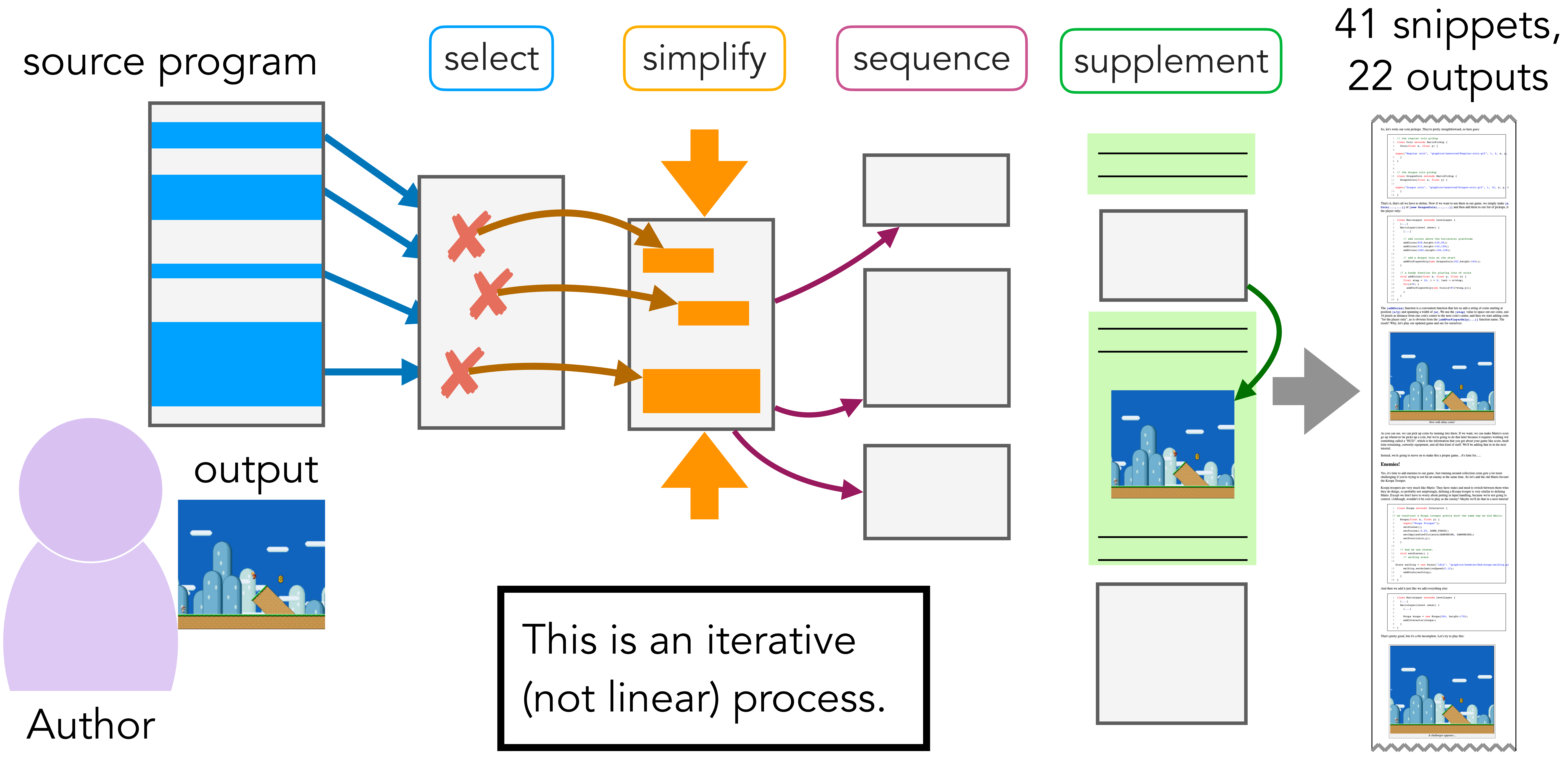
How do sample programs get written?



How do sample programs get written?

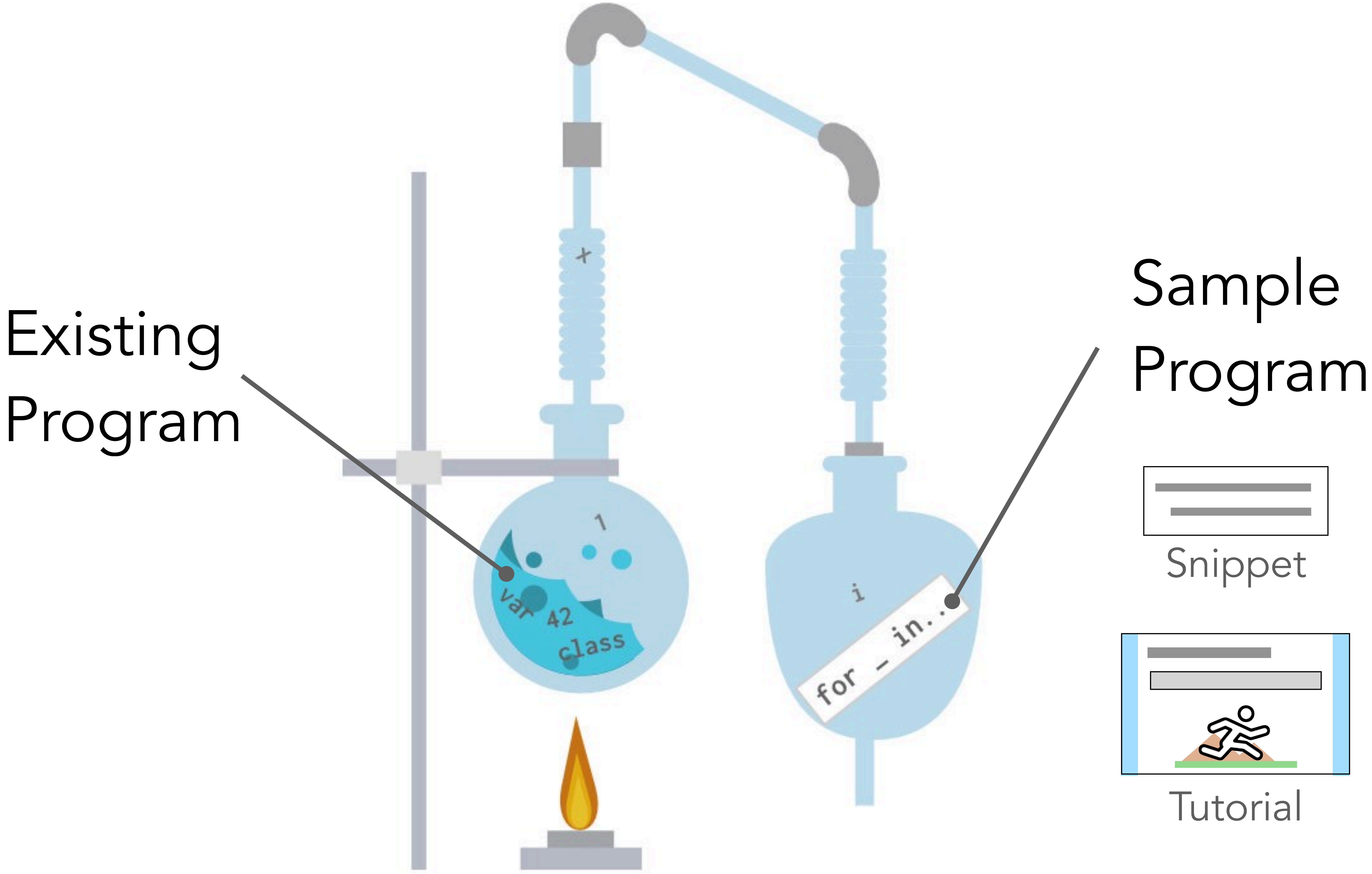


How do sample programs get written?

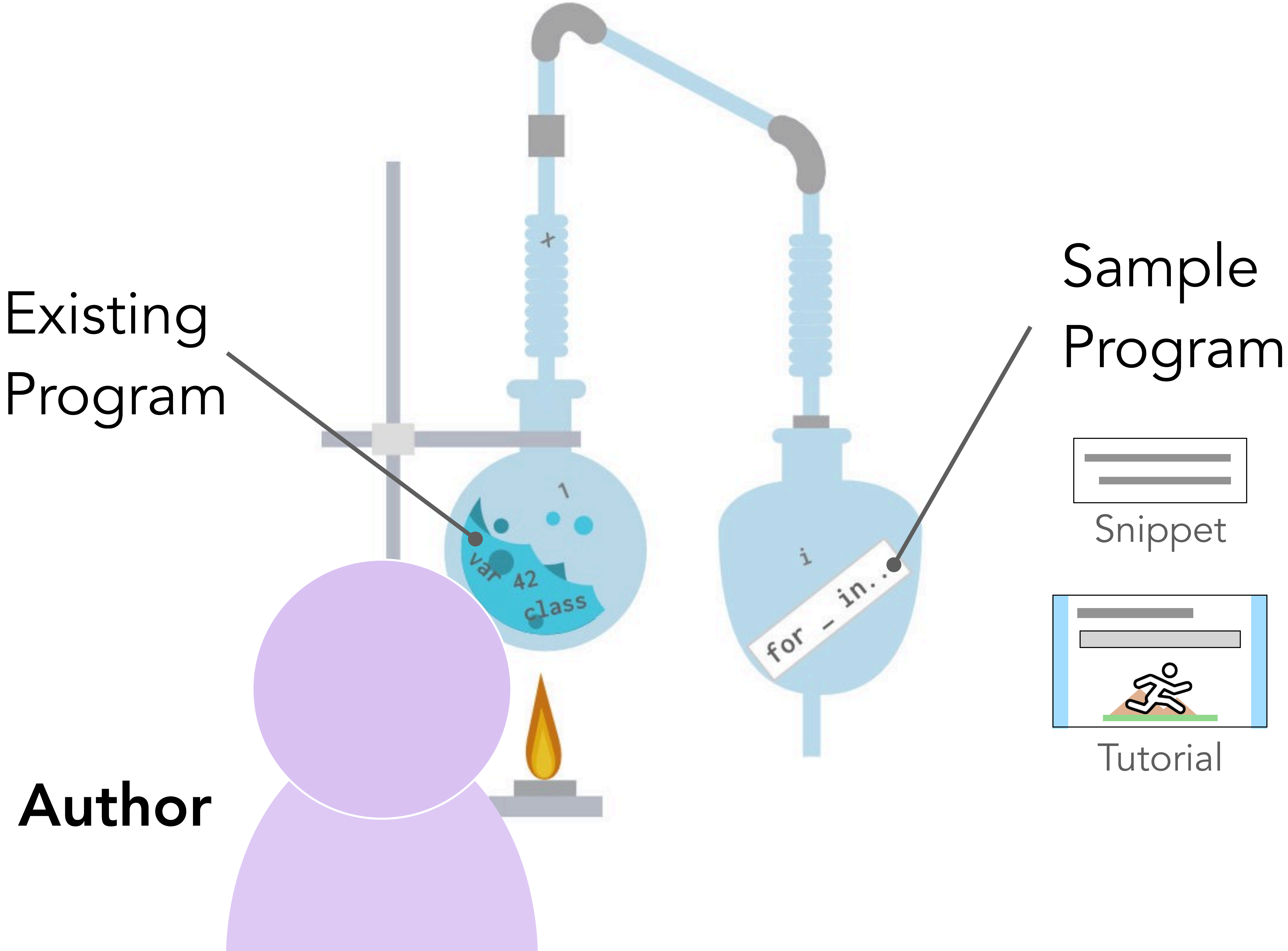


This is an iterative (not linear) process.

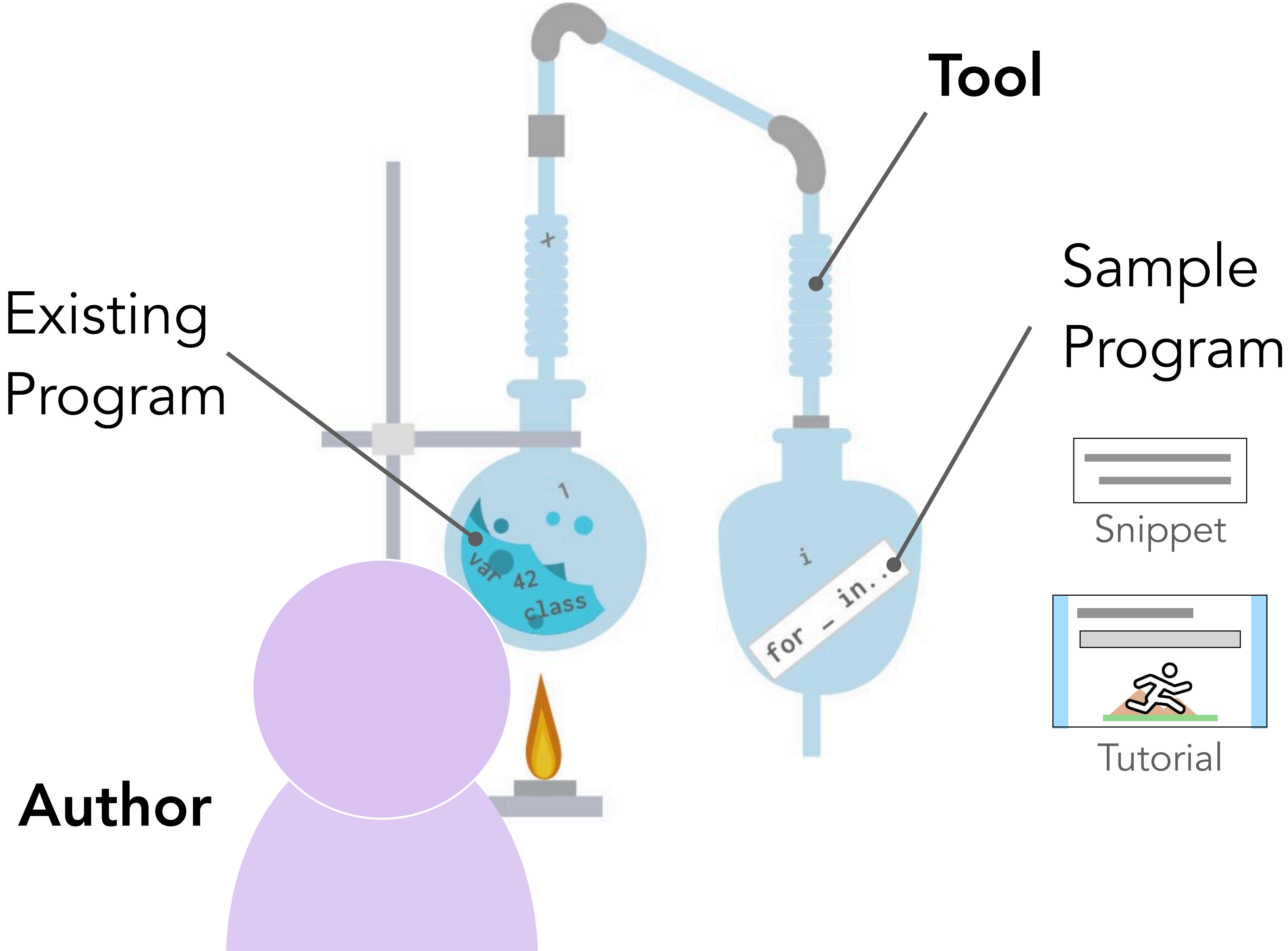
Program distillation



Program distillation



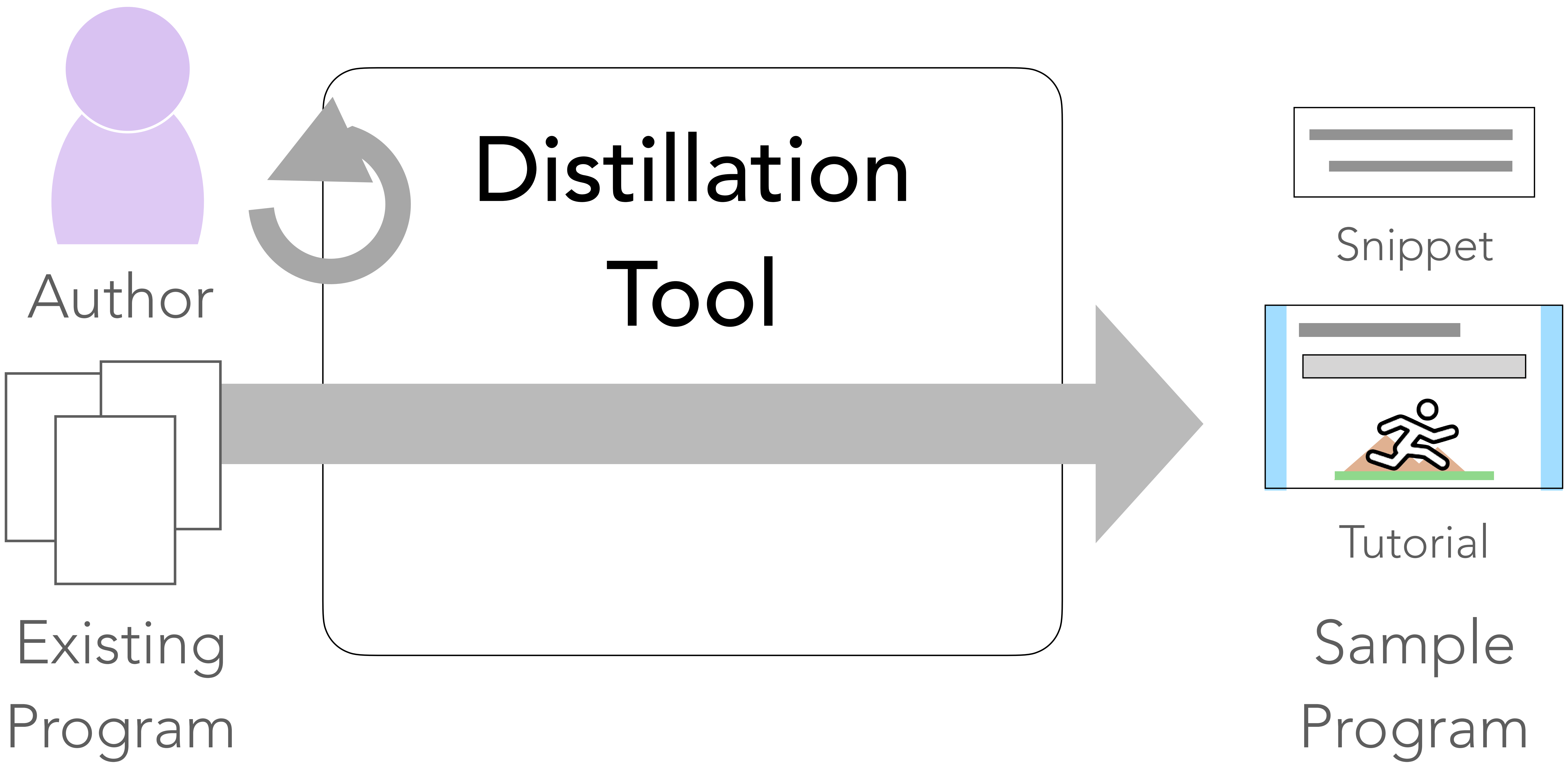
Program distillation



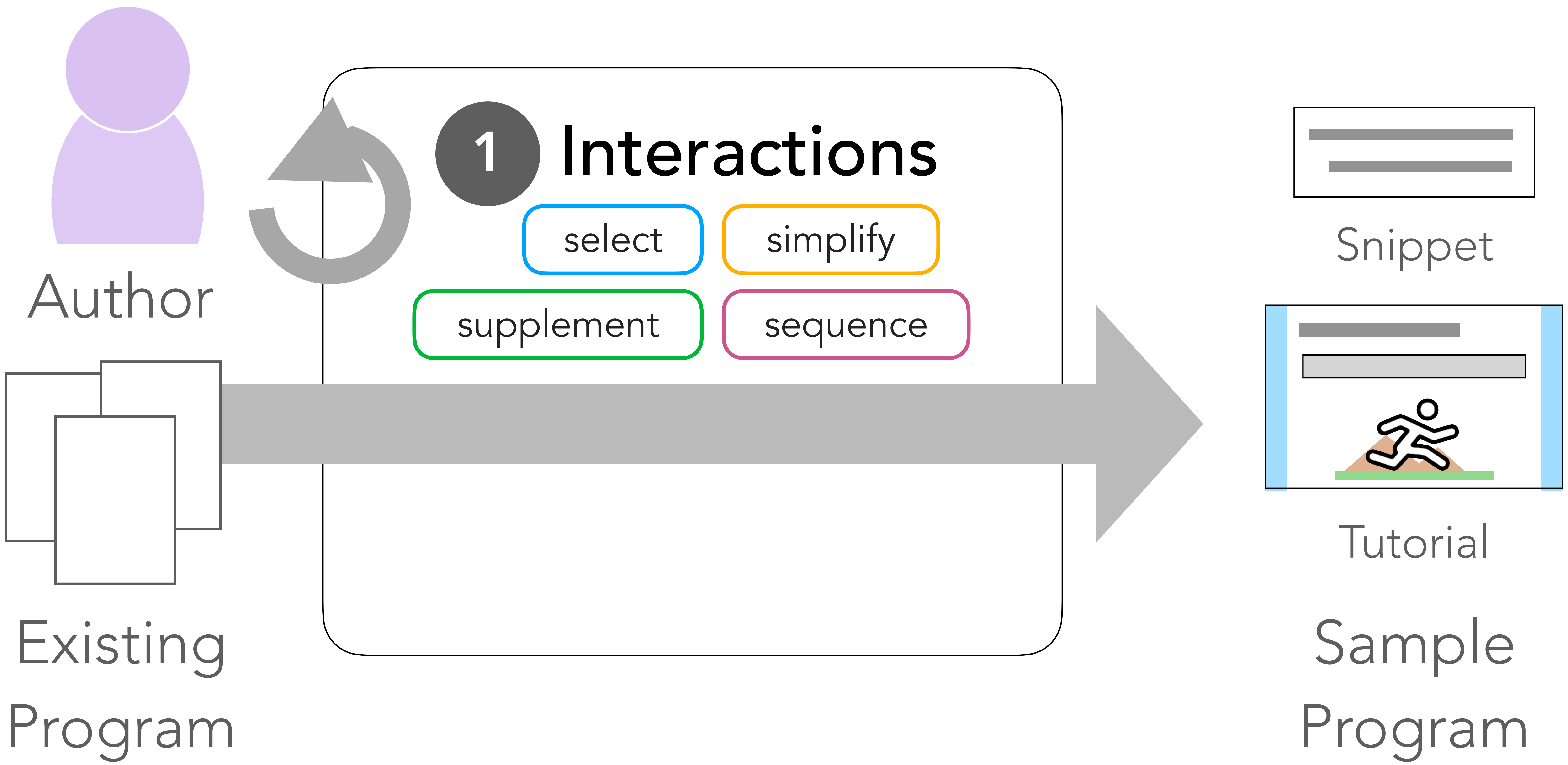
Thesis

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

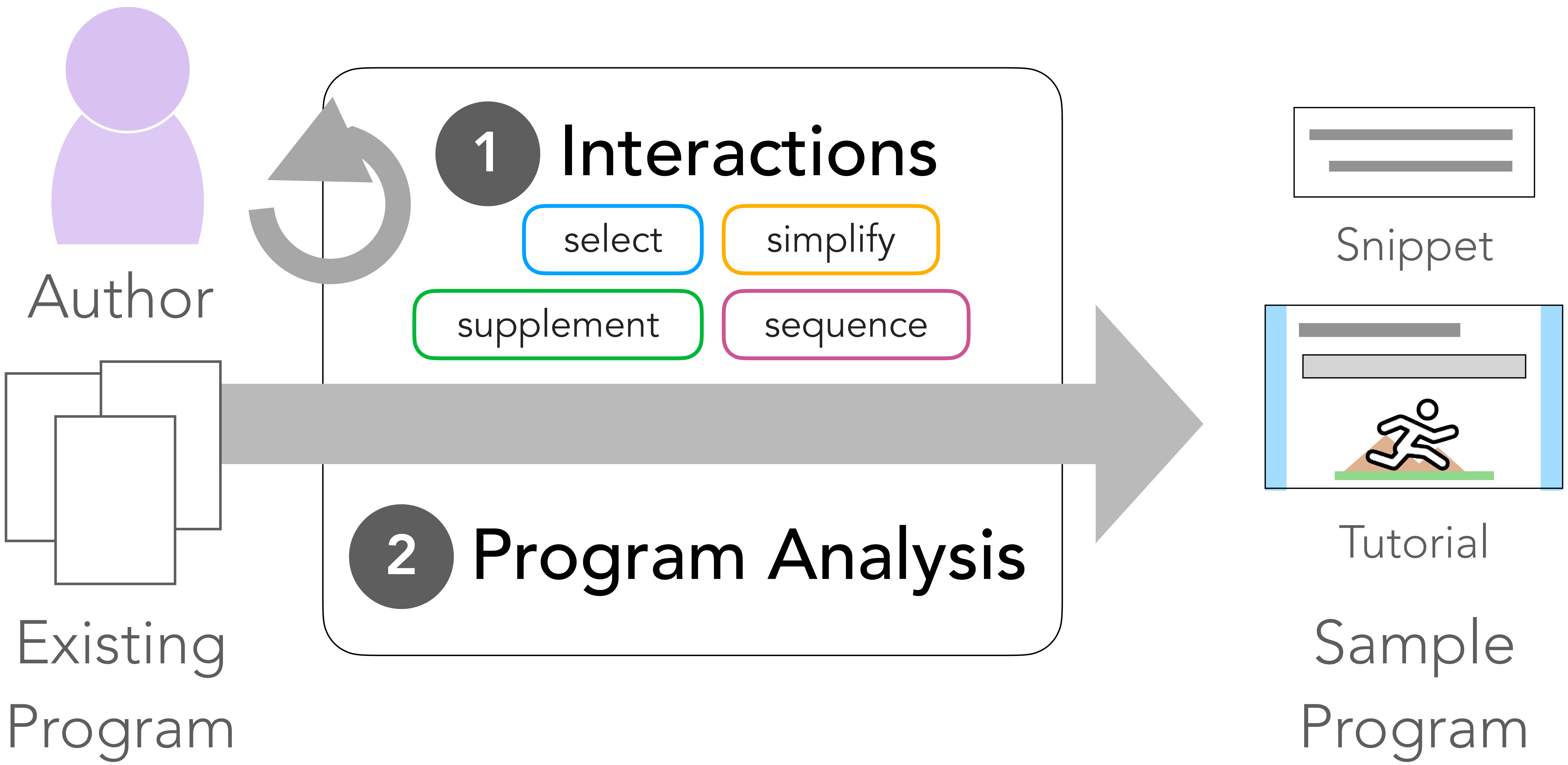
The Essence of Distillation Tools



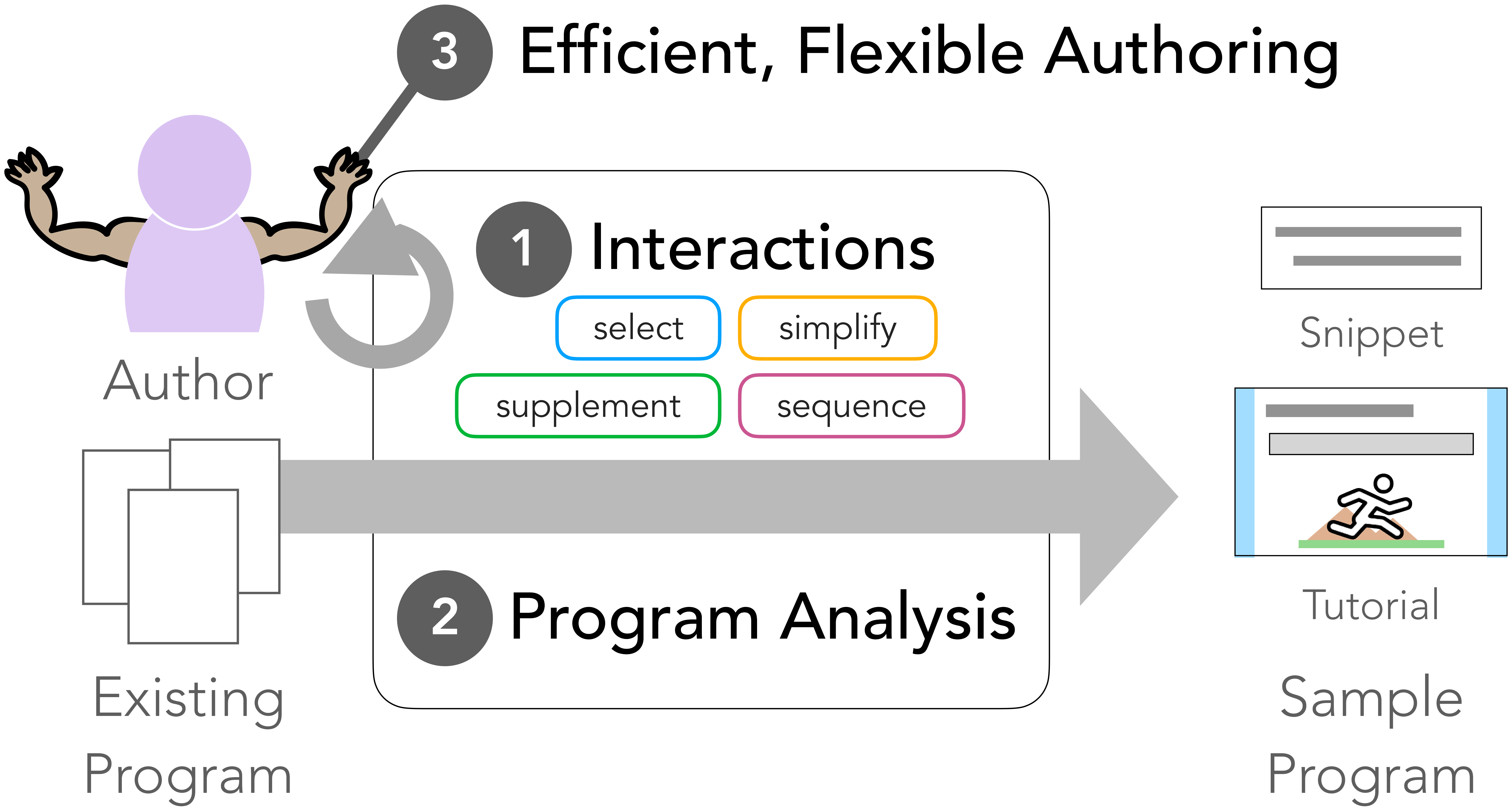
The Essence of Distillation Tools



The Essence of Distillation Tools



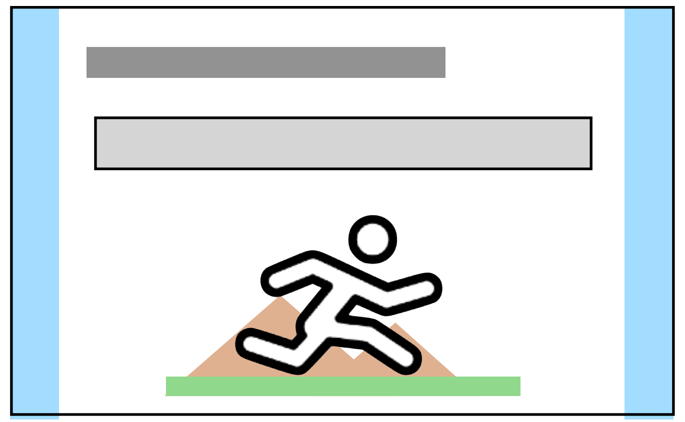
The Essence of Distillation Tools



This Talk

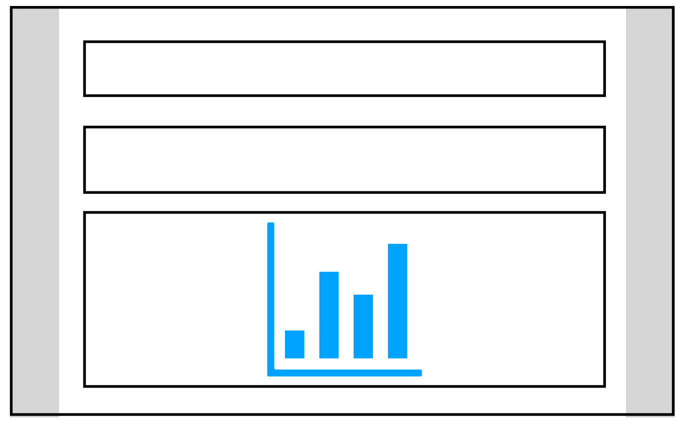
CodeScoop [CHI '18]

code editor | snippets | select | simplify



Torii [CHI '20]

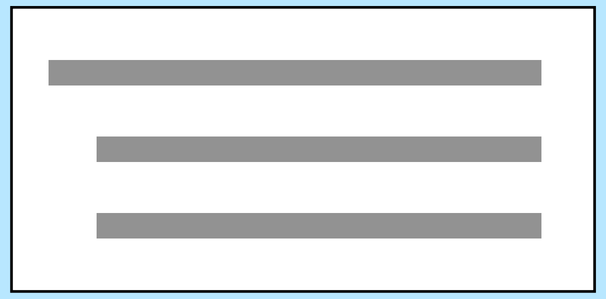
code editor | tutorials | supplement | sequence



Gathering tools [CHI '19]

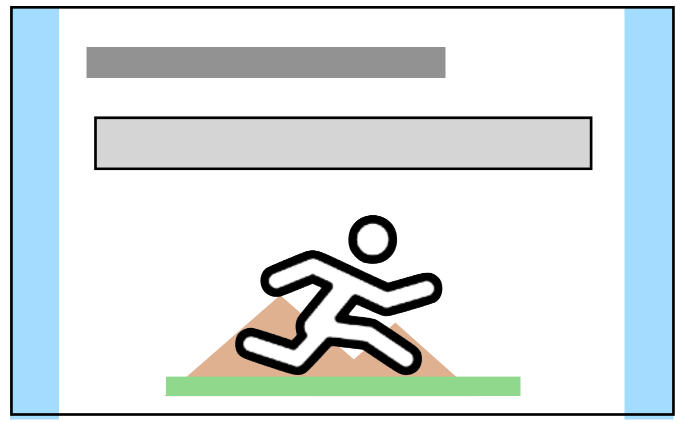
notebook | snippets | tutorials | select | supplement

This Talk



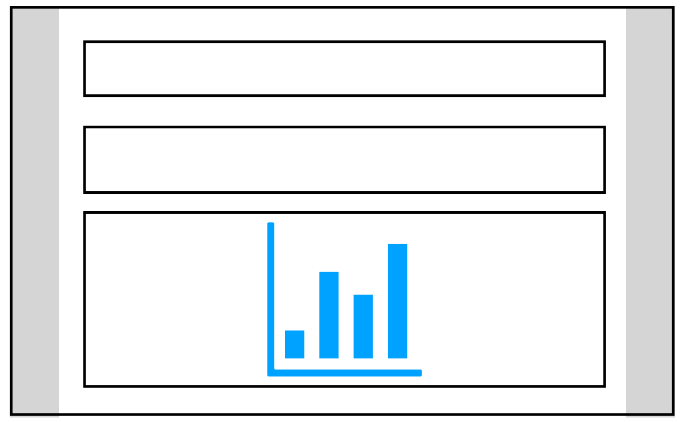
CodeScoop [CHI '18]

code editor | snippets | select | simplify



Torii [CHI '20]

code editor | tutorials | supplement | sequence



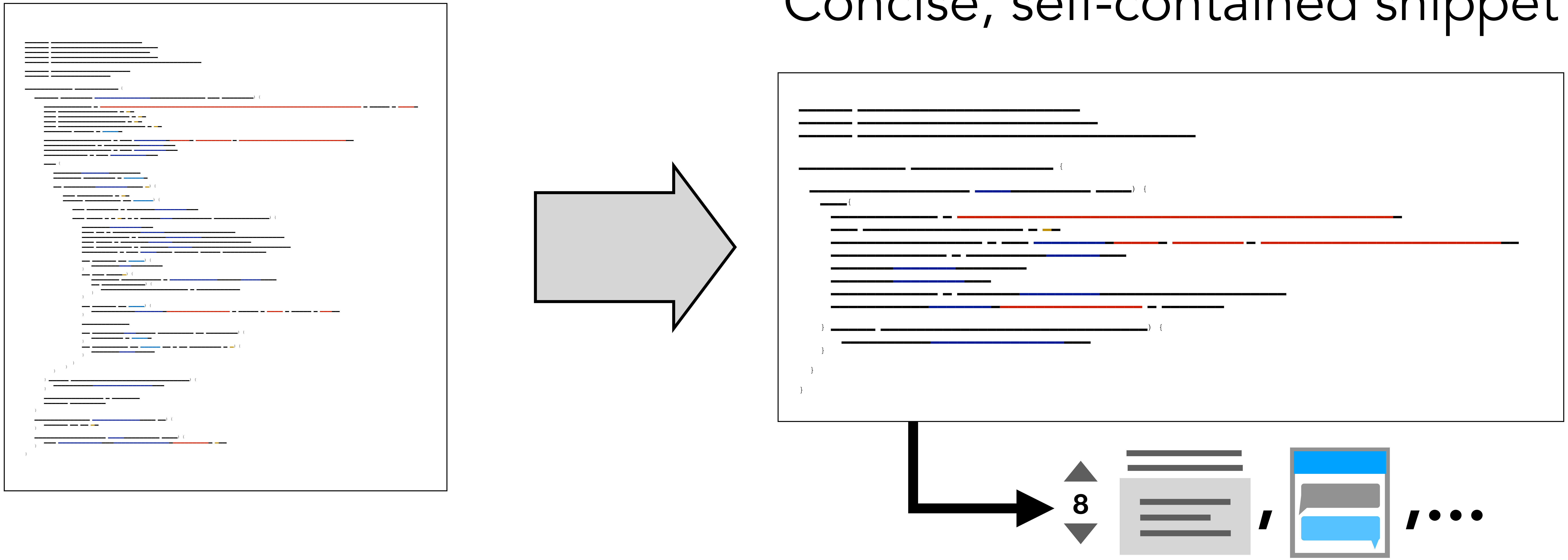
Gathering tools [CHI '19]

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.


UNREGISTERED
Untitled.py — Edited
n 3
Language
Run
Stop
Run Settings...
View
Untitled.py
rviews

Sample Editor
UNREGISTERED
Untitled.py — Edited
Language
Run
Stop
Run Settings...
View
Untitled.py


Untitled.py — Edited
n 3
Language
Run
Stop
Run Settings...
Untitled.py
rviews
Existing Program

Testing Environment


Browser
Untitled.py — Edited
Language
Run
Stop
Run Settings...
View
Untitled.py
UNREGISTERED


UNREGISTERED
Untitled.py — Edited
n 3
Language
Run
Stop
Run Settings...
View
Untitled.py
rviews


Transcription errors
UNREGISTERED
Untitled.py — Edited
Language
Run
Stop
Run Settings...
View
Untitled.py


Transcription errors
Forgotten code
Untitled.py — Edited
Language
Run
Stop
Run Settings...
Untitled.py
View
UNREGISTERED


Transcription errors
Forgotten code
...and time-consuming removal of dead code
Untitled.py — Edited
Language
Run
Stop
Run Settings...
View
Untitled.py
UNREGISTERED


UNREGISTERED
Transcription errors
Forgotten code
Edit errors
Untitled.py — Edited
Language
Run
Stop
Run Settings...
View
Untitled.py
...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

```
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) {
```

Scoop

Undo

Run

Reset


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);
etString(COLUMN_INDEX_TITLE);
t(COLUMN_INDEX_YEAR);
getInt(COLUMN_INDEX_NUM_PAGES);
, title, year, num_pages);
(1) Author **selects** pattern
Scoop
Undo
Run
Reset

int rowNumber = 0;
while (finished == false) {
int rowCount = cursor.rowCou
for (int i = 0; i < Math.min
cursor.fetchone();
int id = cursor.getInt(C
if (id != -1) {
(1) Author **selects** pattern
(2) Editor creates snippet,
public class ExtractedExample {
public static void main(String[
int id = cursor.getInt(COLUMN
}
}
Scoop
Undo
Run
Reset


int rowNumber = 0;
while (finished == false) {
int rowCount = cursor.rowCou
for (int i = 0; i < Math.min
cursor.fetchone();
int id = cursor.getInt(C
if (id != -1) {
(1) Author **selects** pattern
(2) Editor creates snippet,
(3) Flags errors,
public class ExtractedExample {
public static void main(String[
int id = cursor.getInt(COLUMN
}
}
Scoop
Undo
Run
Reset


Database database = new Datab
Cursor cursor = database.curs
Booklist booklist = new Bookl
List titles = new ArrayList()
try {
cursor.execute(QUERY);
boolean finished = false;
for (int i = 0; i
(1) Author **selects** pattern
(2) Editor creates snippet,
(3) Flags errors,
(4) Suggests code fixes,
public class ExtractedExample {
public static void main(String[
int id = cursor.getInt(COLUMN
}
}
Add code
Stub out
Line 23
Scoop
Undo
Run
Reset


(1) Author **selects** pattern
(2) Editor creates snippet,
(3) Flags errors,
(4) Suggests code fixes,
(5) Suggests simplifications,
Define
Scoop
Undo
Run
Reset


cursor.fet
int id = c
String tit
int year =
int num_pa
Book book
if (title
titles
if (DEBUG
dExample {
main(String[] args) {
etInt(0);
ook(id, title, year, 330);
Scoop
Add code
Set value
250
200
330
Reset
(1) Author selects pattern
(2) Editor creates snippet,
(3) Flags errors,
(4) Suggests code fixes,
(5) Suggests simplifications,

(1) Author **selects** pattern
(2) Editor creates snippet,
(3) Flags errors,
(4) Suggests code fixes,
(5) Suggests **simplifications**,
(6) And makes automatic fixes.
import org.acme.database.Databas
import org.acme.database.Cursor;
import org.acme.database.Book;
public class ExtractedExample {
public static void main(String
Database database = new Data
Cursor cursor = database.cur
cursor.execute("SELECT id, t
cursor.fetchone();
int id = cursor.getInt(0);
Book book = new Book(id, "Da
}
}
int rowCount
for (int i =
cursor.f
int id =
String t
int year
int num_
Scoop
Undo
Run
Reset

```
for (____ _ = _; _ < ____.___(_______, _______); ++

    cursor.fetchone();

    int id = cursor.getInt(_______________);

    ______ _____ = ______.________(__________________

    ___ ____ = ______.______(________________);

    ___ ________ = ______.______(___________________)

    Book book = new Book(id, _____, ____, ________);
```

A row of data is fetched from the database.

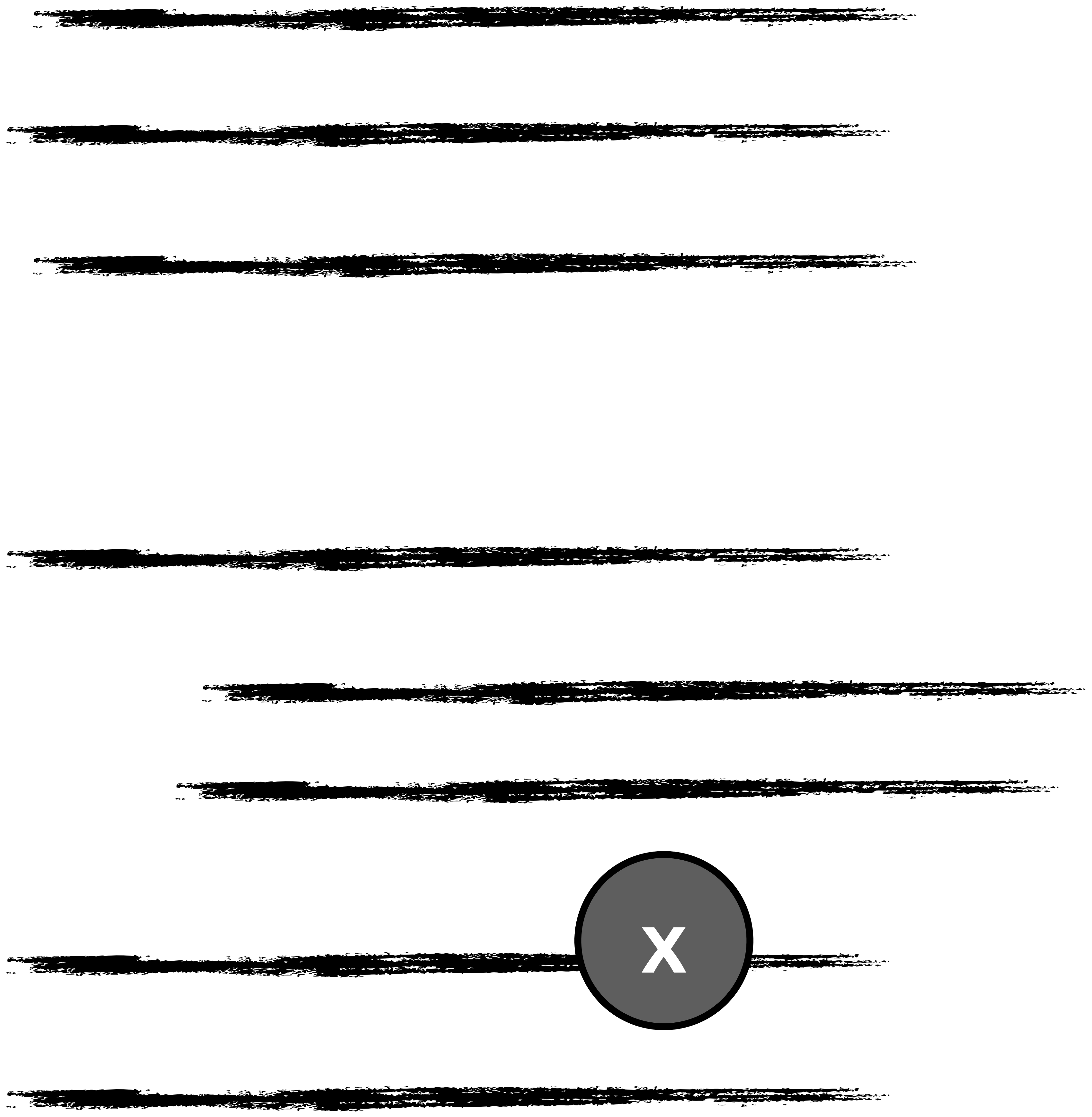
```
for (____ _ = _; _ < ____.___(_______, _______); ++

    cursor.fetchone();

    int id = cursor.getInt(_______________);

    ______ _____ = ______.________(____________________

    ___ ____ = ______.______(_______________);

    ___ ________ = ______.______(_______________)

    Book book = new Book(id, _____, ____, ________);
```

A row of data is fetched from the database.

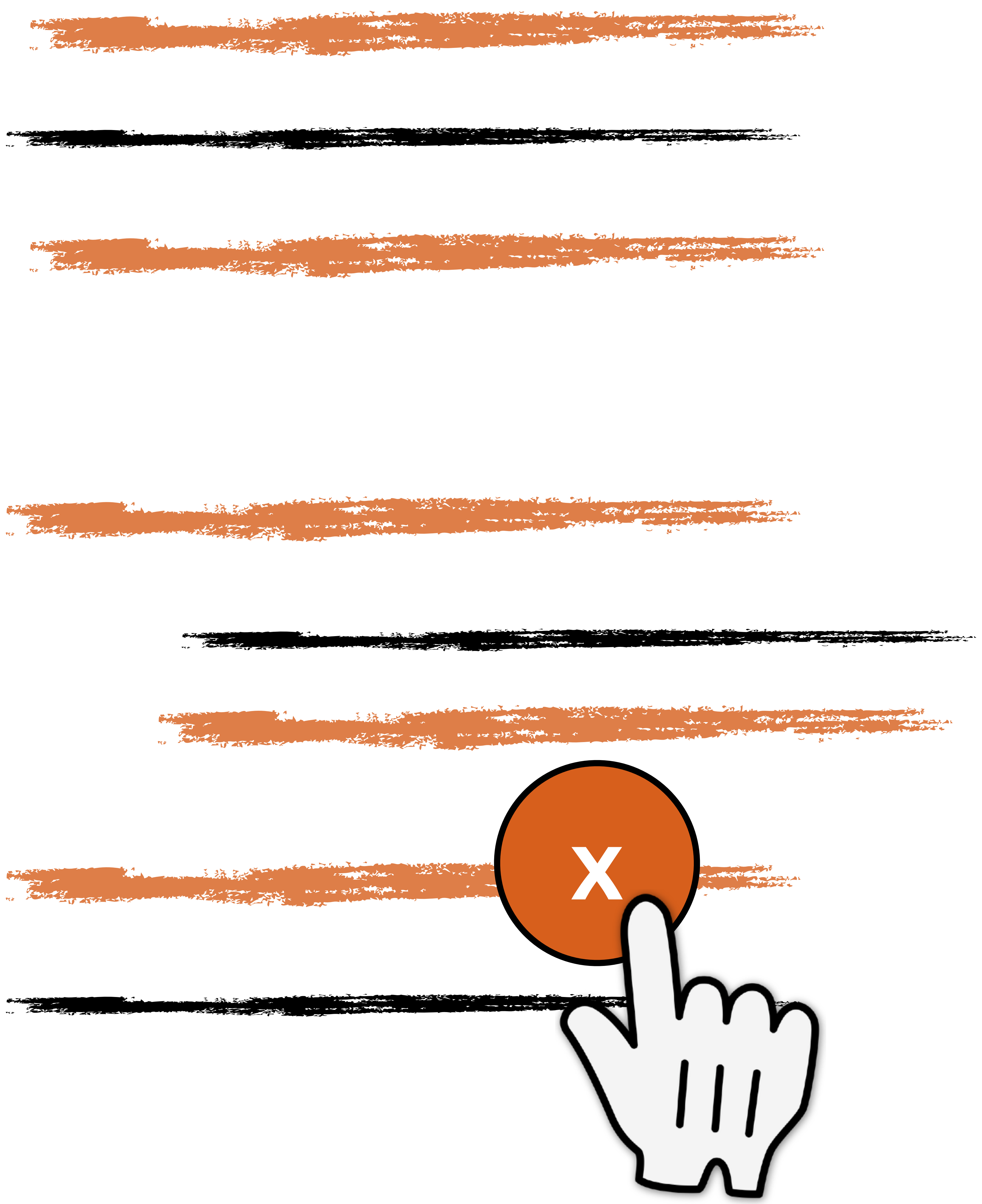
This is data for a book.

```
for (int i = 0; i < Math.min(rowCount, maxBooks); ++
  cursor.fetchone();
  int id = cursor.getInt(COLUMN_INDEX_ID);
  String title = cursor.getString(COLUMN_INDEX_T
  int year = cursor.getInt(COLUMN_INDEX_YEAR);
  int num_pages = cursor.getInt(COLUMN_INDEX
  Book book = new Book(id, title, year, num_pag
```

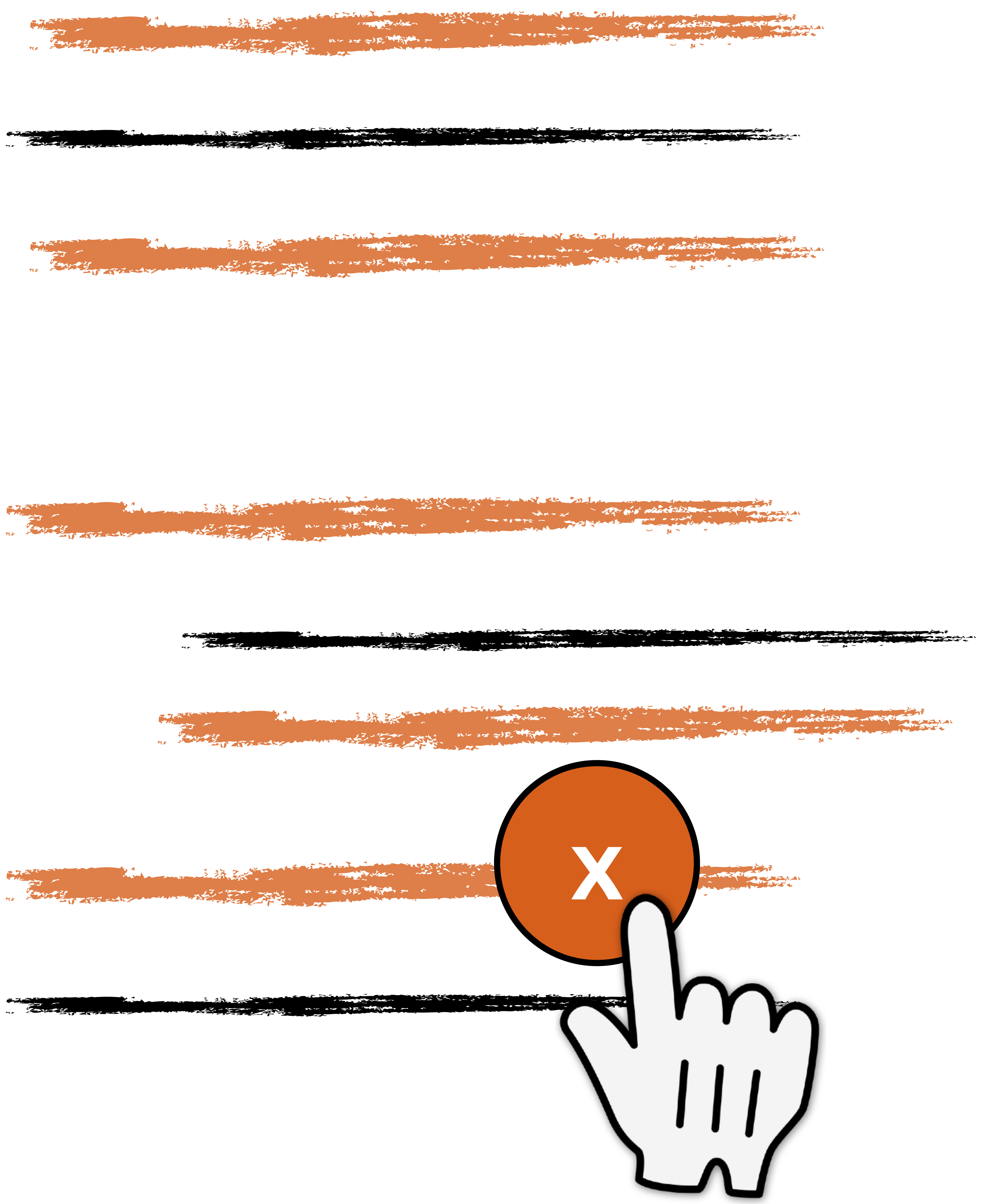
Program **Slicing**



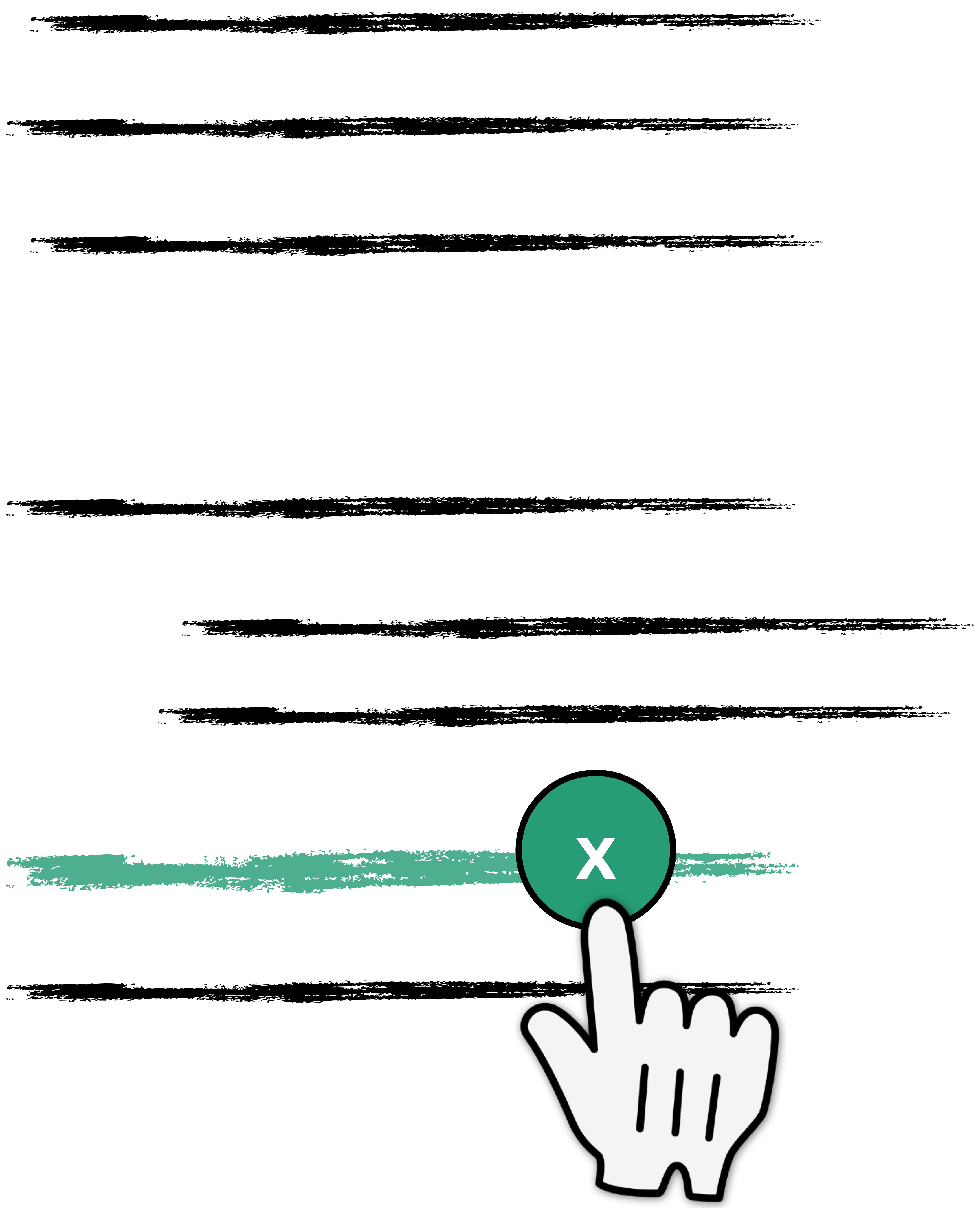
Program **Slicing**



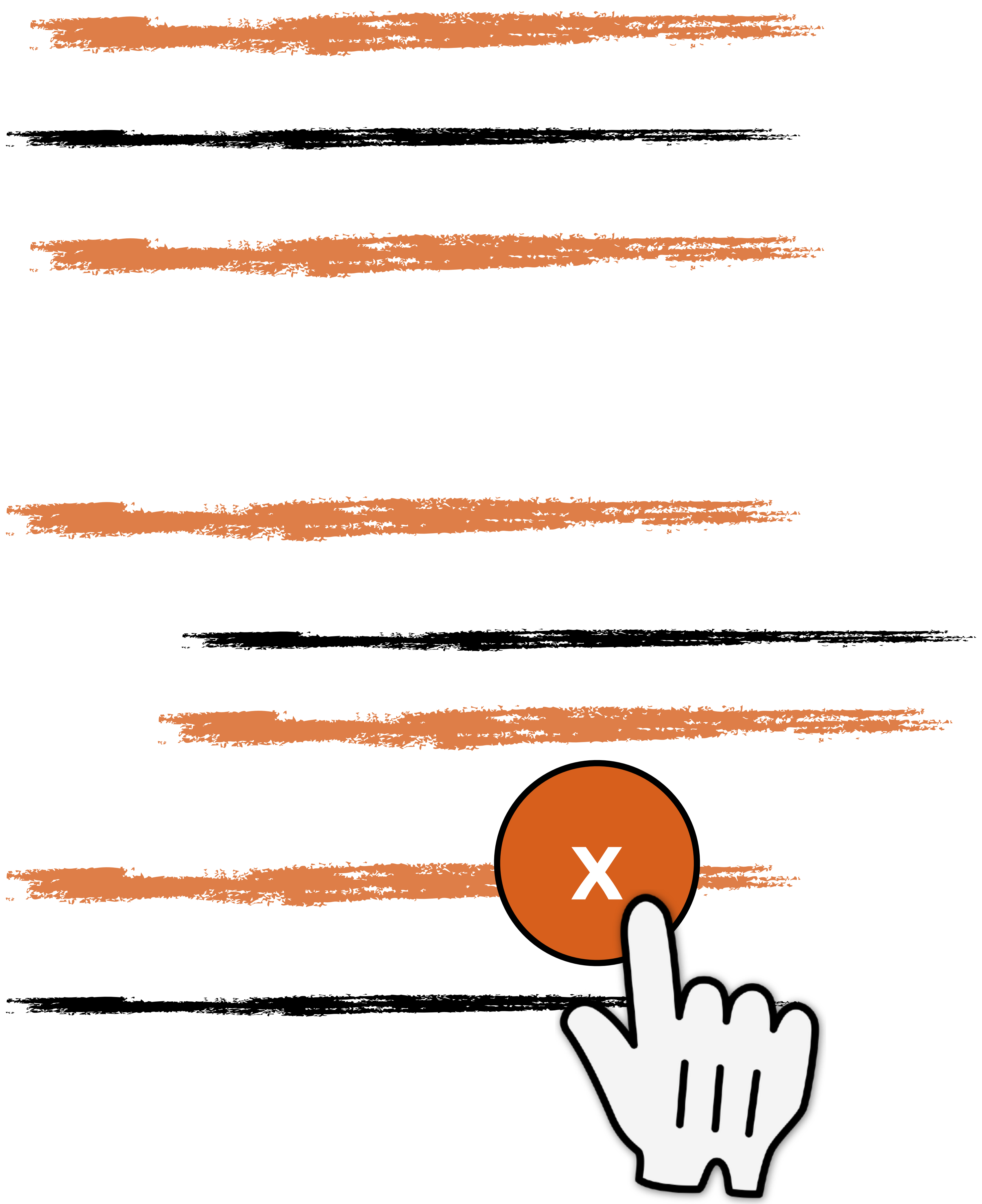
Program **Slicing**



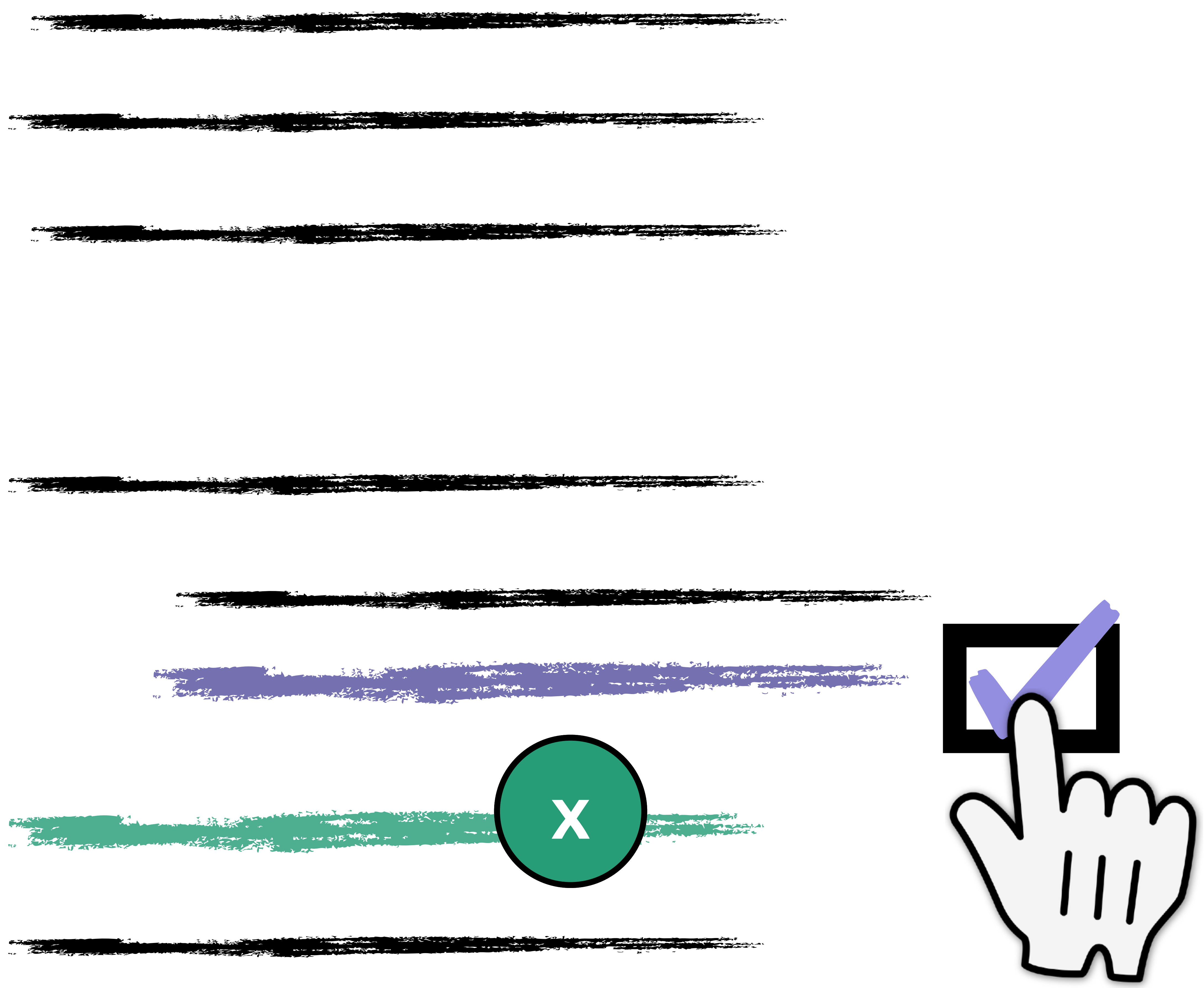
Program **Scooping**



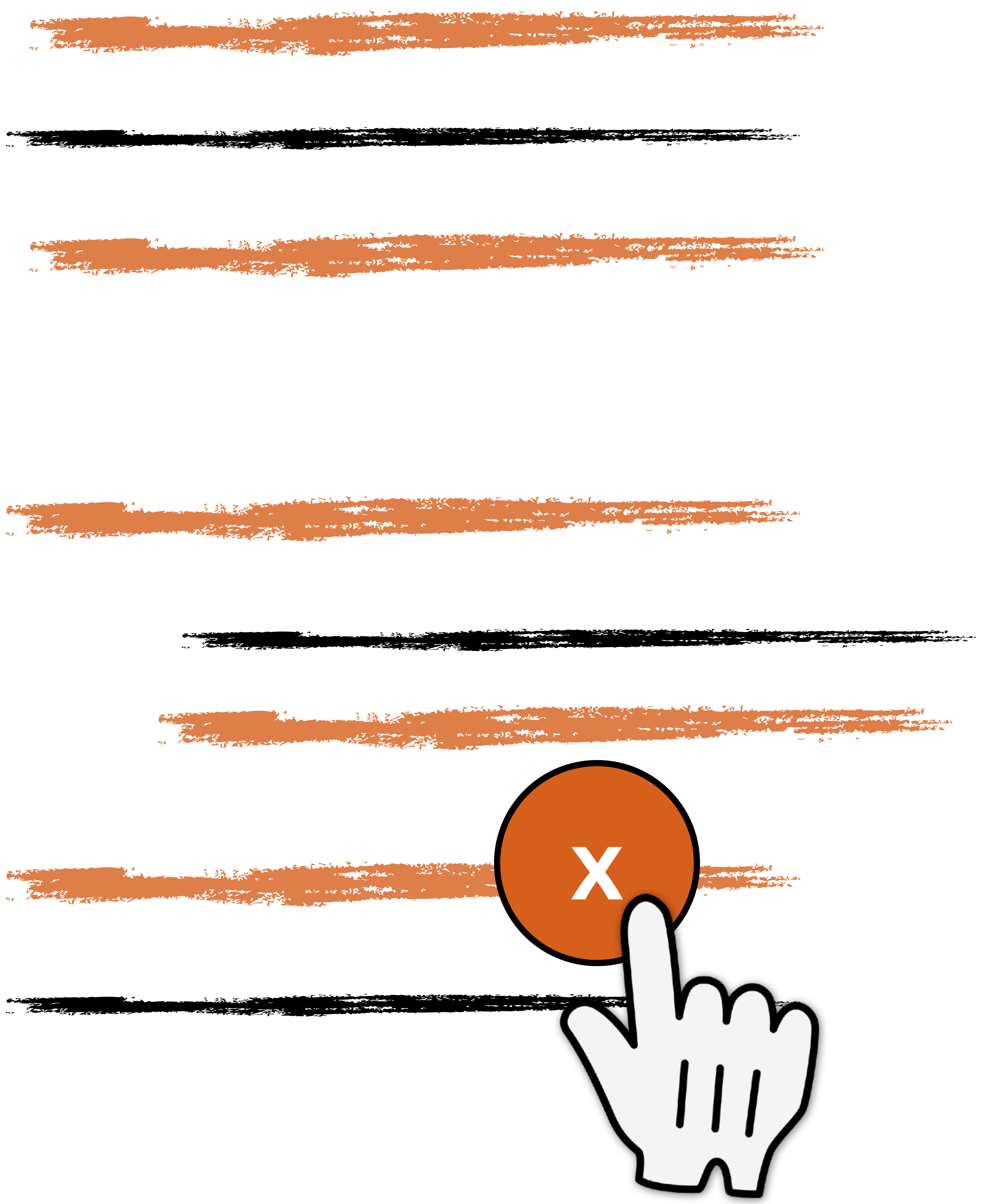
Program **Slicing**



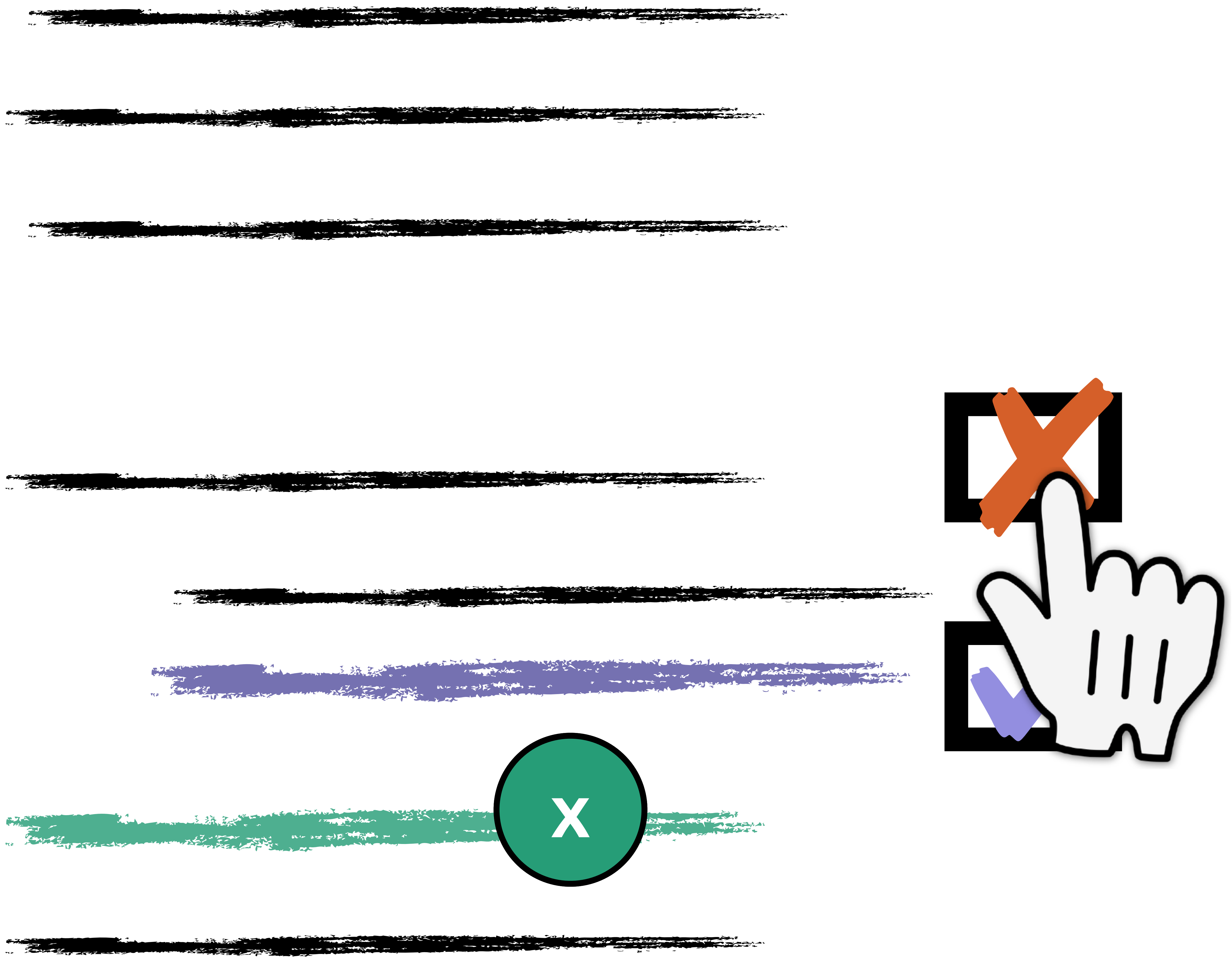
Program **Scooping**



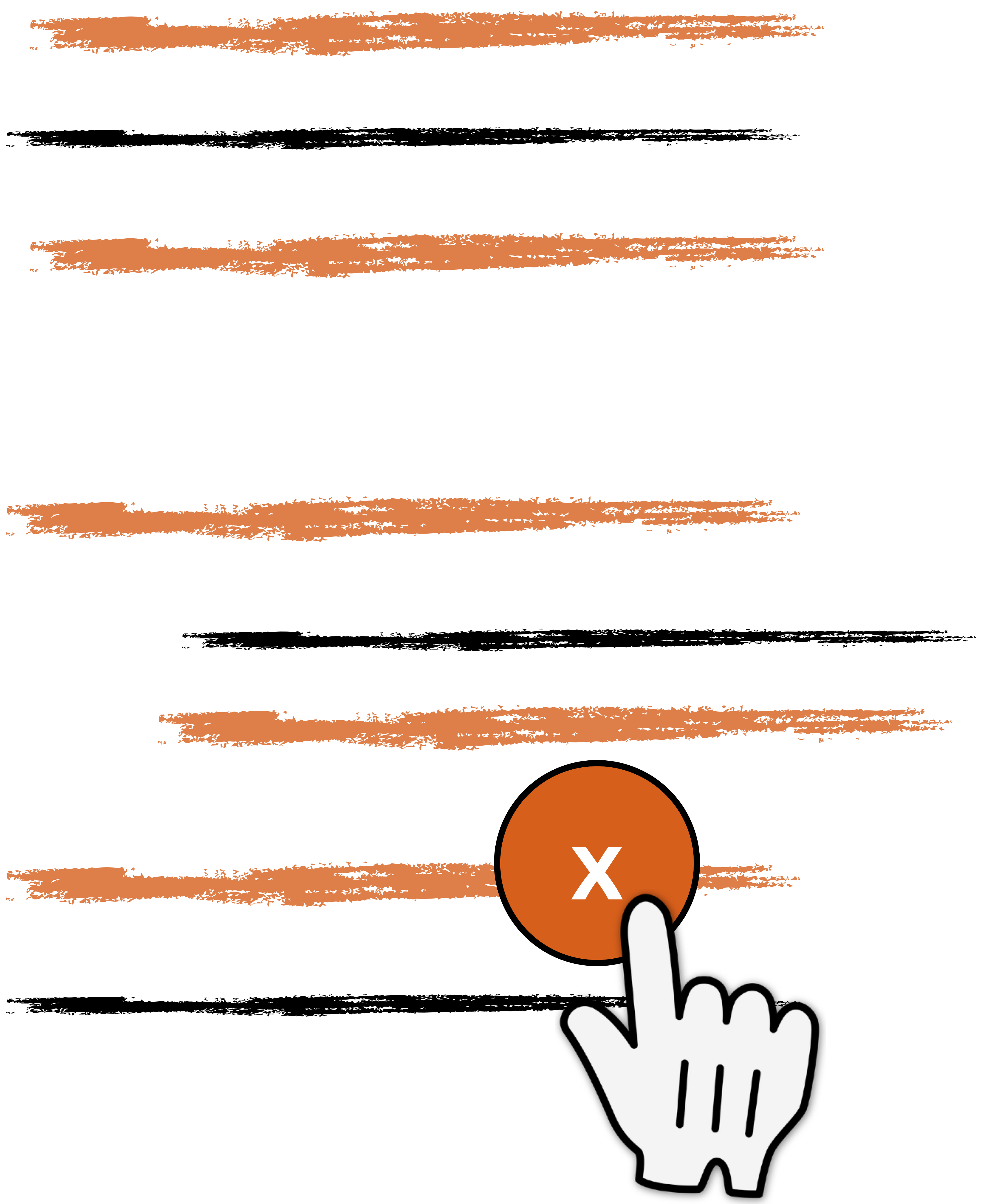
Program **Slicing**



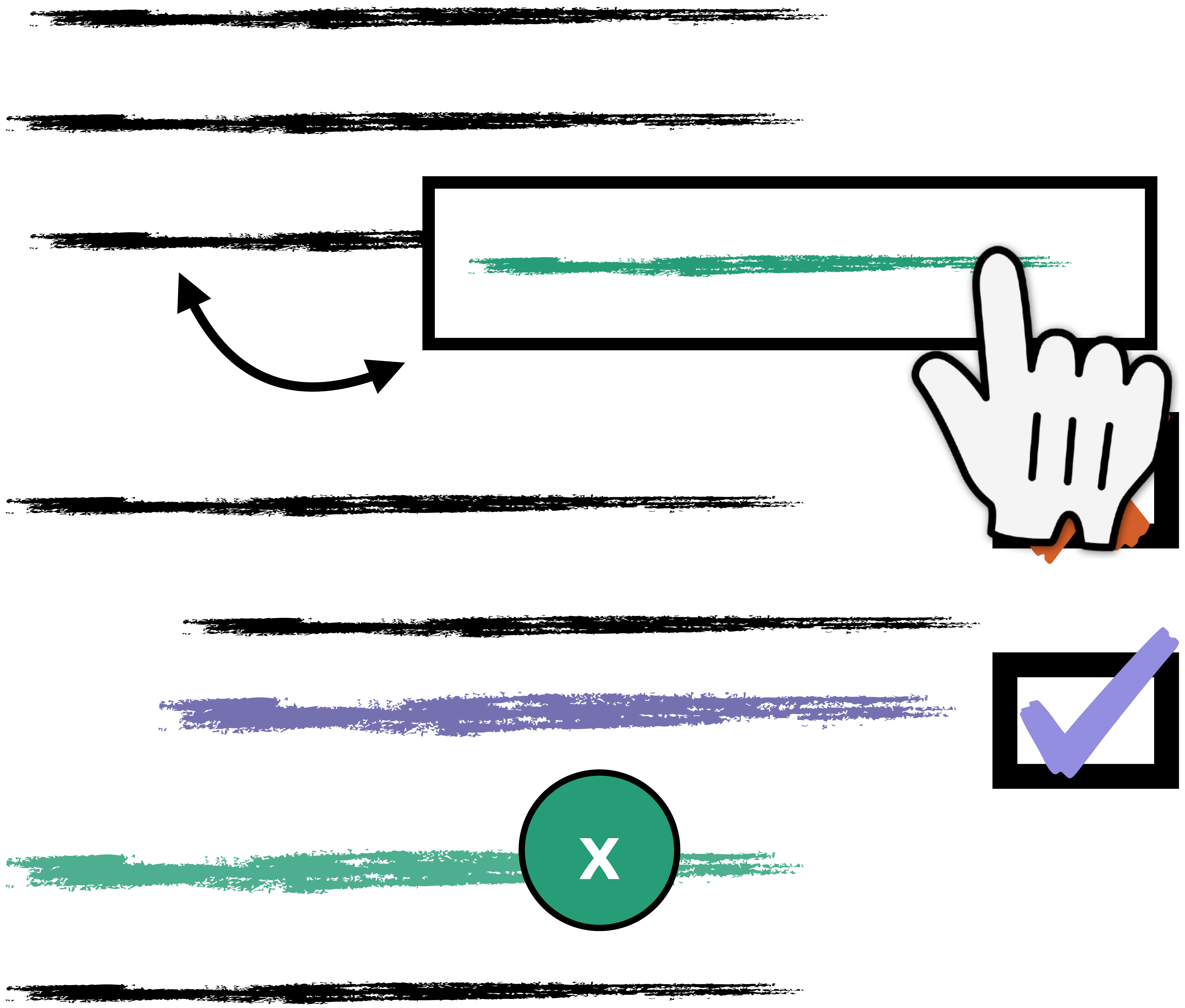
Program **Scooping**



Program **Slicing**



Program **Scooping**



```
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) {
            boolean bestseller = isBestseller(book.getId());
            if (bestseller) {
                booklist.hasBestseller = bestseller;
            }
        }

        if (DEBUG == true) {
            System.out.println("Fetched book: " + title + " (" + genre + ")");
```

Scoop

Undo

Run

Reset

Show Help

Static Dataflow Analysis
int rowNumber = 0;
String title = cursor.
if (title != null) {
if (DEBUG == true) {
System.out.printl
public class ExtractedExample {
public static void main(String[] args) {
String title = cursor.getString(COLUMN_INDE
}
}
Undo
Run
Reset
Show Help


Database database = new Database("lou"
Cursor cursor = database.cursor();
Booklist booklist = new Booklist();
List titles = new ArrayList();
try {
cursor.execute(QUERY);
boolean finished = false;
if (cursor.rowCount() > 0) {
int rowNumber = 0;
while (finished == false) {
int rowCount = cursor.rowC
for (int i = 0; i < Math.m
cursor.fetchone();
int id = cursor.getInt
String title = cursor.
int year = cursor.getI
int num_pages = cursor
Book book = new Book(i
public class ExtractedExample {
public static void main(String[] args) {
String title = cursor.getString(COLUMN_INDE
}
}
Add code
Stub out
Line
Scoop
Undo
Run
Reset
Show Help

Database database = new Database("lou"
Cursor cursor = database.cursor();
Booklist booklist = new Booklist();
List titles = new ArrayList();
try {
cursor.execute(QUERY);
boolean finished = false;
if (cursor.rowCount() > 0) {
int rowNumber = 0;
while (finished == false) {
int rowCount = cursor.rowC
for (int i = 0; i < Math.m
cursor.fetchone();
int id = cursor.getInt
String title = cursor.
int year = cursor.getI
int num_pages = cursor
Book book = new Book(i
public class ExtractedExample {
public static void main(String[] args) {
Cursor cursor = database.cursor();
String title = cursor.getString(COLUMN_IND
}
}
Include 'try' structure?
Accept
Reject
Scoop
Undo
Run
Reset
Show Help

21
22 Database database = new Database("lou"
23 Cursor cursor = database.cursor();
24 Booklist booklist = new Booklist();
25 List titles = new ArrayList();
26
27 try {
28
29 cursor.execute(QUERY);
30 boolean finished = false;
31
32 if (cursor.rowCount() > 0) {
33
34 int rowNumber = 0;
35 while (finished == false) {
36
37 int rowCount = cursor.rowC
38
39 for (int i = 0; i < Math.m
40
41 cursor.fetchone();
42 int id = cursor.getInt
43 String title = cursor.
44 int year = cursor.getI
45 int num_pages = cursor
46 Book book = new Book(i
1 public class ExtractedExample {
2
3 public static void main(String[] args) {
4
5 Cursor cursor = database.cursor();
6 try {
7 cursor.execute(QUERY);
8 cursor.fetchone();
9 String title = cursor.getString(COLUMN_I
10 } catch (Connecti
tion) {
11 }
12
13 }
14
15 }
Do you want any of these uses of cursor?
Line 32
Line 37
Line 42
Scoop
Undo
Run
Reset
Show Help

```
        String QUERY = "SELECT id, title, year
        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("lou"
        Cursor cursor = database.cursor();
        Booklist booklist = new Booklist();
        List titles = new ArrayList();

        try {

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

            if (cursor.rowCount() > 0) {

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

                    int rowCount = cursor.rowC

                    for (int i = 0; i < Math.m
```

```
 ExtractedExample {

tic void main(String[] args) {

ursor = database.cursor();

.execute(QUERY);
.fetchone();
 title = cursor.getString(1);
(ConnectionException excep
```

Add code

Set value

Scoop

Undo

Run

Reset

Show Help

String QUERY = "SELECT id, title, year
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("lou"
Cursor cursor = database.cursor();
Booklist booklist = new Booklist();
List titles = new ArrayList();
try {
cursor.execute(QUERY);
boolean finished = false;
if (cursor.rowCount() > 0) {
int rowNumber = 0;
while (finished == false) {
int rowCount = cursor.rowC
for (int i = 0; i < Math.m
public class ExtractedExample {
public static void main(String[] args) {
Cursor cursor = database.cursor();
try {
cursor.execute
year,
cursor.fetchon
String title =
tString(1);
} catch (Connectio
ception exception) {
}
}
}
Add code
Stub out
ine 22
Scoop
Undo
Run
Reset
Show Help

Scooping Summary

First selections

Scooping Summary

Scooping Summary

Optional control

Code fixups

First selections

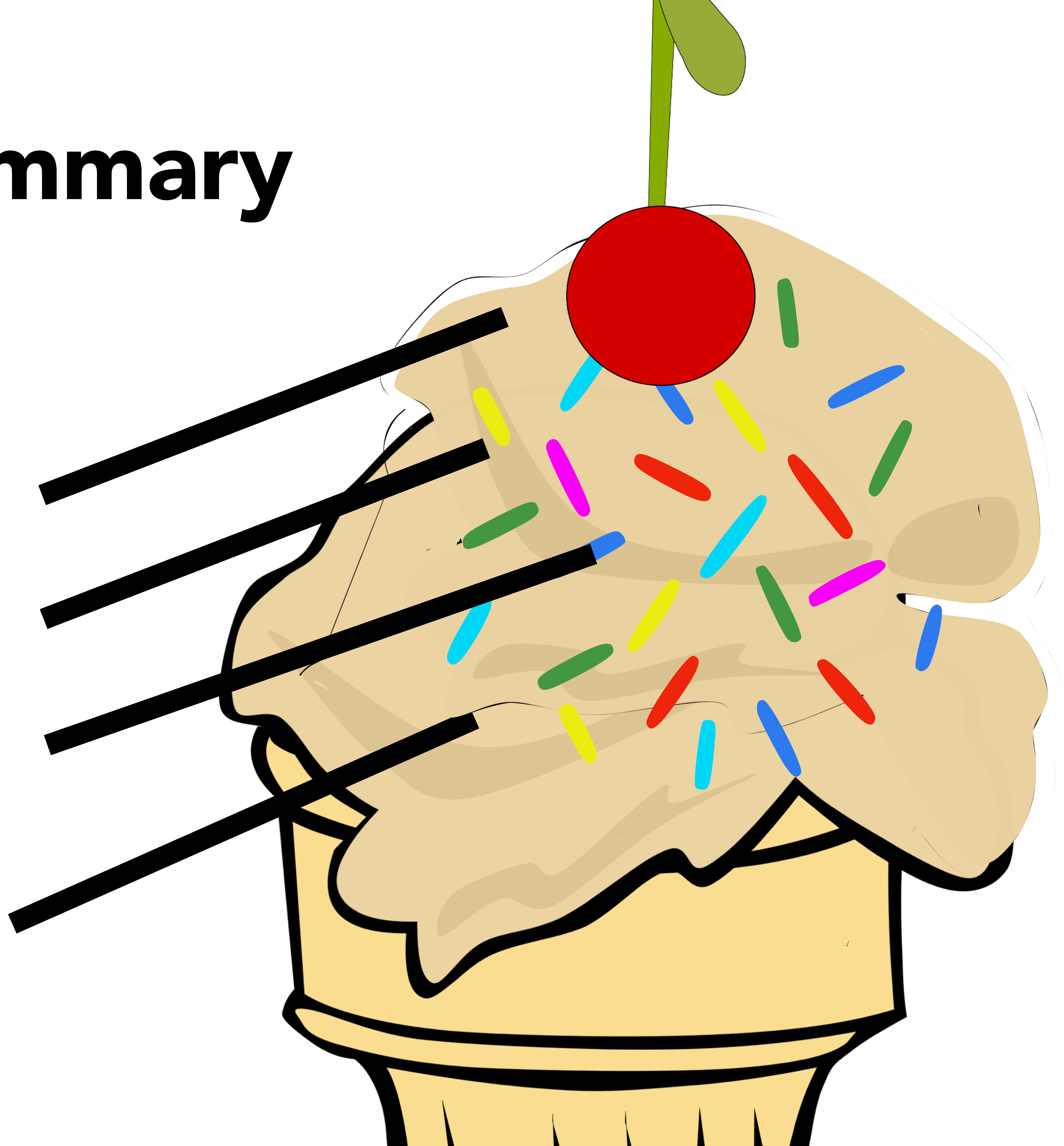
Scooping Summary

Variable substitutions

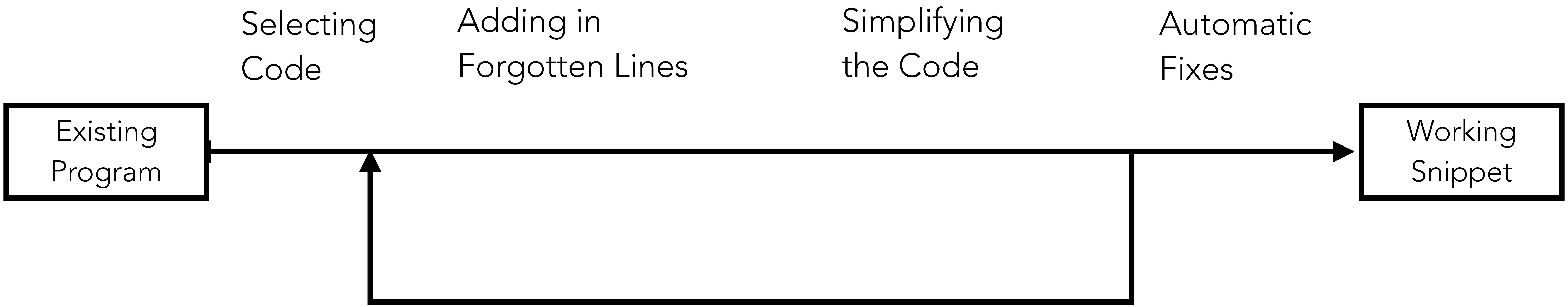
Optional control

Code fixups

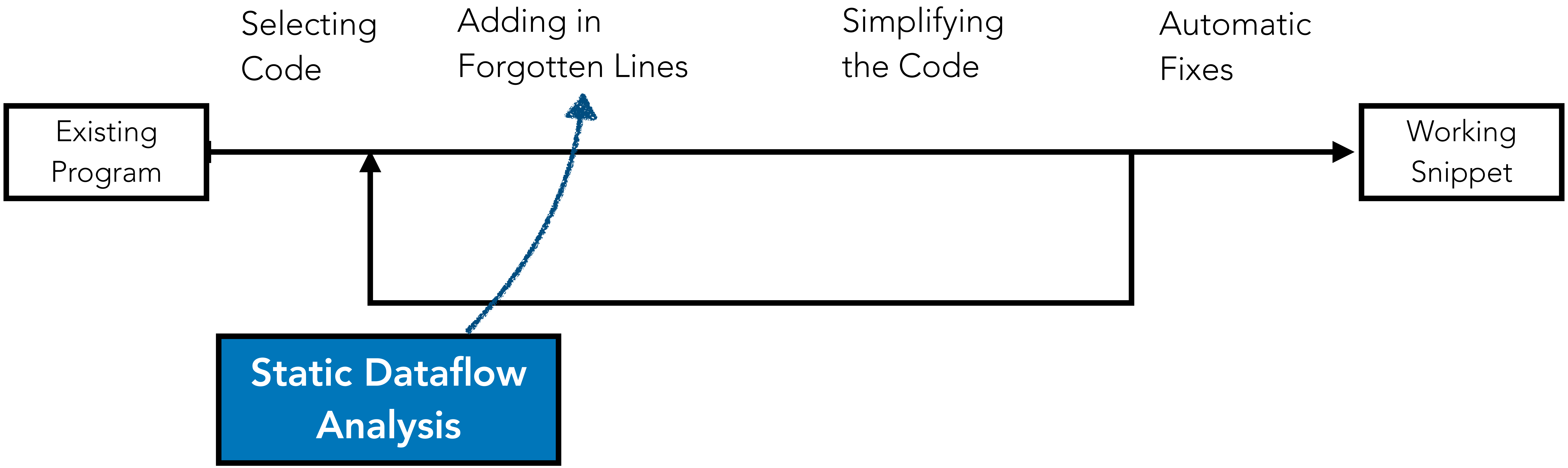
First selections



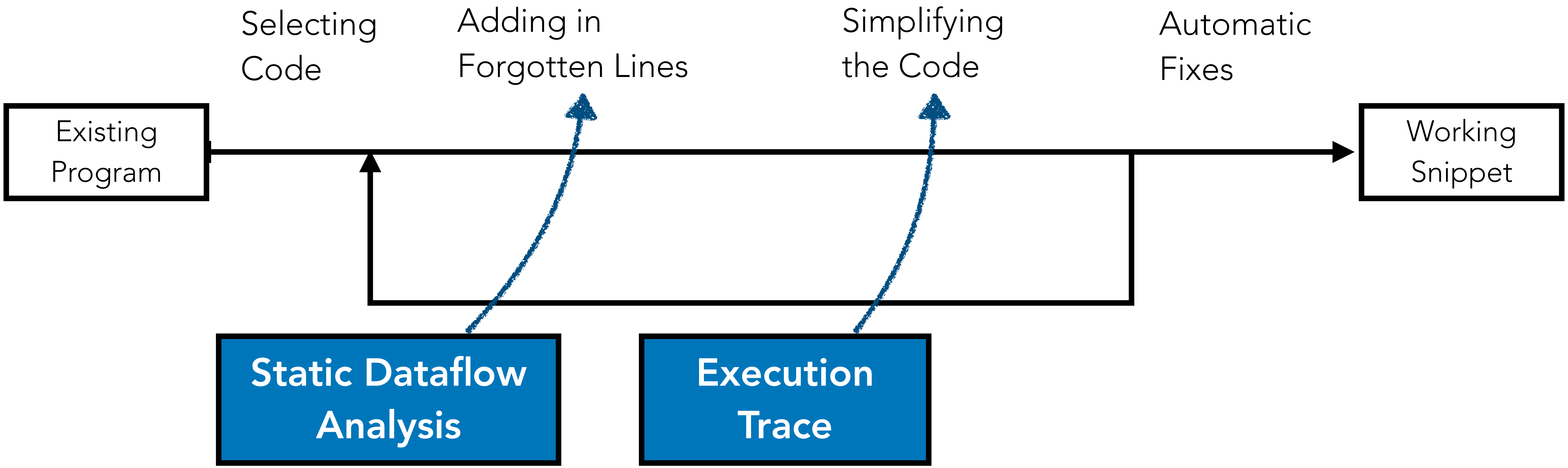
Program Analysis



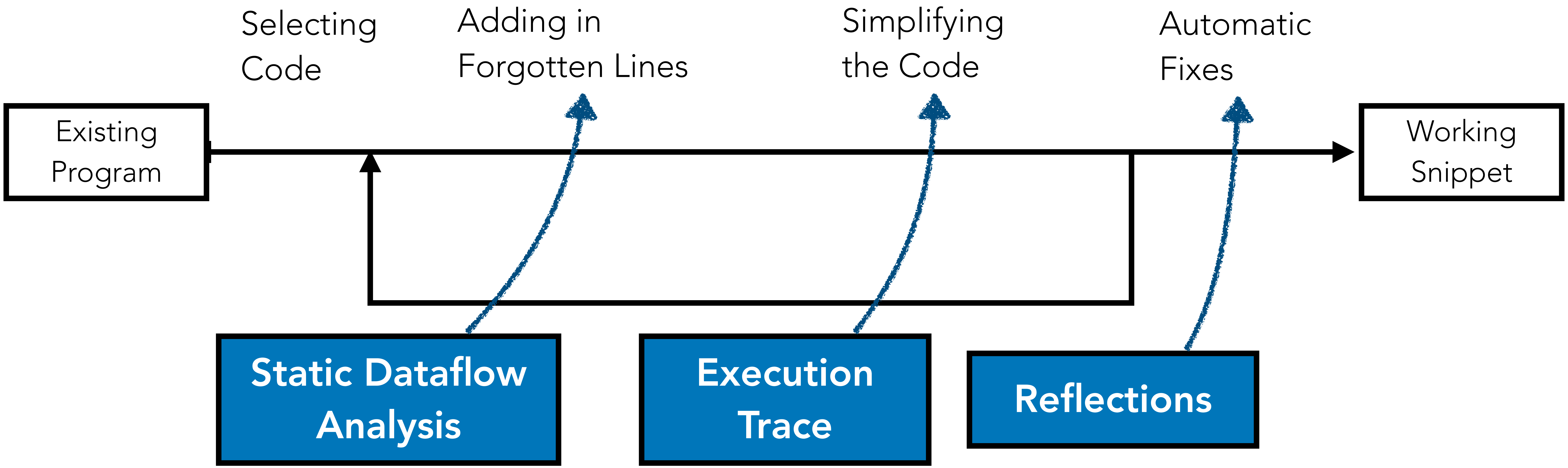
Program Analysis



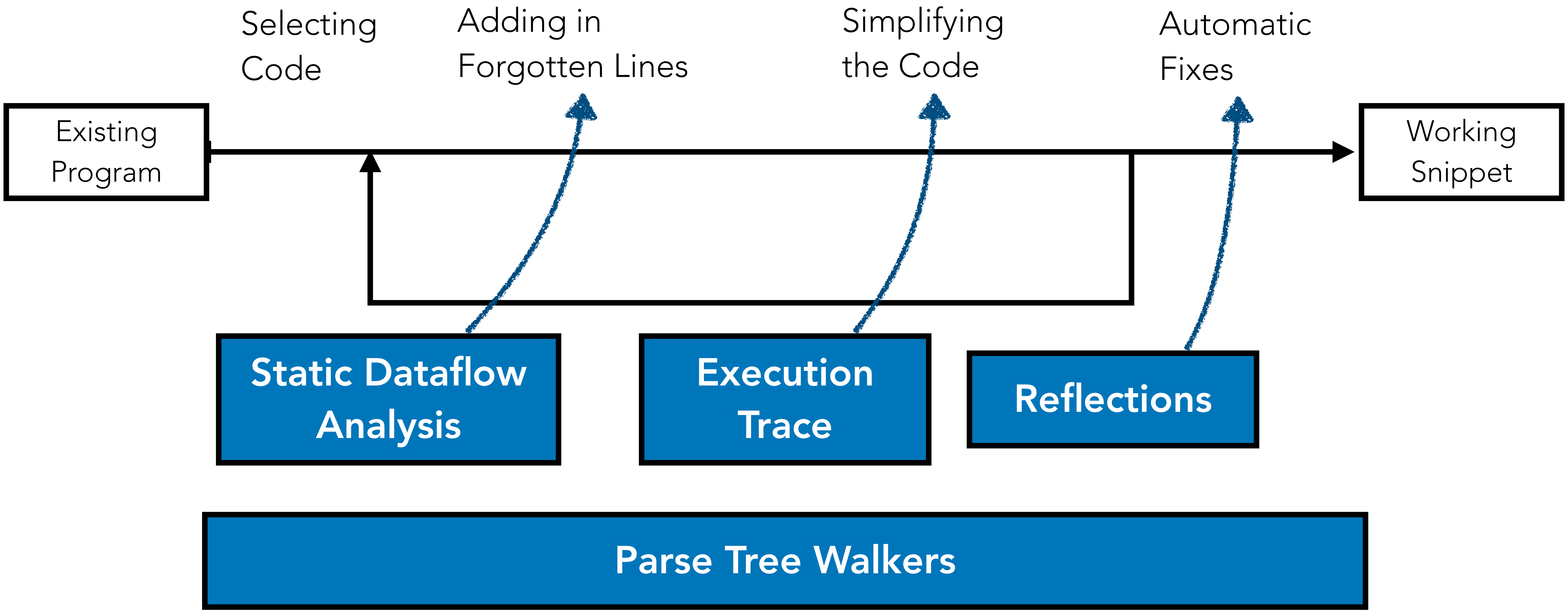
Program Analysis



Program Analysis



Program Analysis



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

Was CodeScoop, vs. the baseline...?

Faster to use?	**Yes**.	5.8 min vs 9.6 min	$p < .001$
Easier to use?	**Yes**.	$\Delta = 3$ (7-pt scale)	$< .01$
More **enjoyable**?	**Yes**.	$\Delta = 3$ "	$< .01$
Producing more **satisfying** samples?	**Yes**.	$\Delta = 2$ "	$< .01$

"[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

Variable	Add Code	Insert Literal
`COLUMN_INDEX_ID`	18	5 6 15
`COLUMN_INDEX_NUM_PAGES`	18	5 12 15
`COLUMN_INDEX_TITLE`	18	5 6 12 15
`COLUMN_INDEX_YEAR`	18	5 12 15
`num_pages`	5 18	
`QUERY`	5 17	6 18

Task 1

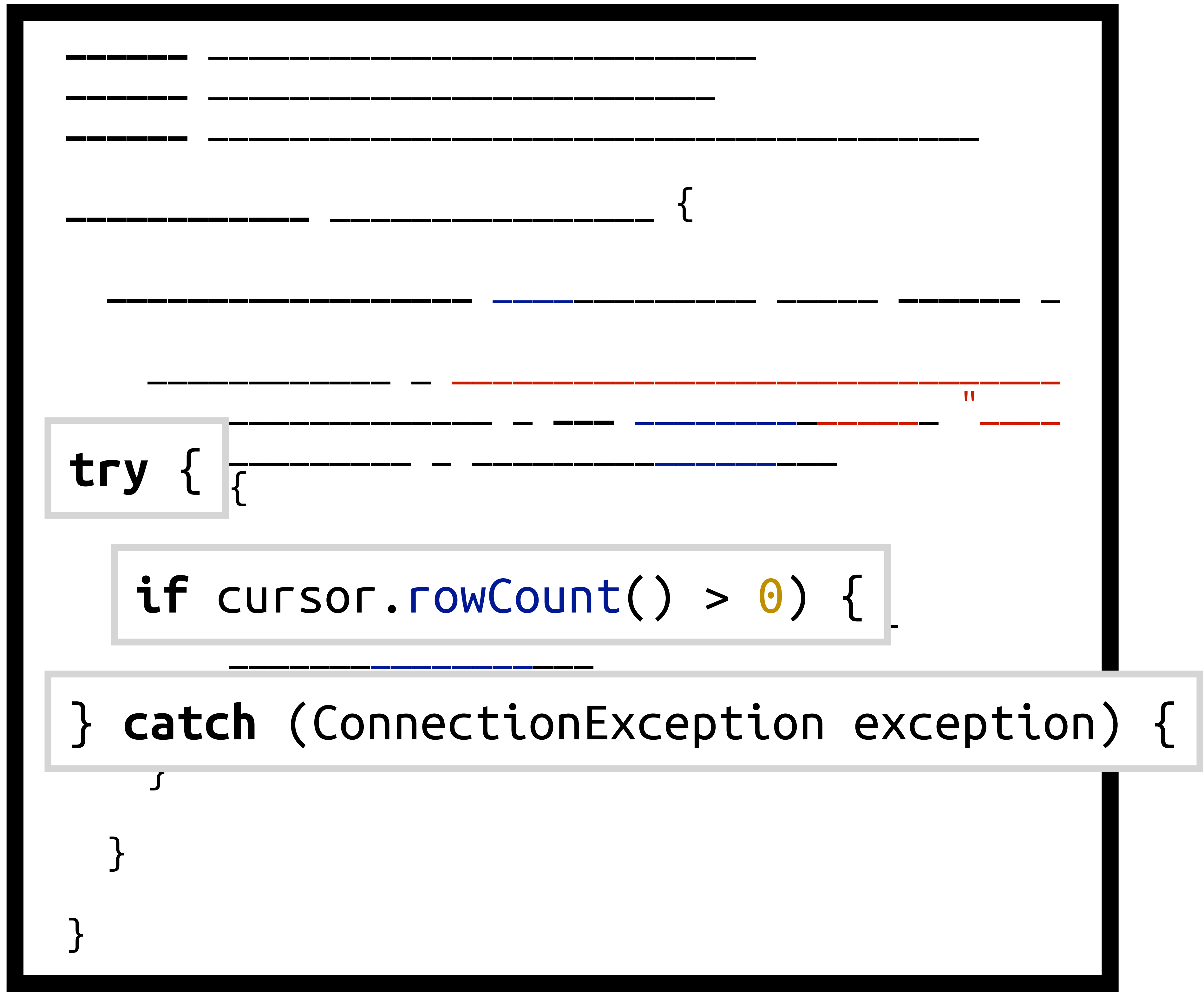
Variable	Add Code	Insert Literal
`arg0`	3 4 8 19	16
`priceInt`	3 4	
`query`	3 4 8 16 19	7 9

Task 2

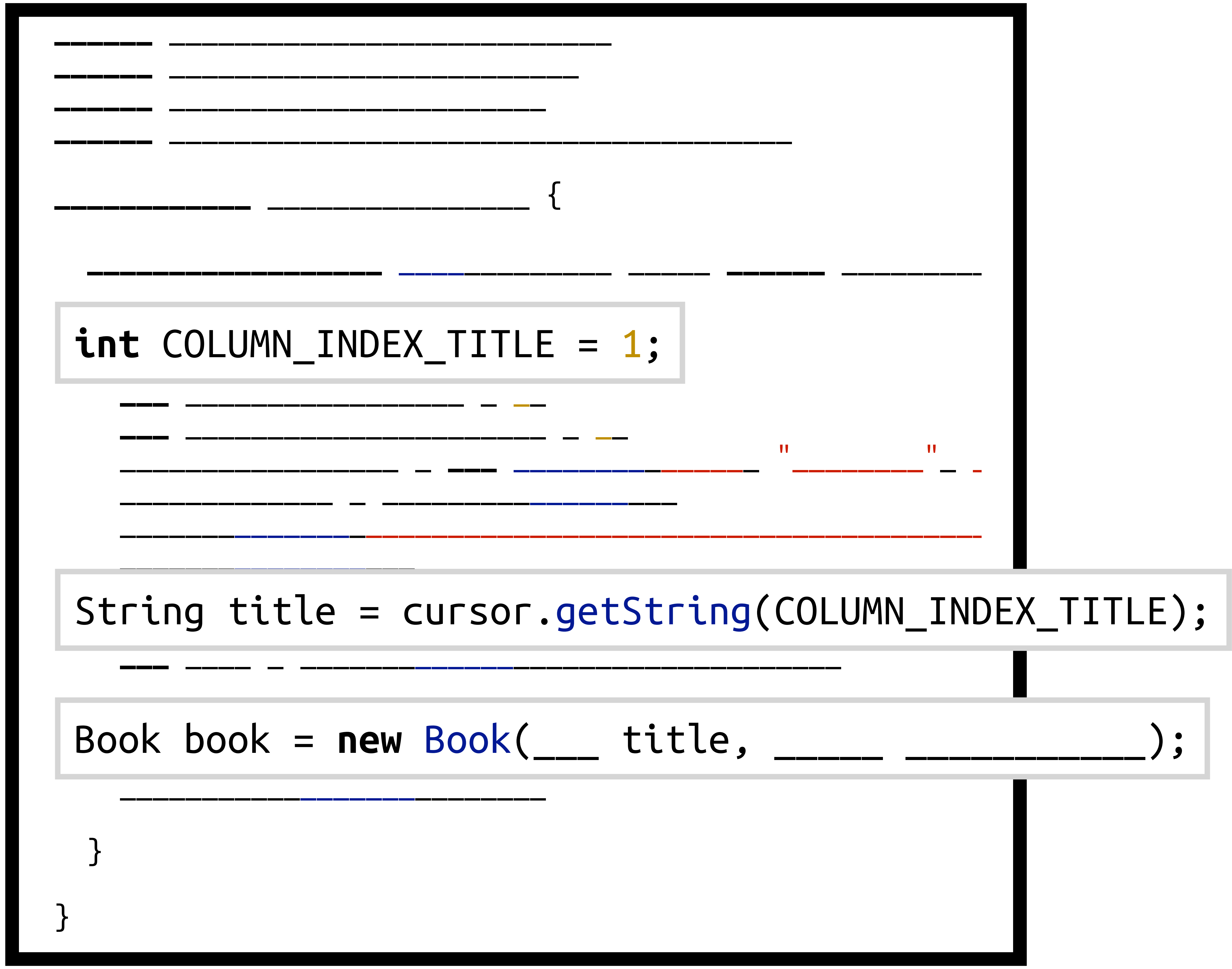
Variable	Add Code	Insert Literal
`arg1`	2 10	14
`destination`	1 2 10 13 14	11
`messageHtml`	1 2 10 14	11 13
`password`	1 2 13	11 14
`sslFactoryClass`	1 2 10 11 13	14
`username`	1 2 13	11 14

Task 3

Flexibility: Extracting snippets in different ways



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



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

Takeaways from Study

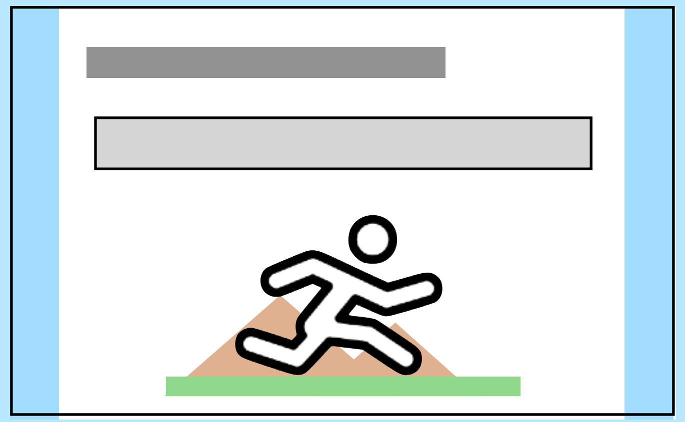
Q1. Scooping was *efficient* compared to using the baseline text editor.

Q2. Scooping provided *flexibility* in influencing the appearance of snippets.

This Talk

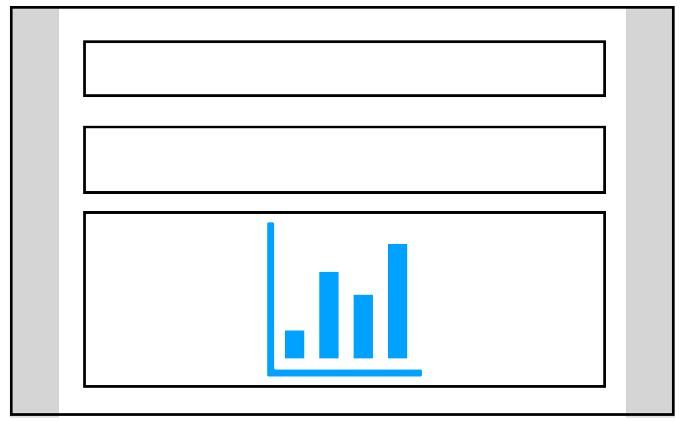
CodeScoop [CHI '18]

code editor | snippets | select | simplify



Torii [CHI '20]

code editor | tutorials | supplement | sequence



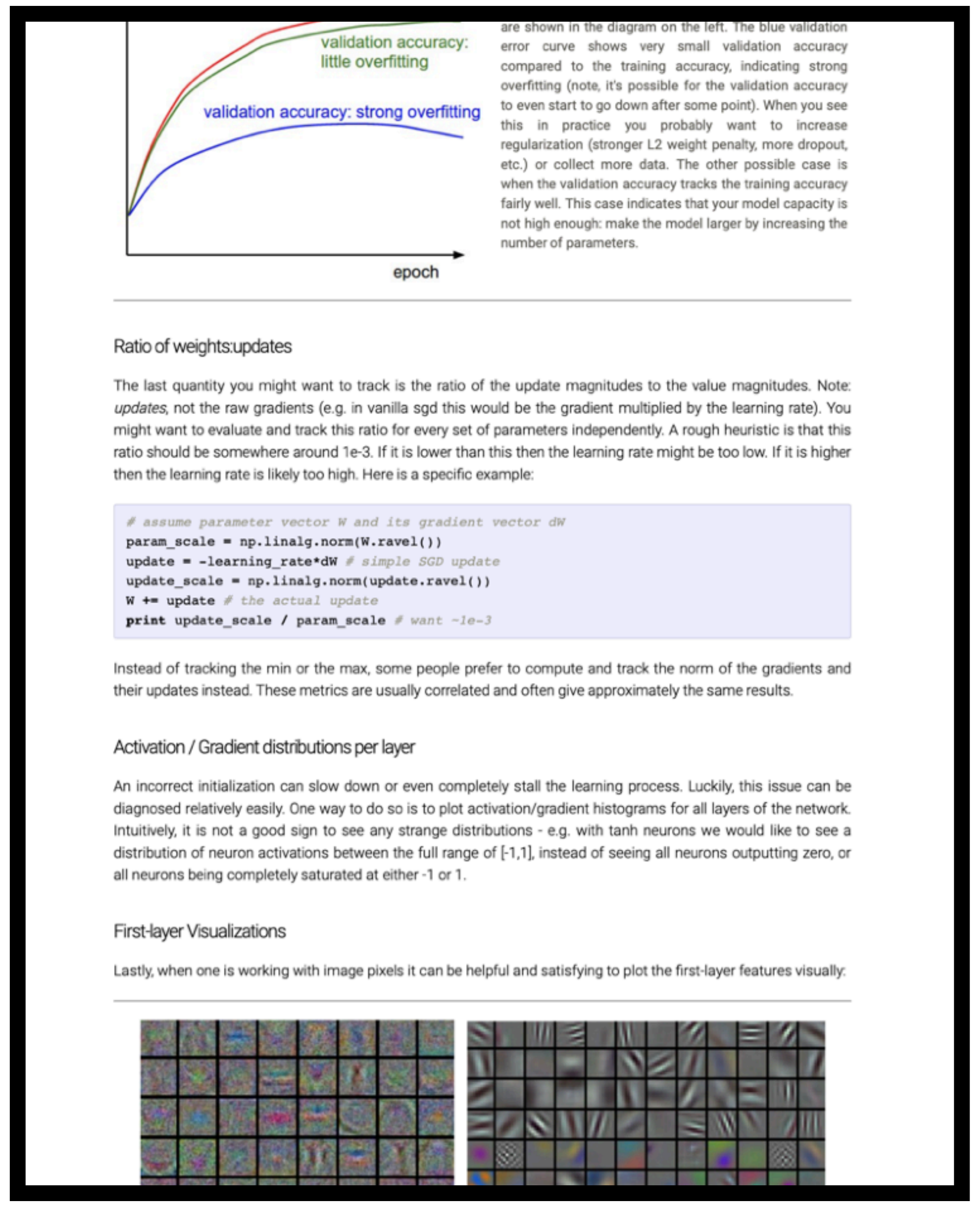
Gathering tools [CHI '19]

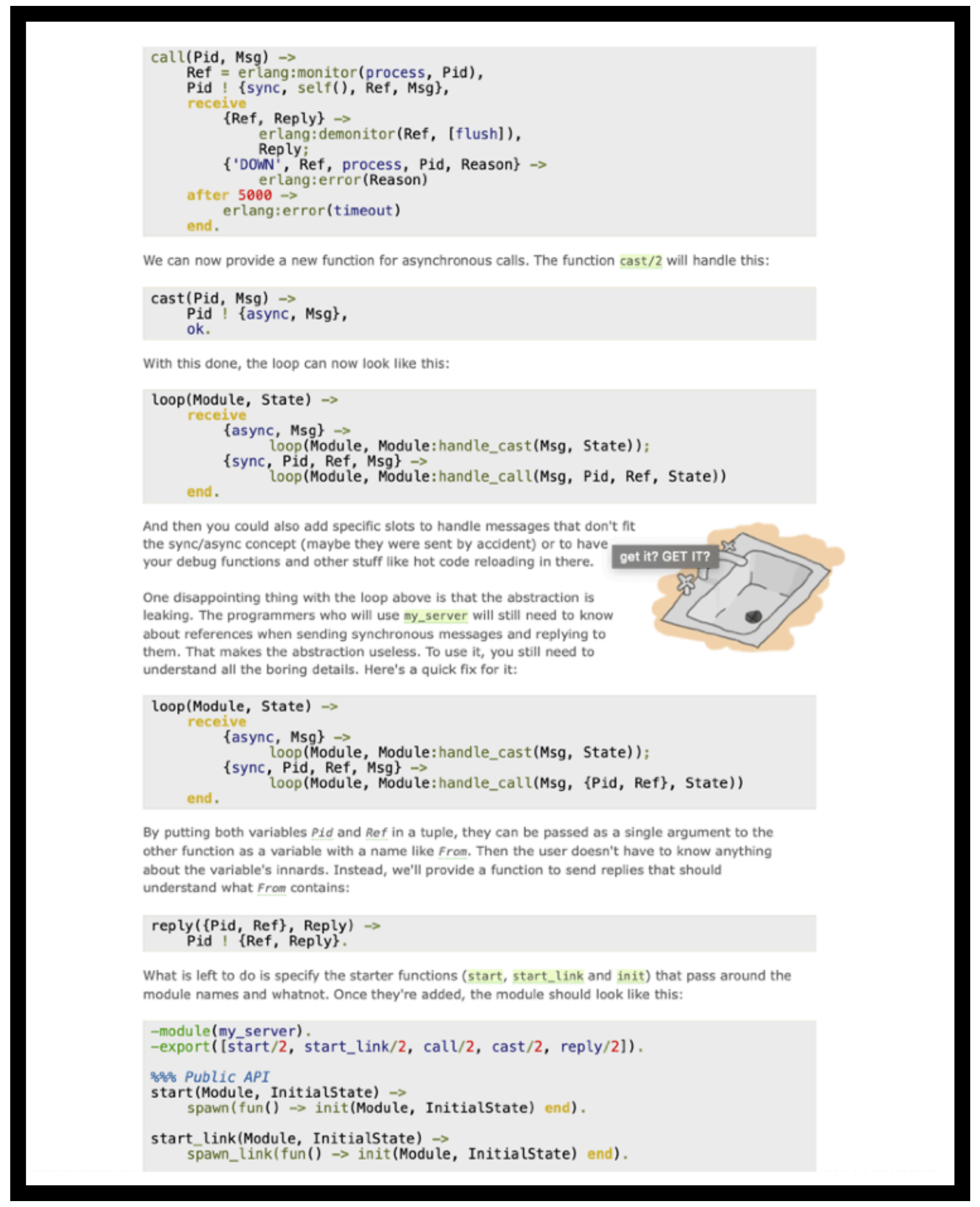
notebook | snippets | tutorials | select | supplement

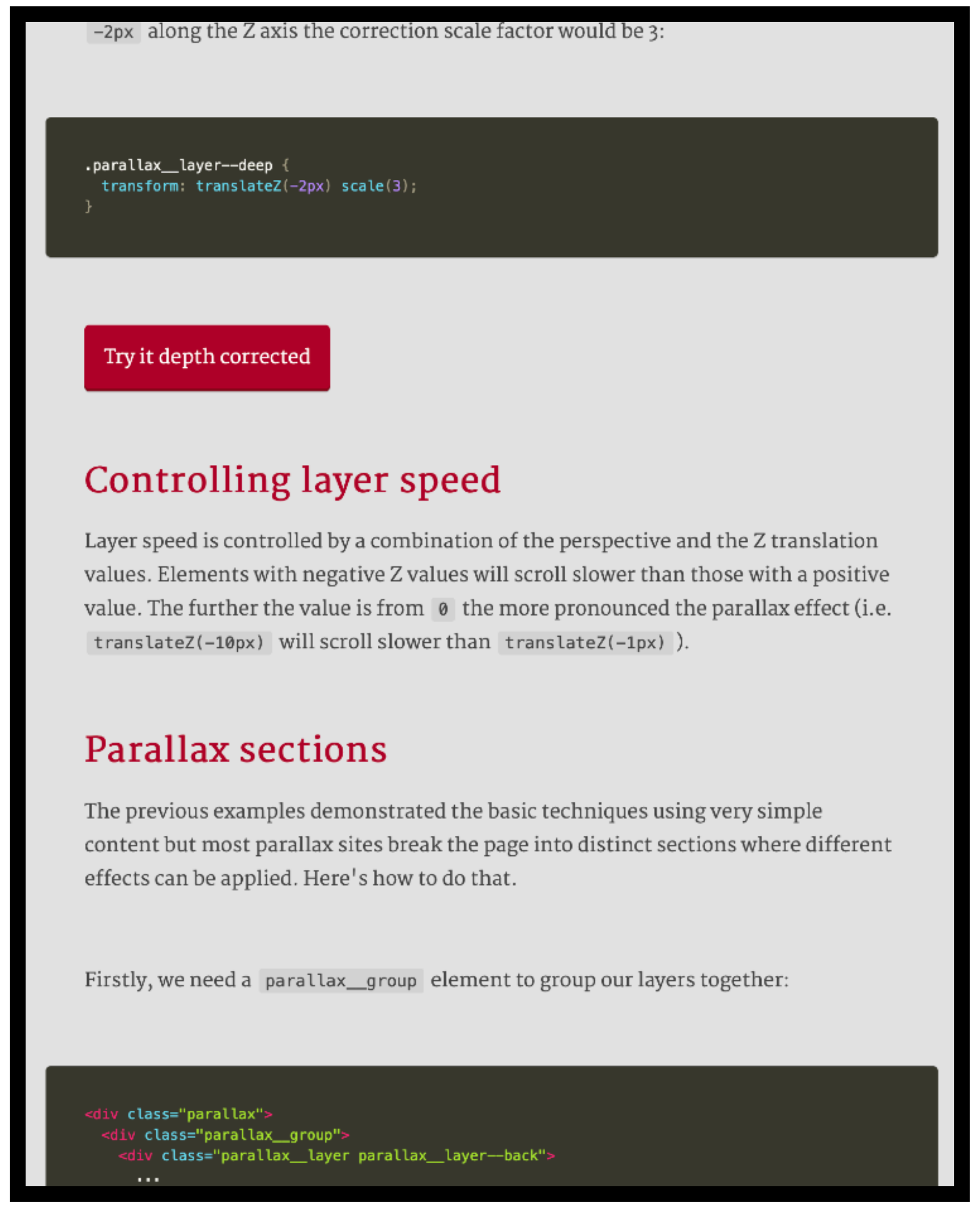
A Content Analysis of the contents, structure, and format of 200 programming tutorials

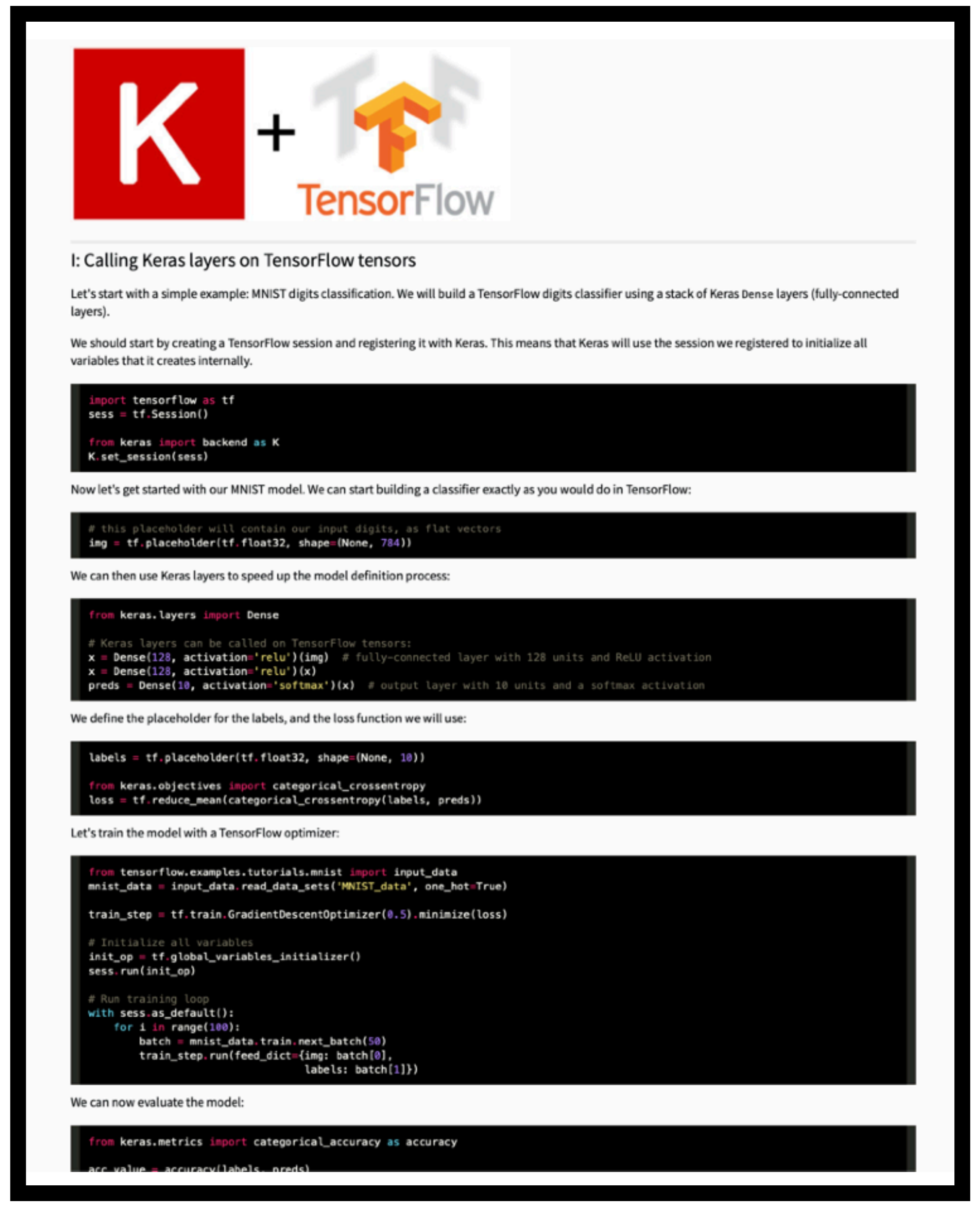
Written in many languages, about many topics.

Each tutorial was analyzed for the presence of 23 properties.



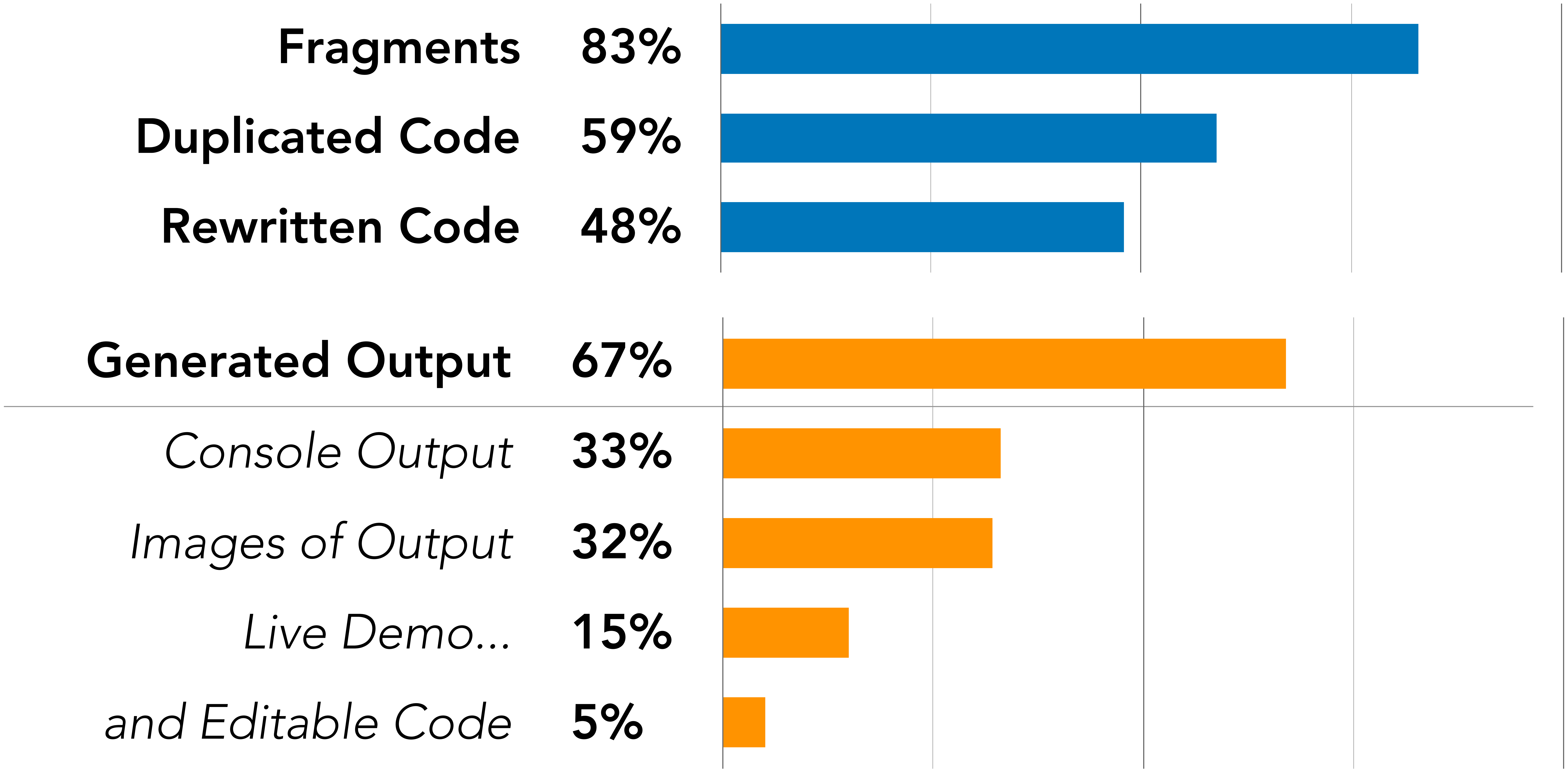






Structure and dependencies in tutorials

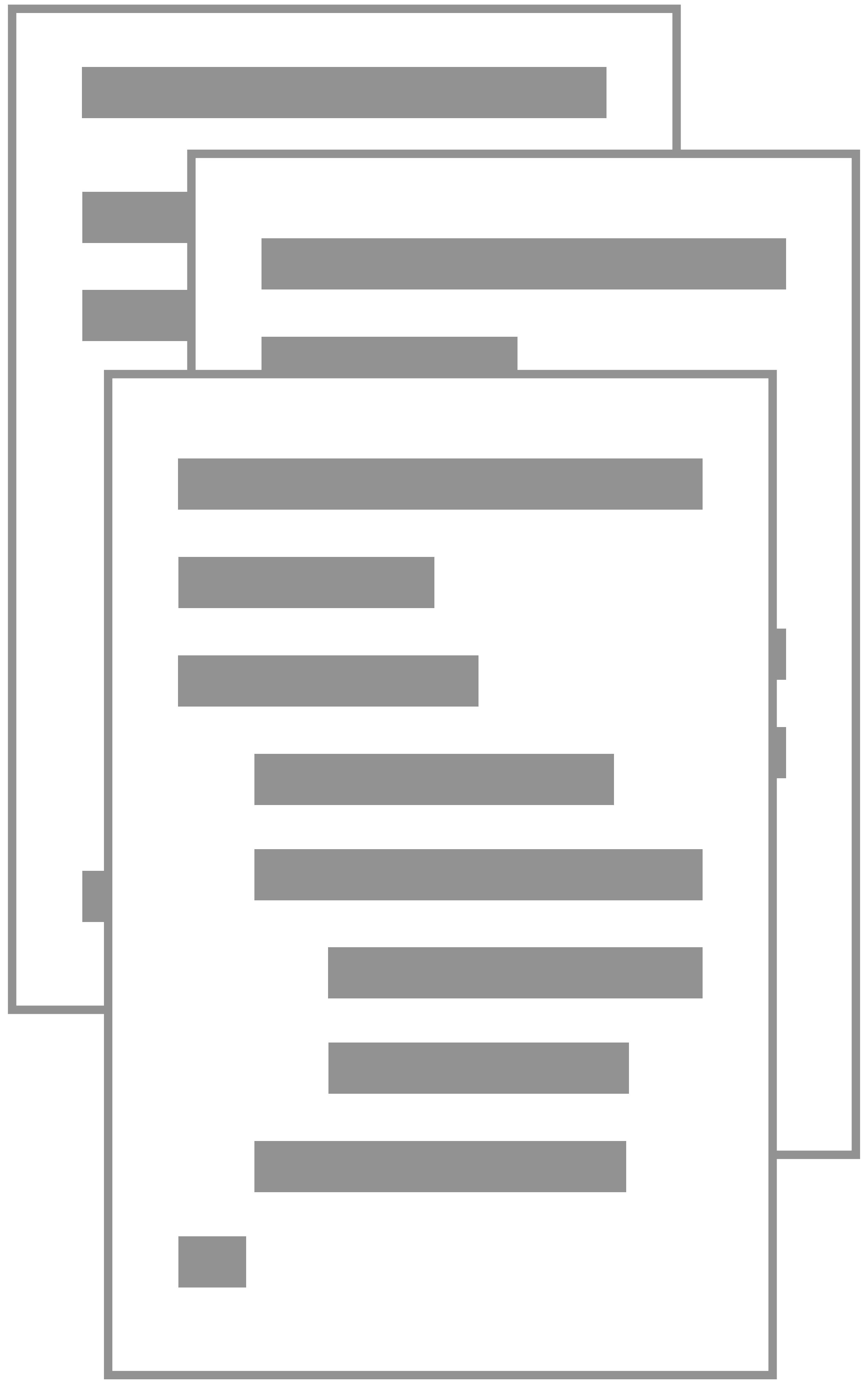
Structure and dependencies in tutorials



Interviews with 12 accomplished authors of programming tutorials.

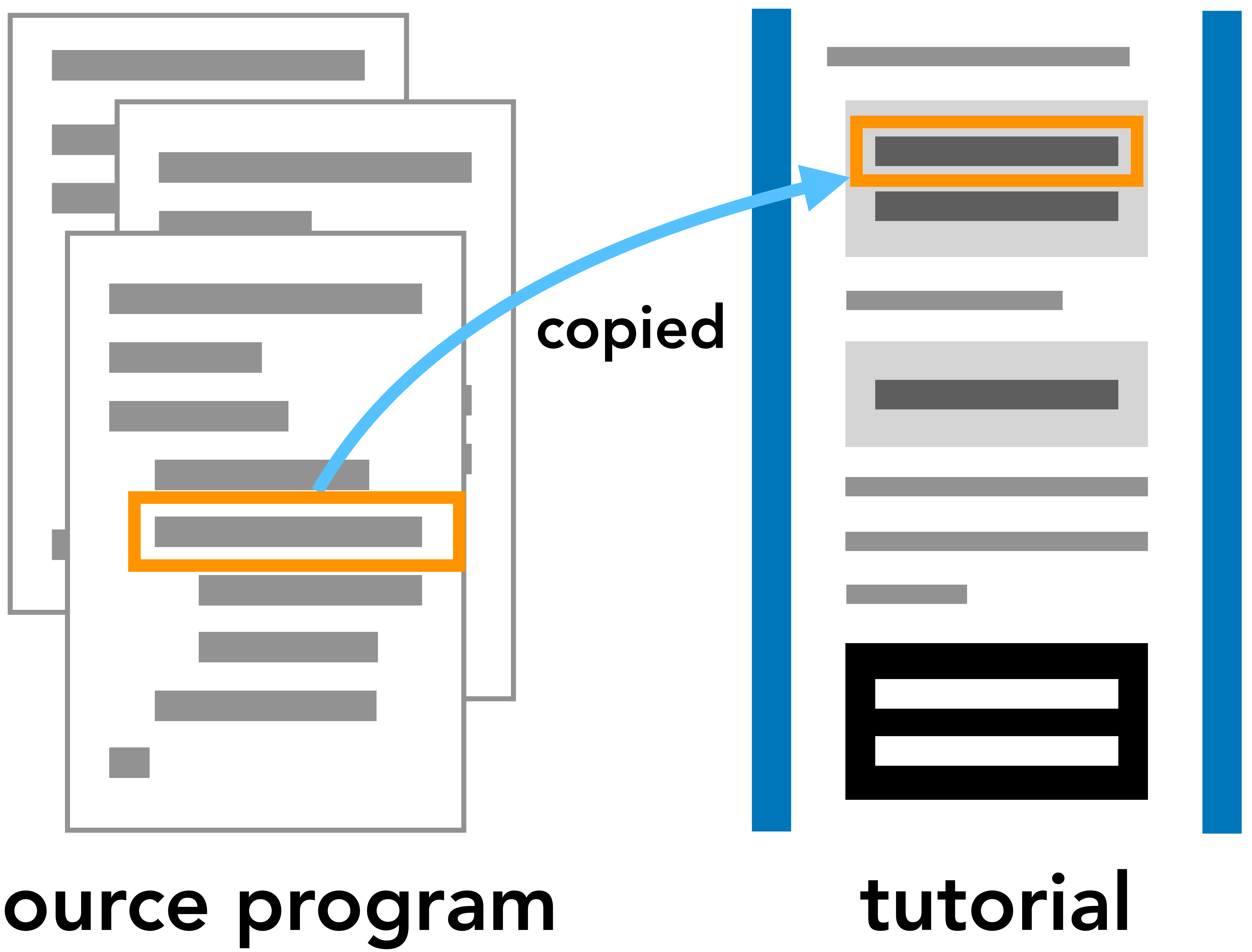
Each author had written between a few and hundreds of tutorials.

Sequencing code → parallel views of the code

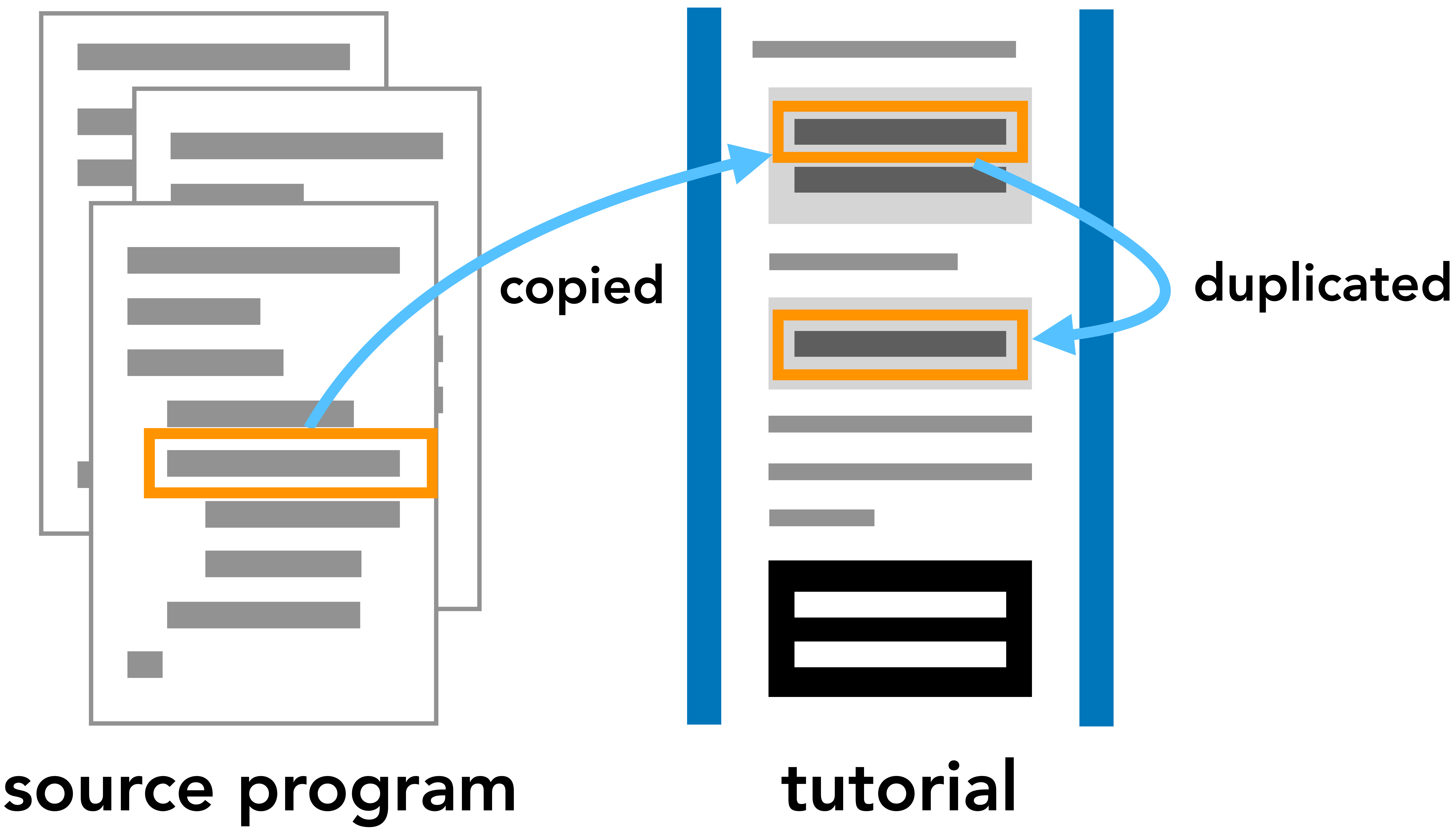


source program

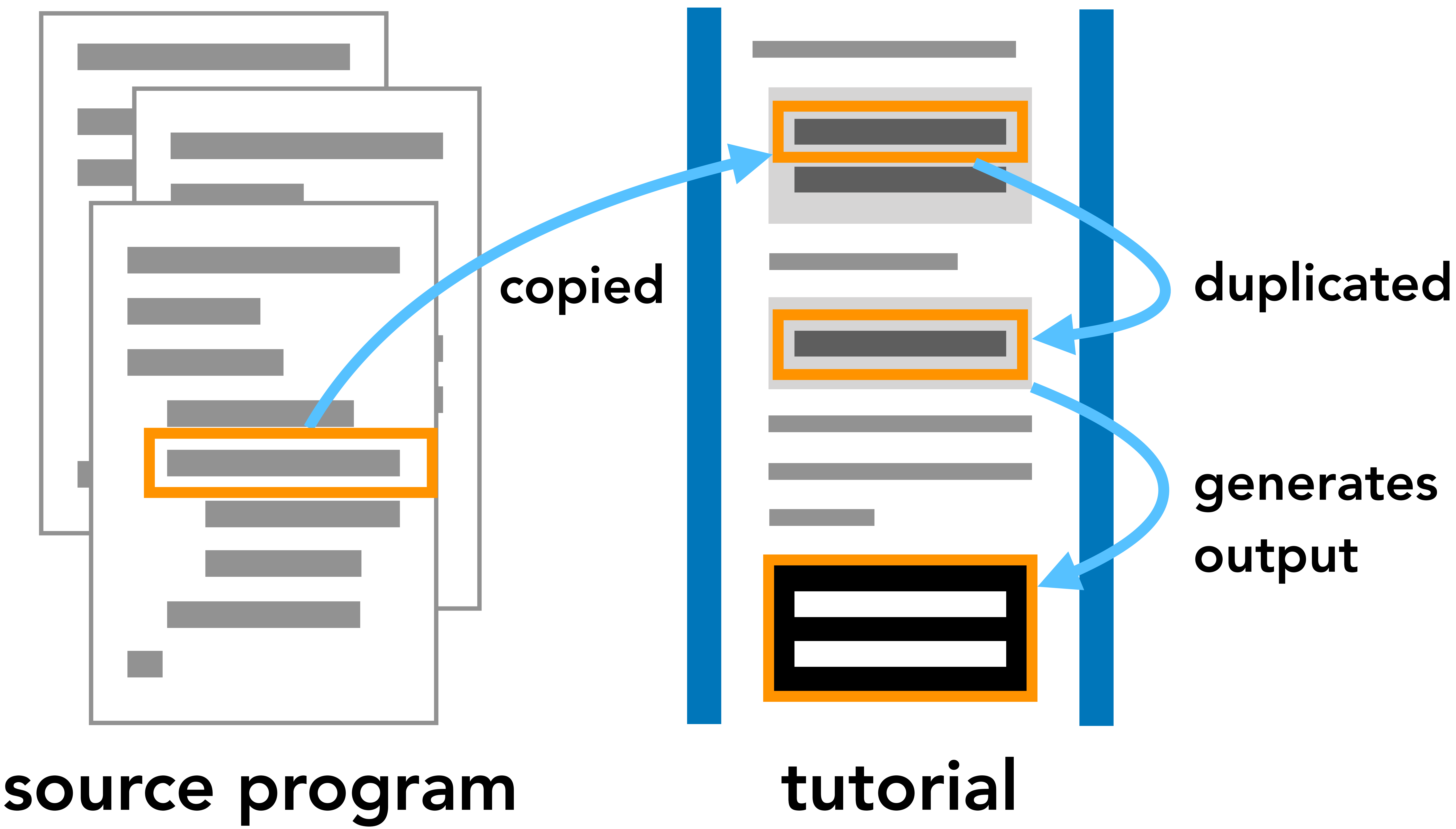
Sequencing code → parallel views of the code



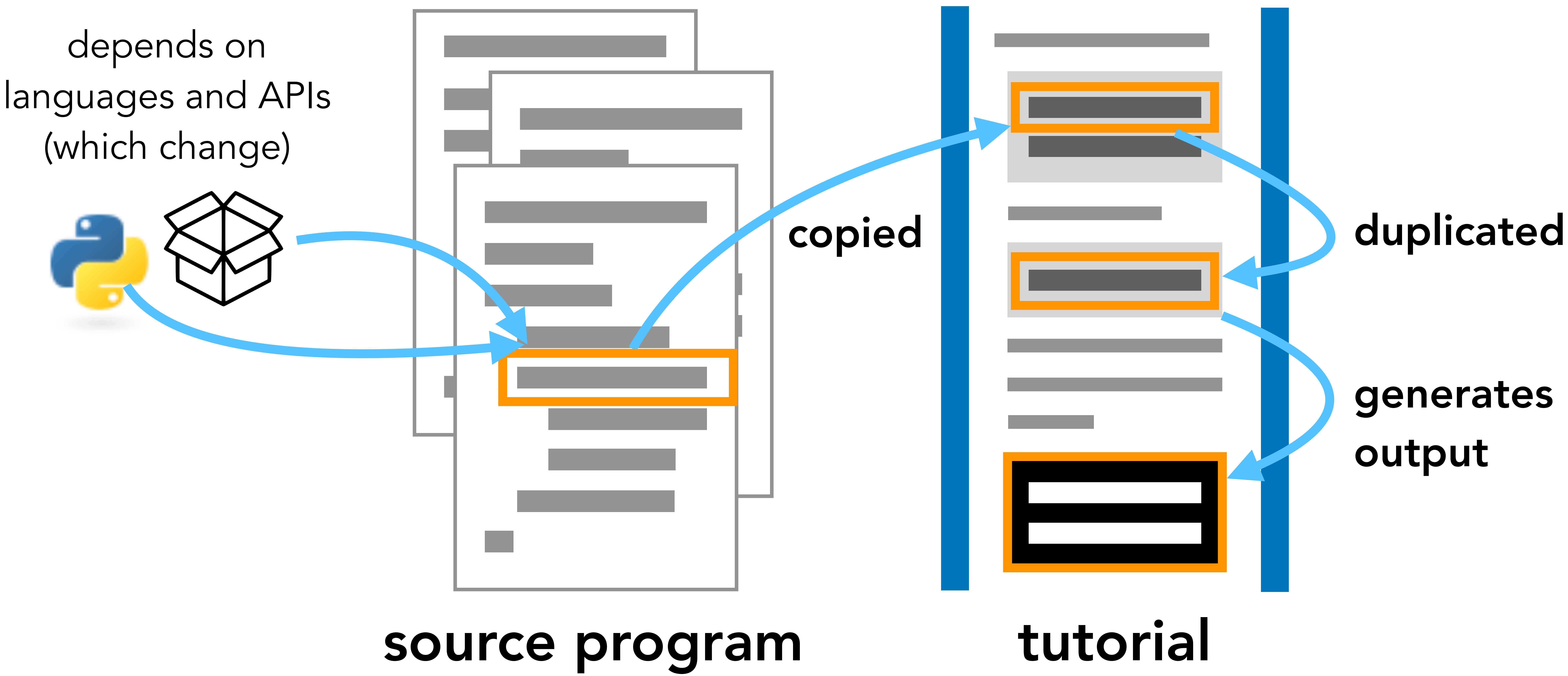
Sequencing code → parallel views of the code



Sequencing code → parallel views of the code



Sequencing code → parallel views of the code

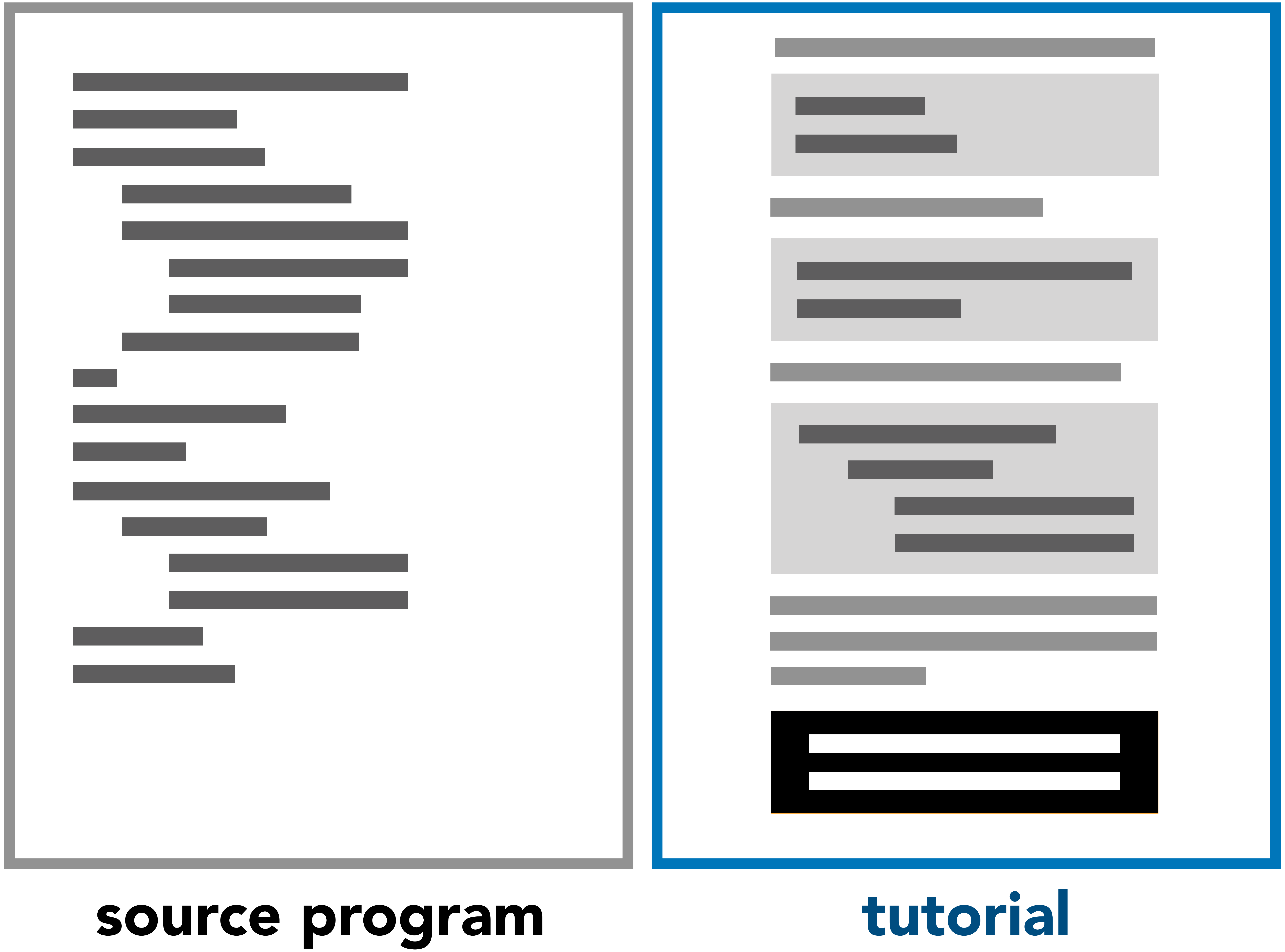


How to keep parallel views of code consistent

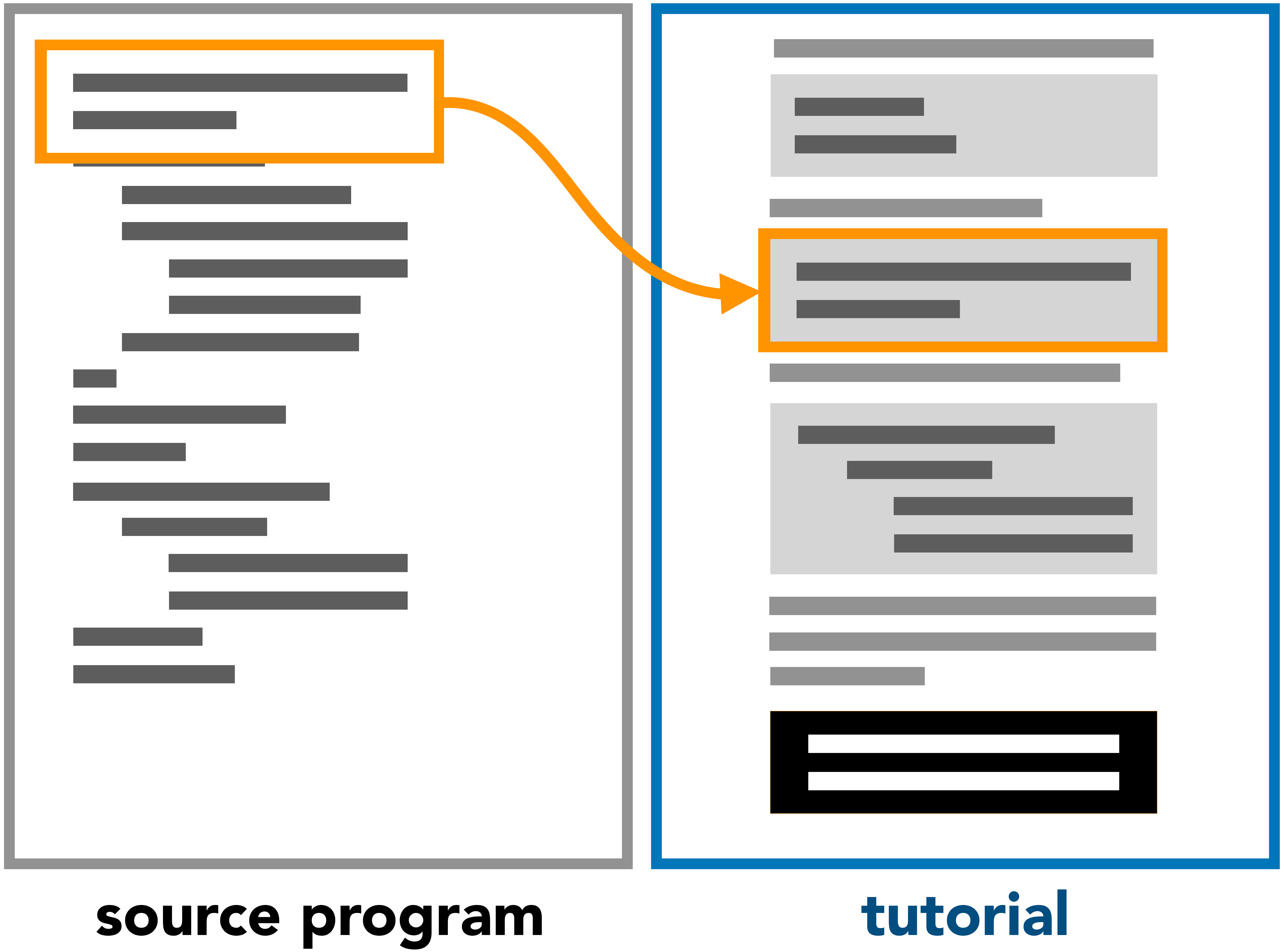
Proactive approaches: Starting with a reference implementation; Using version control.

Corrective approaches: Manually following one's own tutorial and checking the end result; Regenerating outputs as code changes.

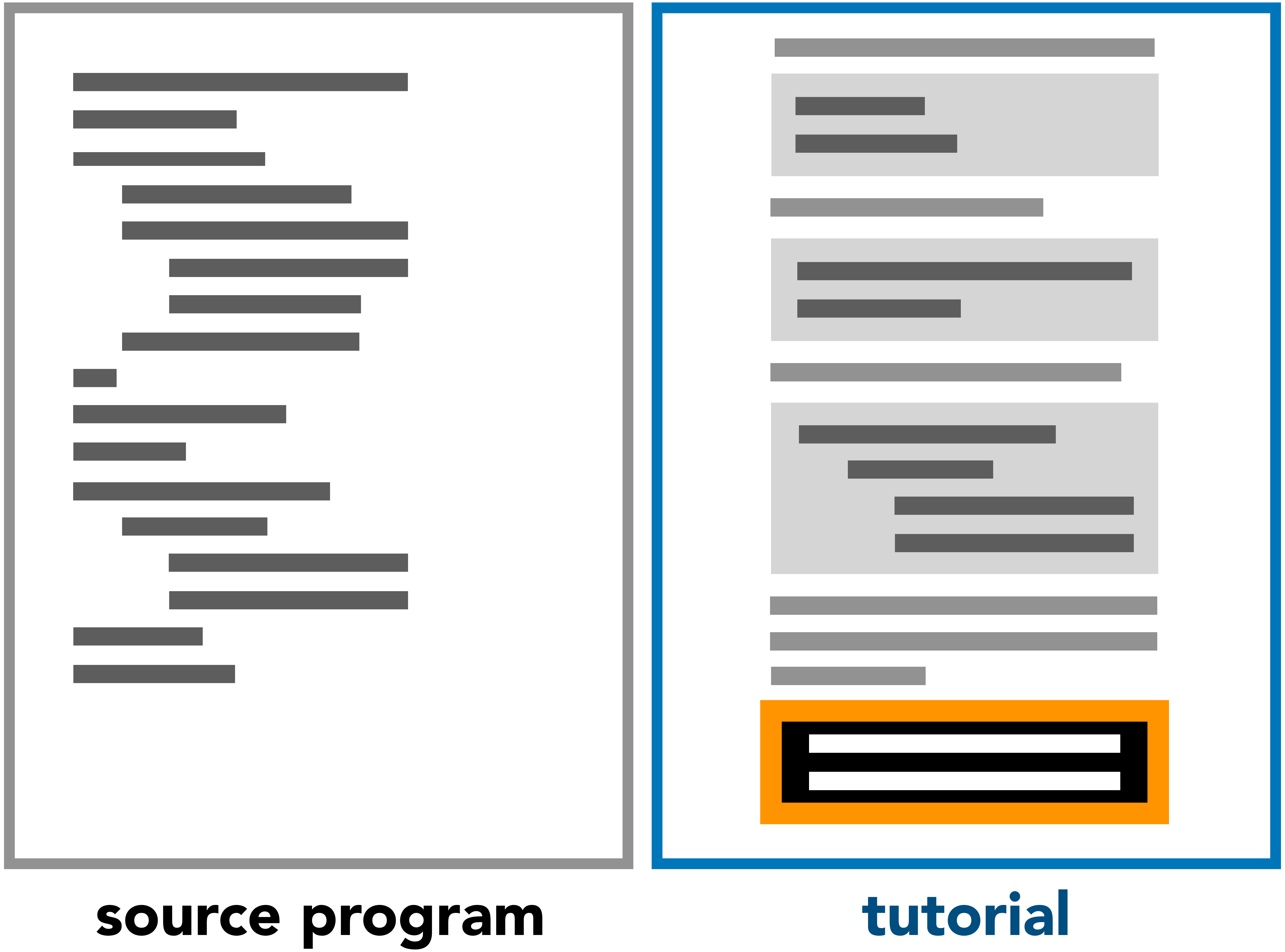
Simultaneous support of **sequencing** and **supplementing**



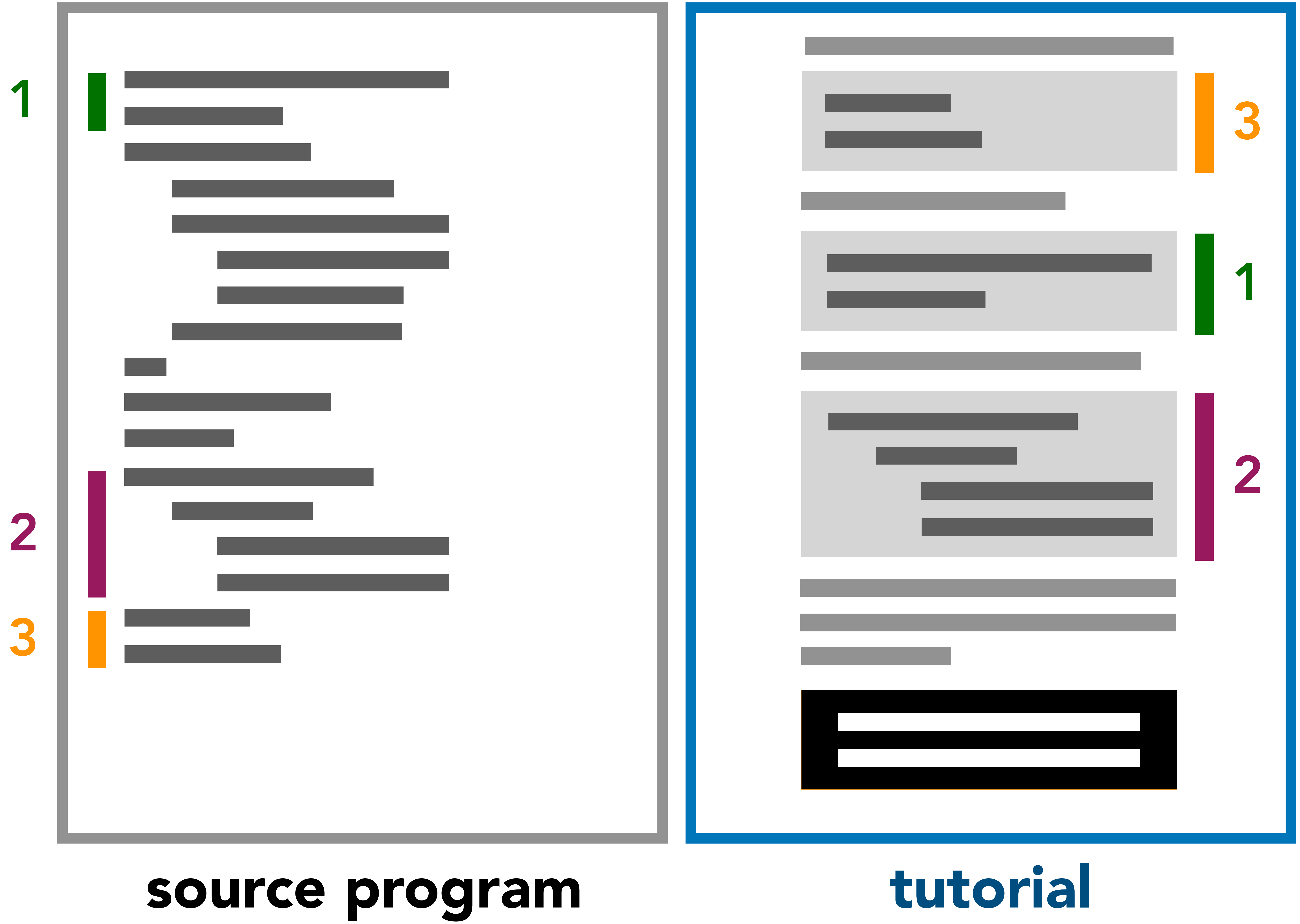
Simultaneous support of **sequencing** and **supplementing**



Simultaneous support of **sequencing** and **supplementing**

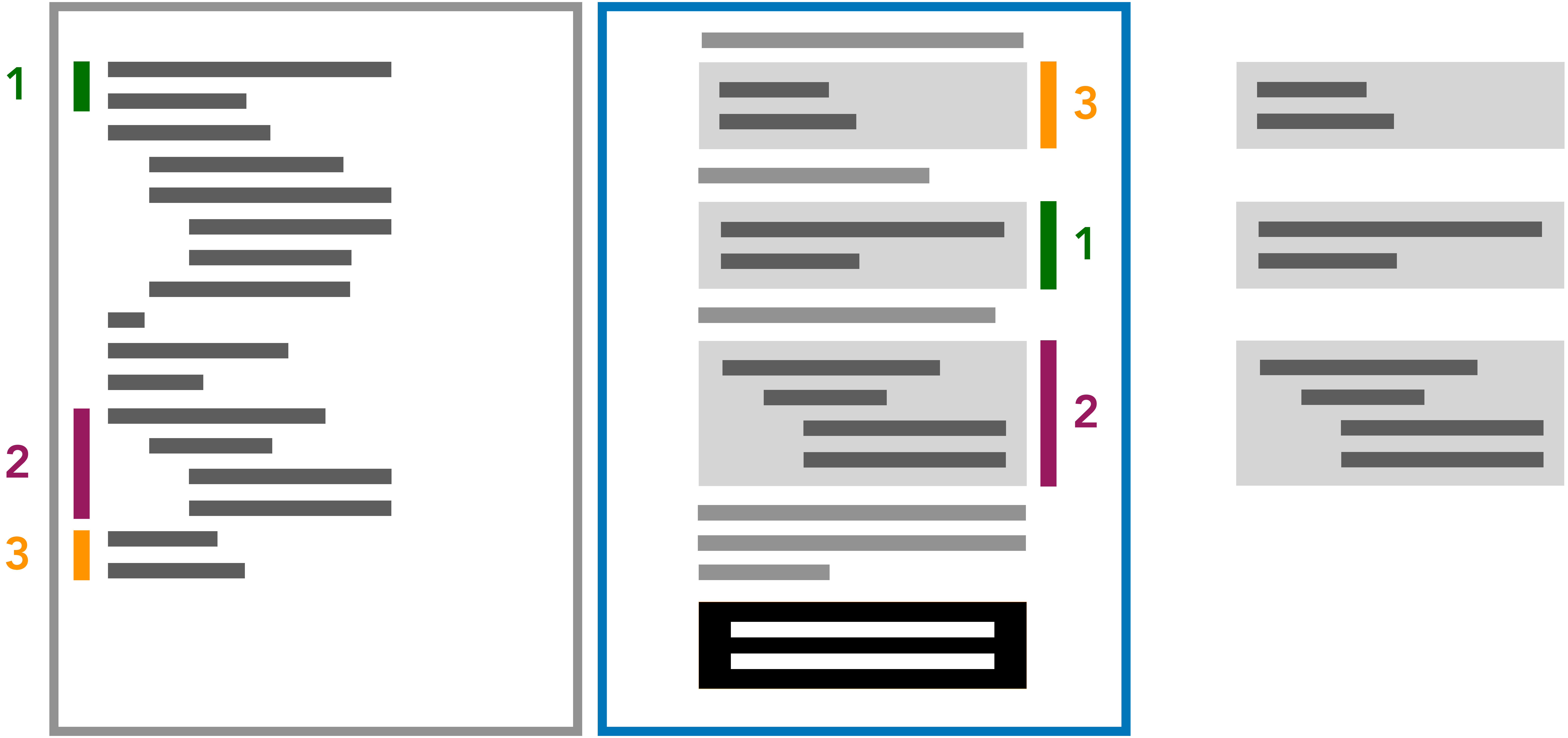


Simultaneous support of sequencing and supplementing

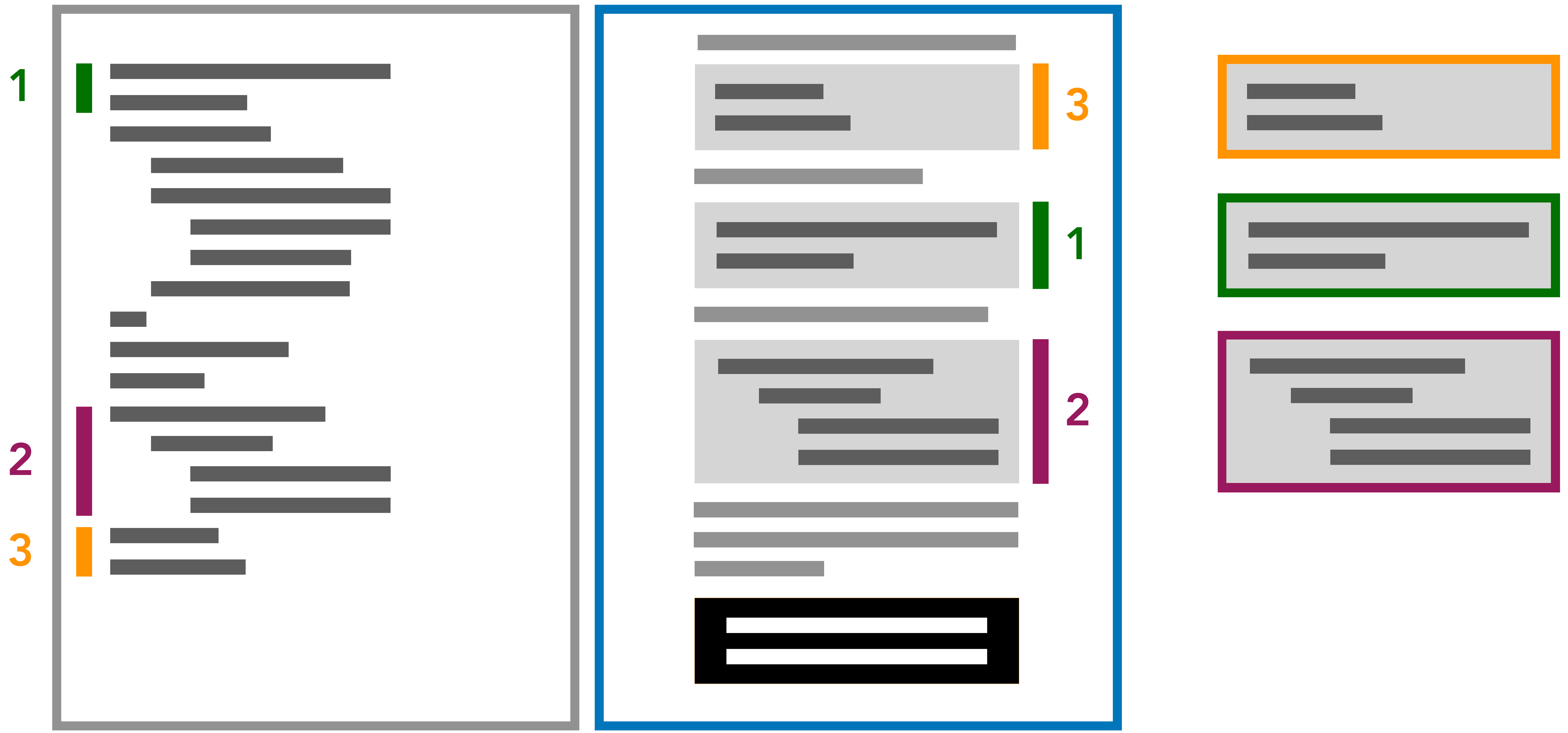


Simultaneous support of **sequencing** and **supplementing**

Simultaneous support of **sequencing** and **supplementing**



Simultaneous support of **sequencing** and **supplementing**



Simultaneous support of **sequencing** and **supplementing**

Simultaneous support of **sequencing** and **supplementing**

objects.py ×

objects.py > ...

```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)
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 i
```

Tutorial Editor ×

Torii   ADD TEXT   ADD SNIPPET   UNDO

TUTORIAL

ster*   Python 3.6.4 64-bit   0   0   Ln 1, Col 24   Spaces: 4   UTF-8   LF   Python objects.py × | Tutorial Editor ×

objects.py > ...

```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)
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 i
```

Torii | ADD TEXT | ADD SNIPPET | UNDO

TORII

aster* | Python 3.6.4 64-bit | 0 | 0 | Ln 1, Col 24 | Spaces: 4 | UTF-8 | LF | Python objects.py | Tutorial Editor

Torii | ADD TEXT | ADD SNIPPET | UNDO

```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)
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 i
```

Python 3.6.4 64-bit | Ln 1, Col 24 | Spaces: 4 | UTF-8 | LF | Python

rich text

objects.py

```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)
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 i
```

Tutorial Editor

Torii

ADD TEXT | ADD SNIPPET | UNDO

Write | Preview

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?

rich text

snippets

Python 3.6.4 64-bit

objects.py

objects.py > rectangle

```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)
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 i
```

Python 3.6.4 64-bit ⊗ 0 ⚠ 0

Tutorial Editor

Torii   ADD TEXT   ADD SNIPPET   UNDO

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?

VIEW AS   SNIPPET   PROGRAM SNAPSHOT

```python
rectangle = Rectangle(90, 45)
```

rich text

snippets

linked edits

objects.py

objects.py > ...

```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)
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 i
```

Tutorial Editor

Torii   ADD TEXT   ADD SNIPPET   UNDO

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?

VIEW AS   SNIPPET   PROGRAM SNAPSHOT

```python
rectangle = Rectangle(90, 45)
```

ADD CONSOLE OUTPUT

ster*   Python 3.6.4 64-bit   ⊗ 0 ⚠ 0

rich text

snippets

linked edits

objects.py ●

objects.py > Rectangle

```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)
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 i
```

Tutorial Editor ×

Torii   ADD TEXT   ADD SNIPPET   UNDO

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)
```

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 of 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.

```python
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"
```

Write   Preview

ster*   Python 3.6.4 64-bit   ⊗ 0 ⚠ 0

rich text

snippets

linked edits

objects.py

objects.py > ...

```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)
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 i
```

Tutorial Editor

Torii ADD TEXT ADD SNIPPET UNDO

```python
        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"
```

ADD CONSOLE OUTPUT

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:

VIEW AS SNIPPET PROGRAM SNAPSHOT

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

rich text

outputs

snippets

linked edits

Python 3.6.4 64-bit ⓧ 0 ⚠ 0

objects.py ×

objects.py > ...

```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)
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 i
```

aster* Python 3.6.4 64-bit ⊗ 0 ⚠ 0

Tutorial Editor ×

Torii ADD TEXT ADD SNIPPET UNDO

and methods with a call that looked like this?

```python
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 of 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.

```python
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
```

rich text

outputs

output updates

snippets

linked edits

objects.py ×

objects.py > ...

```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)
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 i
```

aster* Python 3.6.4 64-bit ⊗ 0 ⚠ 0

Tutorial Editor ×

Torii ADD TEXT ADD SNIPPET UNDO

and methods with a call that looked like this?

```python
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 of 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.

```python
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
```

rich text

outputs

output updates

snippets

linked edits


objects.py
objects.py > ...
# 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, sca
self.w = self.w * s
self.h = self.h * s
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 i
Python 3.6.4 64-bit
Ln 25, Col 1 Spaces: 4 UTF-8 LF Python
Tutorial Editor
Torii
ADD TEXT
ADD SNIPPET
UNDO
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.
self.w = w
self.h = h
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
What if we want to define other properties of an object when we create it? Well in that
repeated code
rich text
outputs
output updates
snippets
linked edits


objects.py
objects.py > ...
# 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):
class Square(Rectang
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 i
fragmented code
Tutorial Editor
Torii
ADD TEXT
ADD SNIPPET
UNDO
print("And its description is: " + str(rectangle.description))
The width of the rectangle is: 90
And its description is: This is a rectangle
Object data can be accessed as properties. Common object operations can be accessed as methods. The simplest type of method just gets some data from the object. One example of this is the list.indexOf operation on lists.
Let's add a method to the shape objects that gets the area of the object. Place this method right underneat the __init__ method in the Shape class.
def area(self):
return self.w * self.h
What does this method do? When this method is called (like so...)
print("The area of the rectangle is: " + str(rectangle.area()))
Then the area def gets invoked. The object is passed in as the self parameter. The properties on the object---like width and height---are accesible 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
And its description is: This is a rectangle
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 3.6.4 64-bit
Ln 25, Col 1 Spaces: 4 UTF-8 LF Python
rich text
outputs
output updates
snippets
linked edits

objects.py ● ···

objects.py > Square

```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)
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 i
```

Tutorial Editor ×

Torii   ADD TEXT   ADD SNIPPET   UNDO

What does this method do? When this method is called (like so...)

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

Then the `area` def gets invoked. The object is passed in as the `self` parameter. The properties on the object---like width and height---are accesible 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:

VIEW AS   SNIPPET   PROGRAM SNAPSHOT

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

ADD CONSOLE OUTPUT

ster*   Python 3.6.4 64-bit   ⊗ 0 ⚠ 0

rich text

outputs

output updates

snippets

linked edits

hidden snippets

objects.py ●

objects.py > Square

```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)
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 i
```

ster* Python 3.6.4 64-bit ⊗ 0 ⚠ 0

Tutorial Editor ×

Torii   ADD TEXT   ADD SNIPPET   UNDO

What does this method do? When this method is called (like so...)

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

Then the `area` def gets invoked. The object is passed in as the `self` parameter. The properties on the object---like width and height---are accesible 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:

rich text

outputs

output updates

snippets

linked edits

hidden snippets

objects.py ●

objects.py > square

```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 i

# 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))
```

ster* Python 3.6.4 64-bit ⓧ 0 ⚠ 0

Tutorial Editor ×

Torii ADD TEXT ADD SNIPPET UNDO

What does this method do? When this method is called (like so...)

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

Then the `area` def gets invoked. The object is passed in as the `self` parameter. The properties on the object---like width and height---are accesible 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
square = Square(100)
print("The square's width: " + str(square.w))
print("The square's height: " + str(square.h))
```

ADD CONSOLE OUTPUT

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

rich text

outputs

output updates

snippets

linked edits

hidden snippets

objects.py ●

objects.py > square

```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 i

# 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))
```

Tutorial Editor ×

Torii ADD TEXT ADD SNIPPET UNDO

What does this method do? When this method is called (like so...)

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

Then the `area` def gets invoked. The object is passed in as the `self` parameter. The properties on the object---like width and height---are accesible 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
square = Square(100)
print("The square's width: " + str(square.w))
print("The square's height: " + str(square.h))
```

ADD CONSOLE OUTPUT

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

ster* Python 3.6.4 64-bit ⊗ 0 ⚠ 0

rich text

outputs

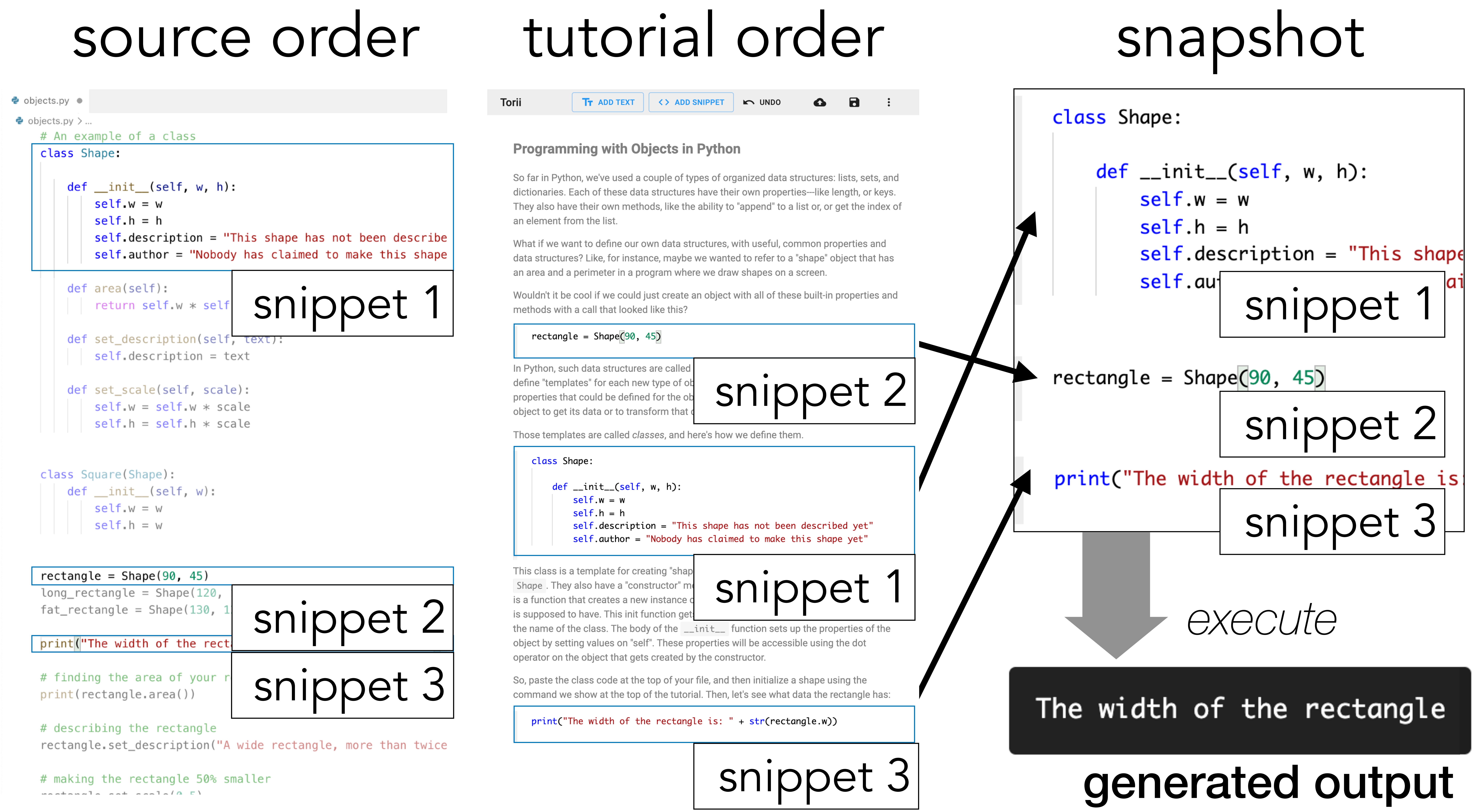
output updates

snippets

linked edits

hidden snippets

Program Analysis



Evaluating Torii

Q1. Does Torii support ***efficient*** editing of tutorials?

Q2. Does Torii support ***flexible*** snippet organization?

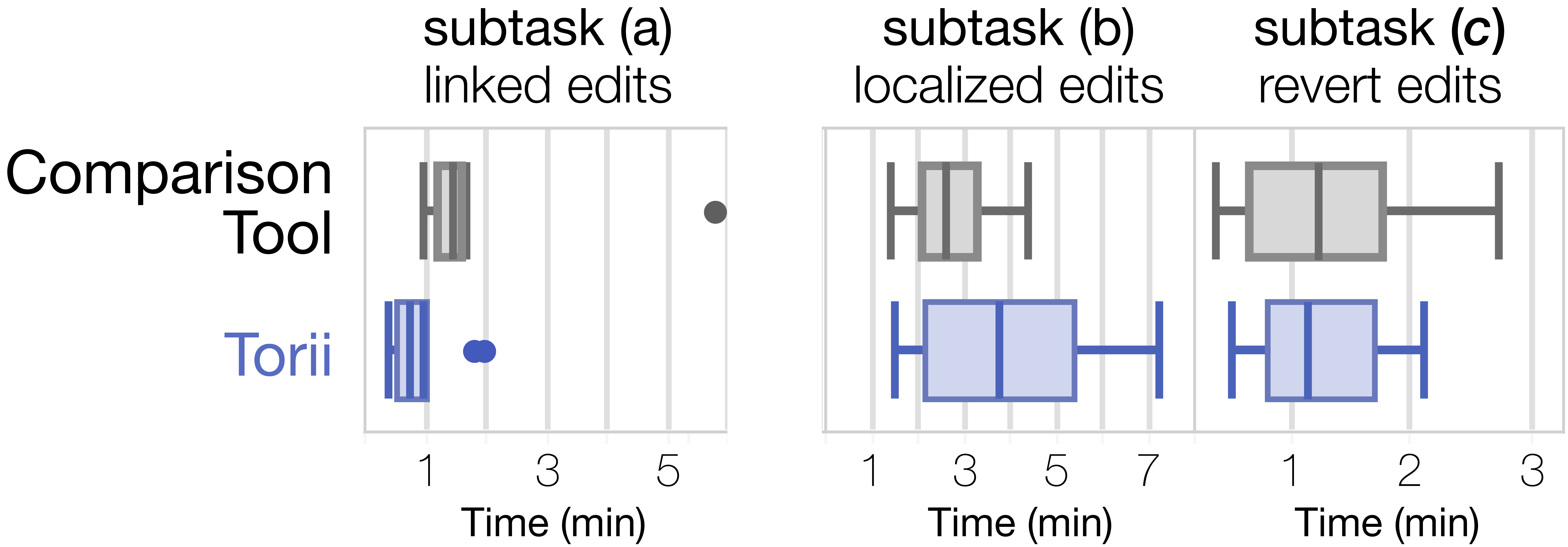
An In-Lab Study of Torii

Participants: *N = 12* tutorial authors

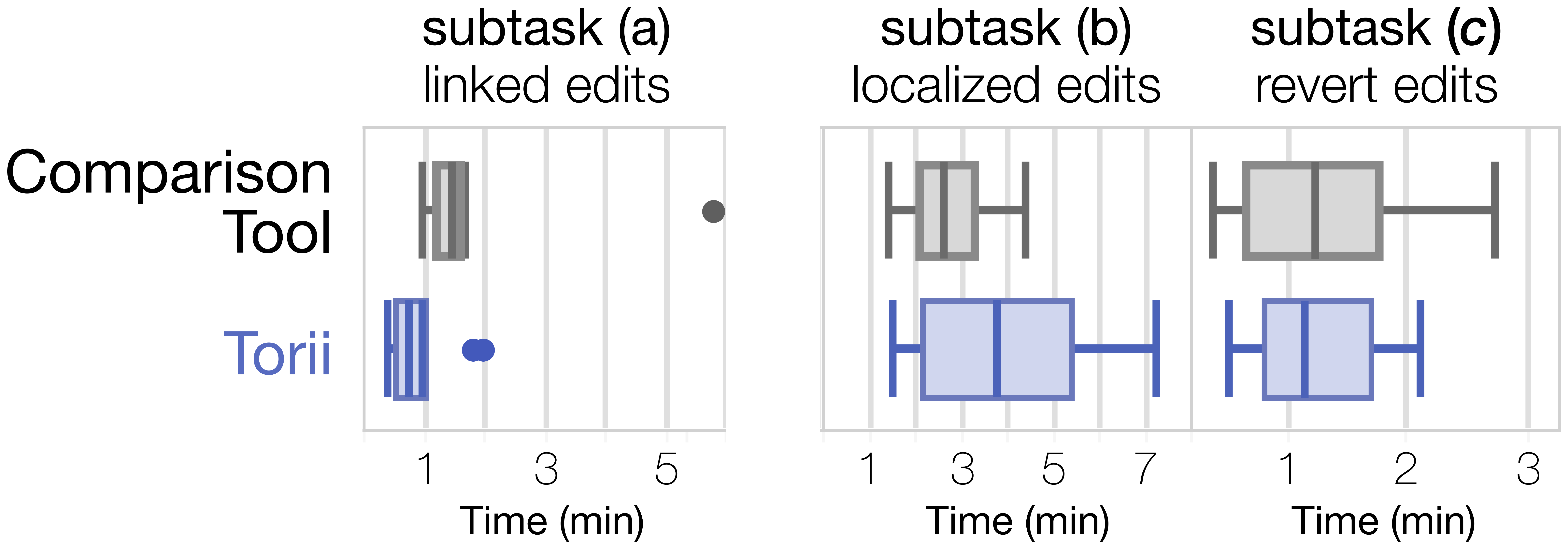
Maintenance Task × 2: Update a tutorial, with Torii and with a baseline tool.

Exploration: Create a tutorial about object-oriented programming, based on an existing program.

Efficiency of editing tasks with Torii



Efficiency of editing tasks with Torii



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 we will be using for the remainder of the tutorial.

```
class Shape:
```

code fragments

From here, we need to find a w… For a shape, some things we might care about are the "w… and "height". In order to portray this, we would include these as instance variables … what we will call a constructor:

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

Every time, we go to create a new "Shape" typ… for the "init" of the shape class and know that an… a "w" and an "h" as its arguments. From there, it … attributes using the body of the constructor:

repeated, embellished code

```
    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:

```
rectangle = Shape(100, 45)
long_rectangle = Shape(120, 10)
fat_rectangle = Shape(130, 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 attribu…

hidden code

name.attribute

A real example of this can be seen here:

```
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:

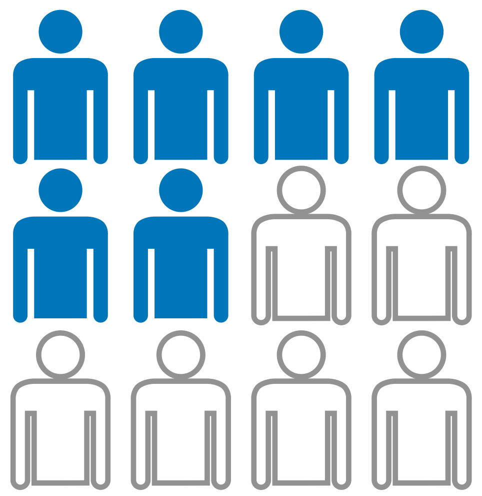
```
100
```

generated output

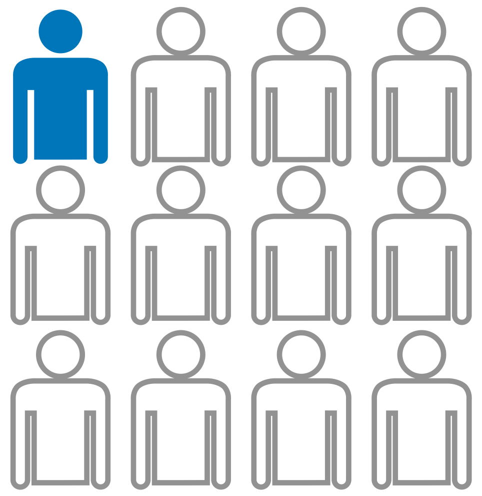
But where did this "Square" … square is really a shape, so… and height.

Because a sqare is so simil… of our Shape class'

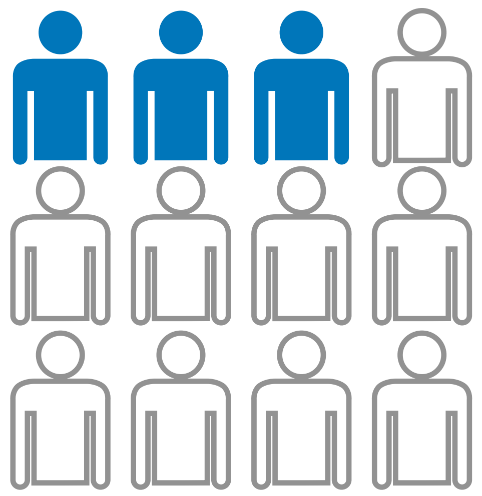
Presenting snippets flexibly



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

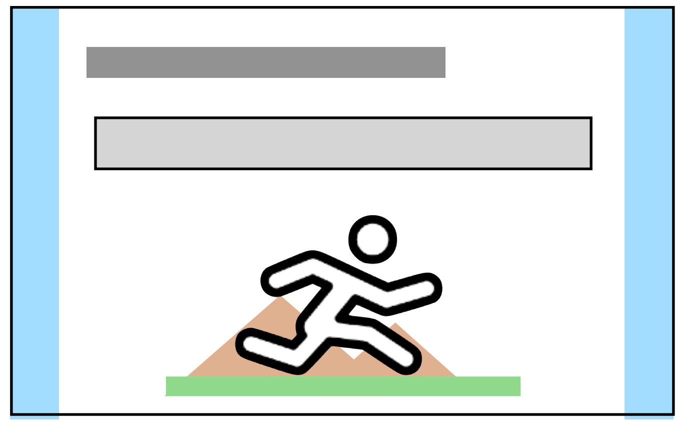
Q1. Authors could perform linked edits more quicky with Torii than a baseline tool, though this is not statistically significant.

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

This Talk

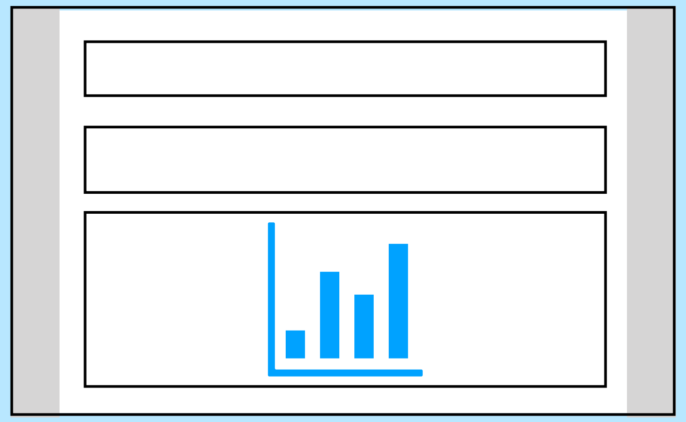
CodeScoop [CHI '18]

code editor | snippets | select | simplify



Torii [CHI '20]

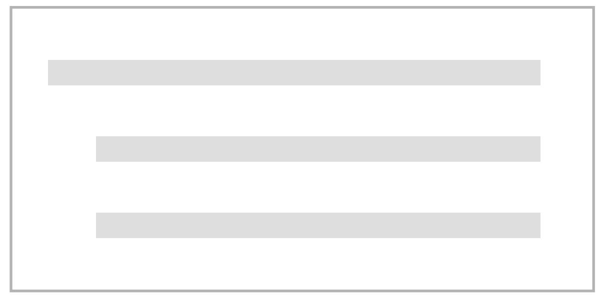
code editor | tutorials | supplement | sequence



Gathering tools [CHI '19]

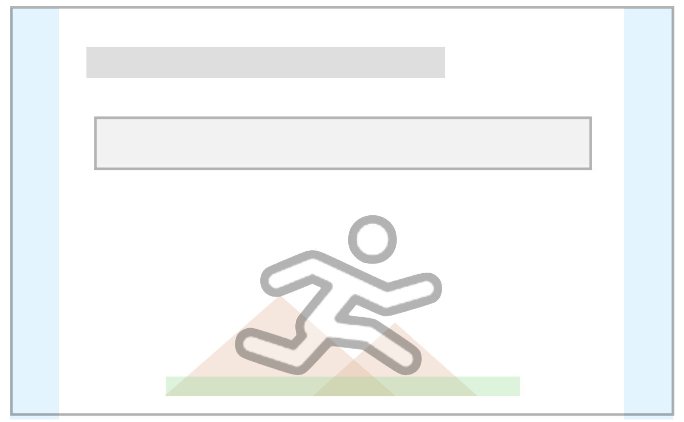
notebook | snippets | tutorials | select | supplement

This Talk



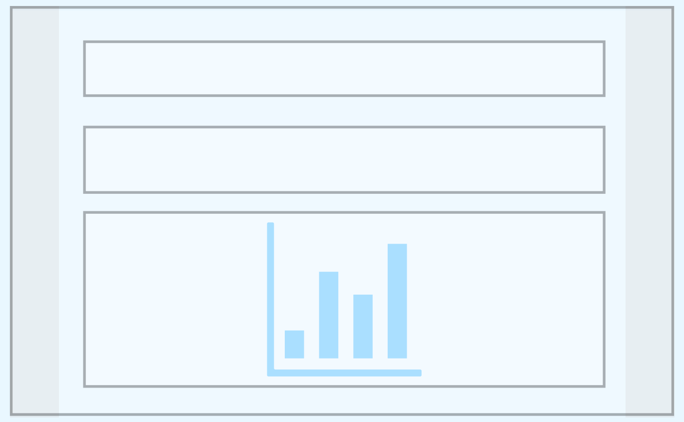
CodeScoop [CHI '18]

code editor | snippets | select | simplify



Torii [CHI '20]

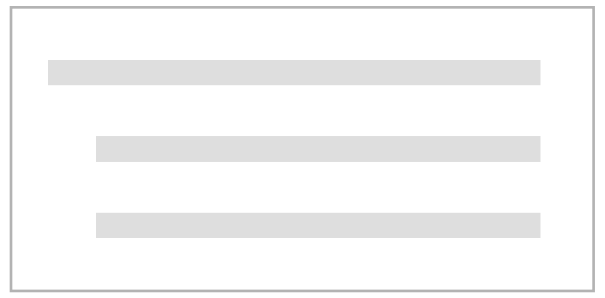
code editor | tutorials | supplement | sequence



Gathering tools [CHI '19]

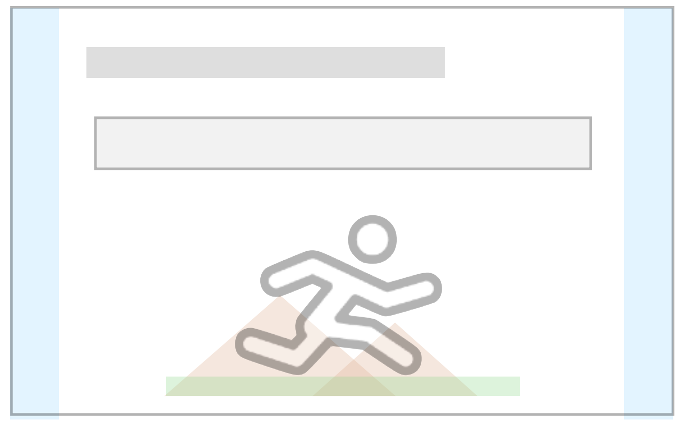
notebook | snippets | tutorials | select | supplement

This Talk



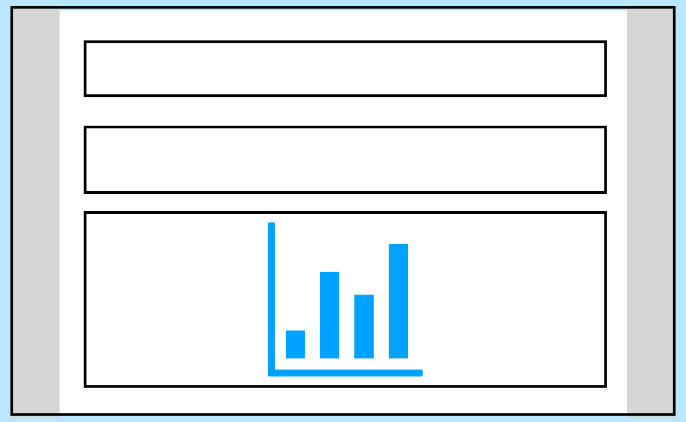
CodeScoop [CHI '18]

code editor | snippets | select | simplify



Torii [CHI '20]

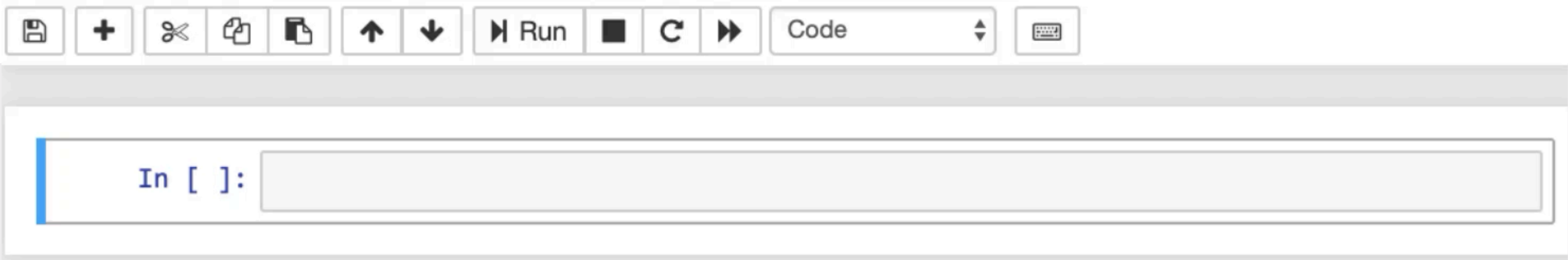
code editor | tutorials | supplement | sequence



Gathering tools [CHI '19]

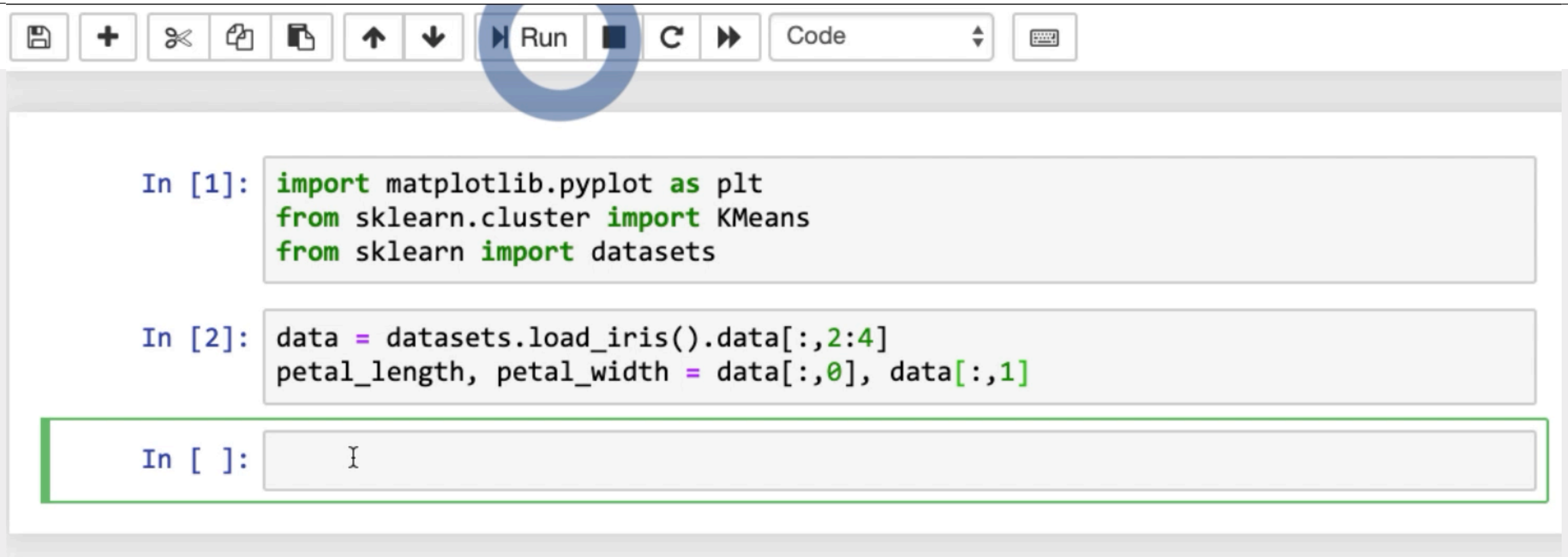
notebook | snippets | tutorials | select | supplement

Notebook Model of Exploratory Programming



1. Incremental execution

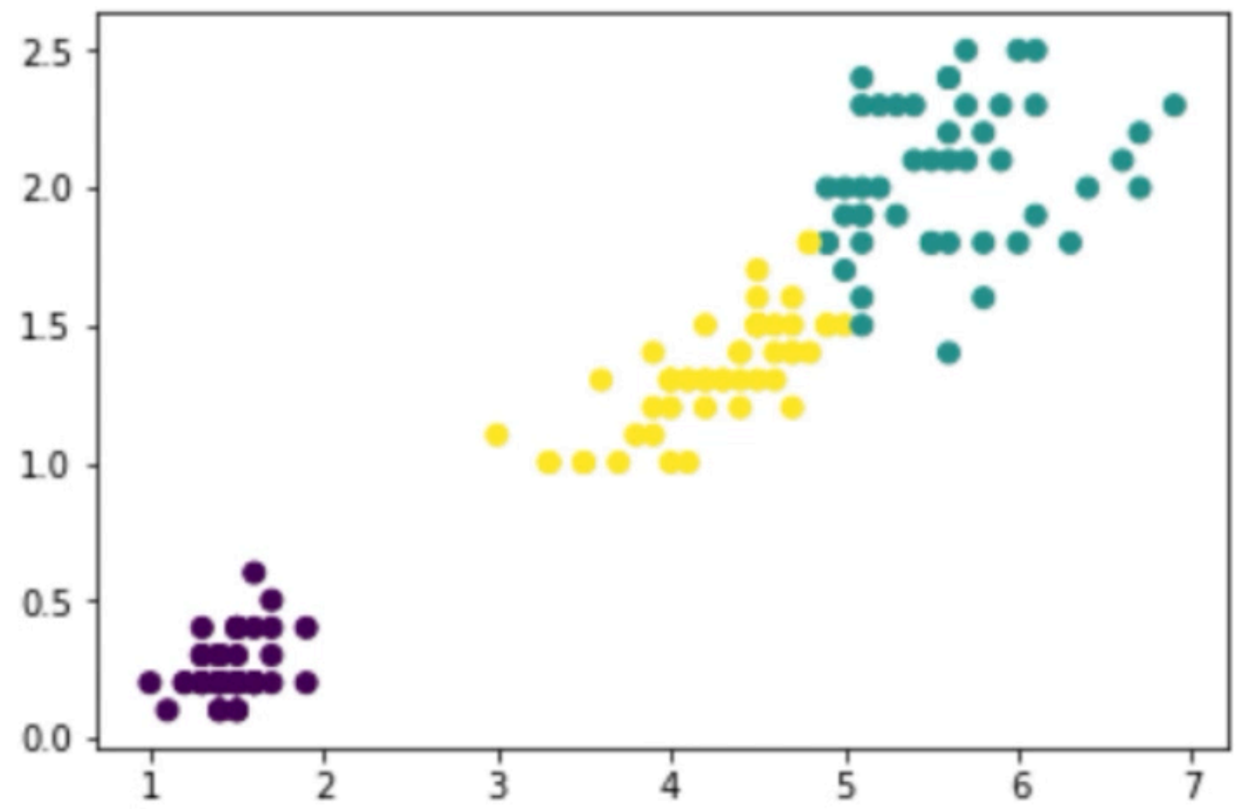
Notebook Model of Exploratory Programming



1. Incremental execution
2. In-situ output

Notebook Model of Exploratory Programming

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

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

In [3]: print("Average petal length: %.3f" % (sum(petal_length) / len(petal_length),))

        Average petal length: 3.758

In [4]: clusters = KMeans(n_clusters=3).fit(data).labels_

In [5]: plt.scatter(petal_length, petal_width, c=clusters)

Out[5]: <matplotlib.collections.PathCollection at 0x124e294e0>
```

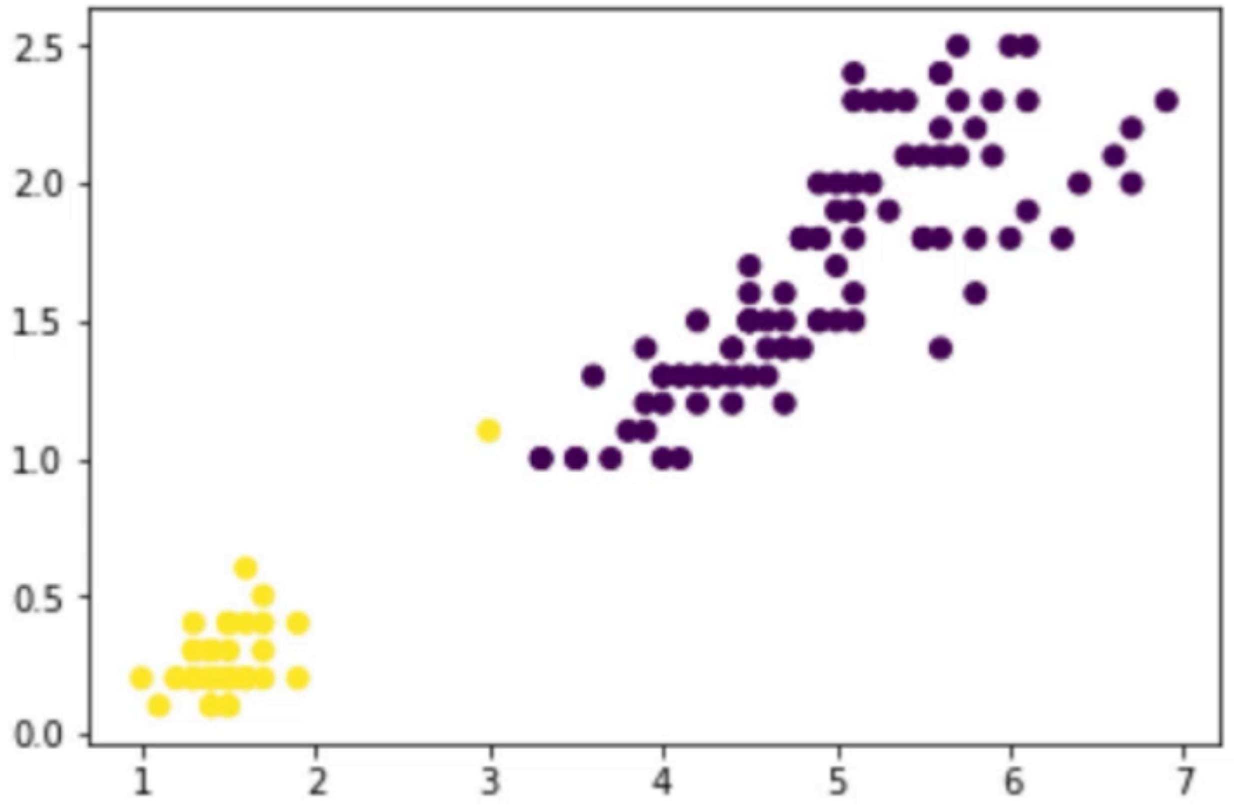



```
In [ ]: |
```

1. Incremental execution
2. In-situ output
3. Incremental changes

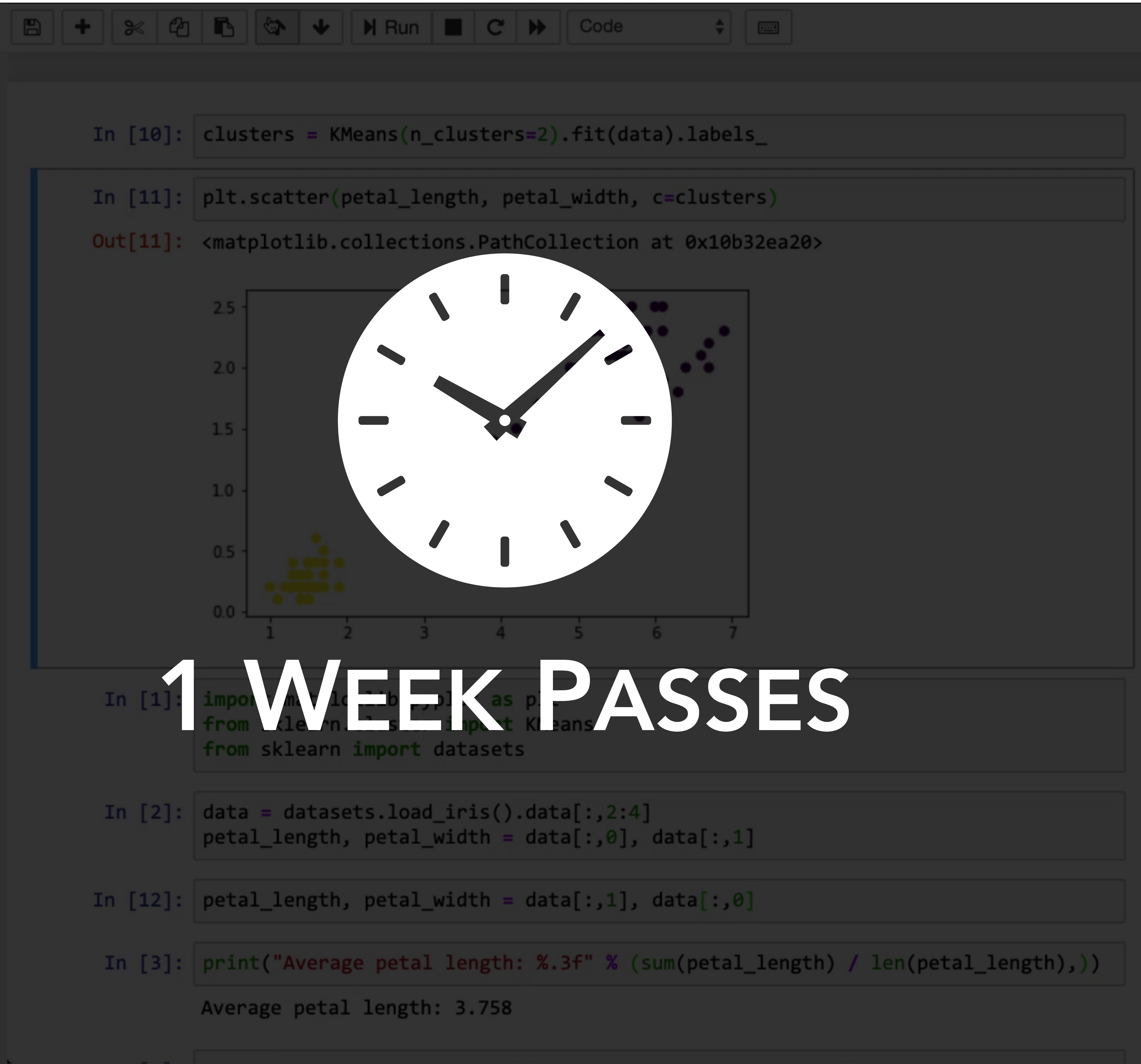
Notebook Model of Exploratory Programming

```
In [1]: import matplotlib.pyplot as plt
        from sklearn.cluster import KMeans
        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

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 0x10b32ea20>
```



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

Notebook Model of Exploratory Programming



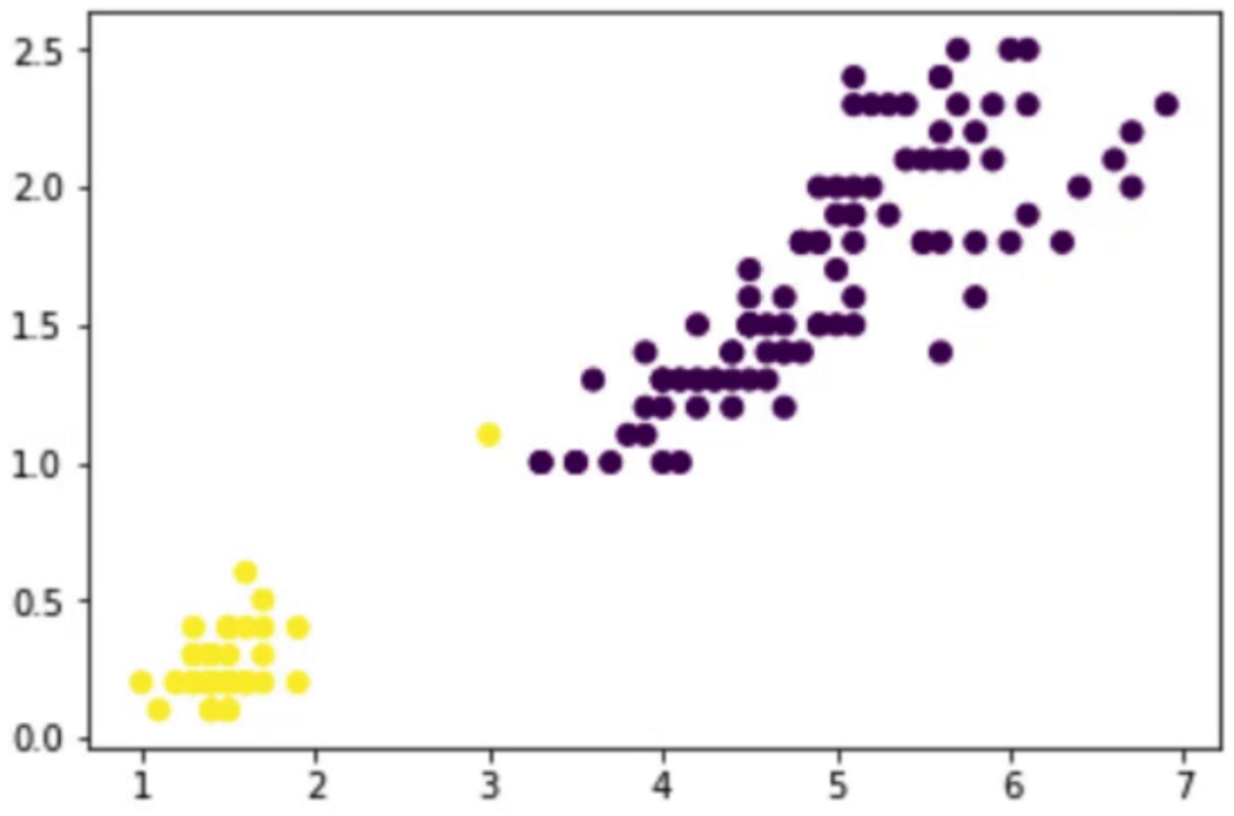
1 WEEK PASSES

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

Notebook Model of Exploratory Programming

Run Code

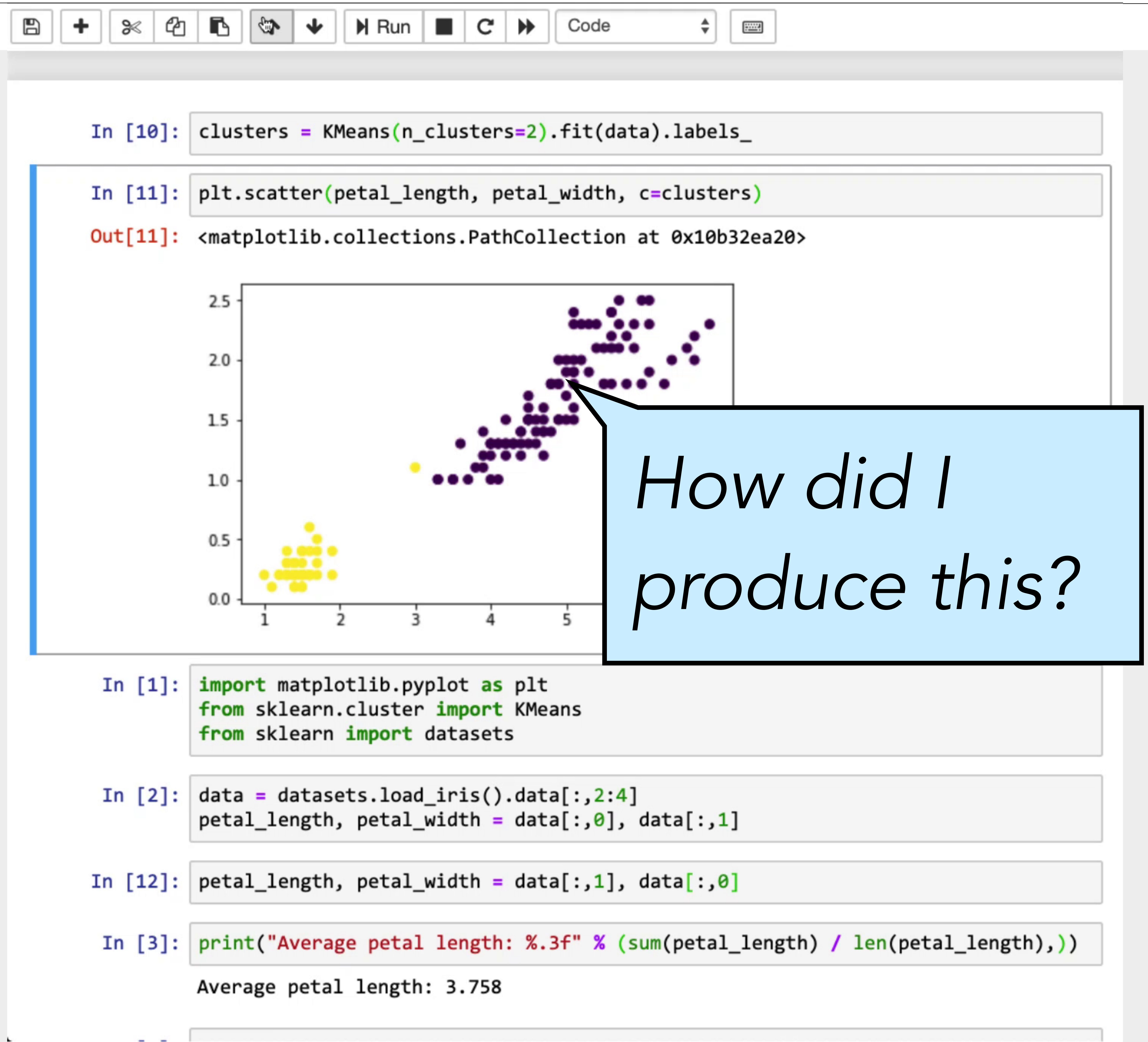
```
In [10]: clusters = KMeans(n_clusters=2).fit(data).labels_

In [11]: plt.scatter(petal_length, petal_width, c=clusters)
Out[11]: <matplotlib.collections.PathCollection at 0x10b32ea20>
```




```
In [1]: import matplotlib.pyplot as plt
        from sklearn.cluster import KMeans
        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
```

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

Notebook Model of Exploratory Programming

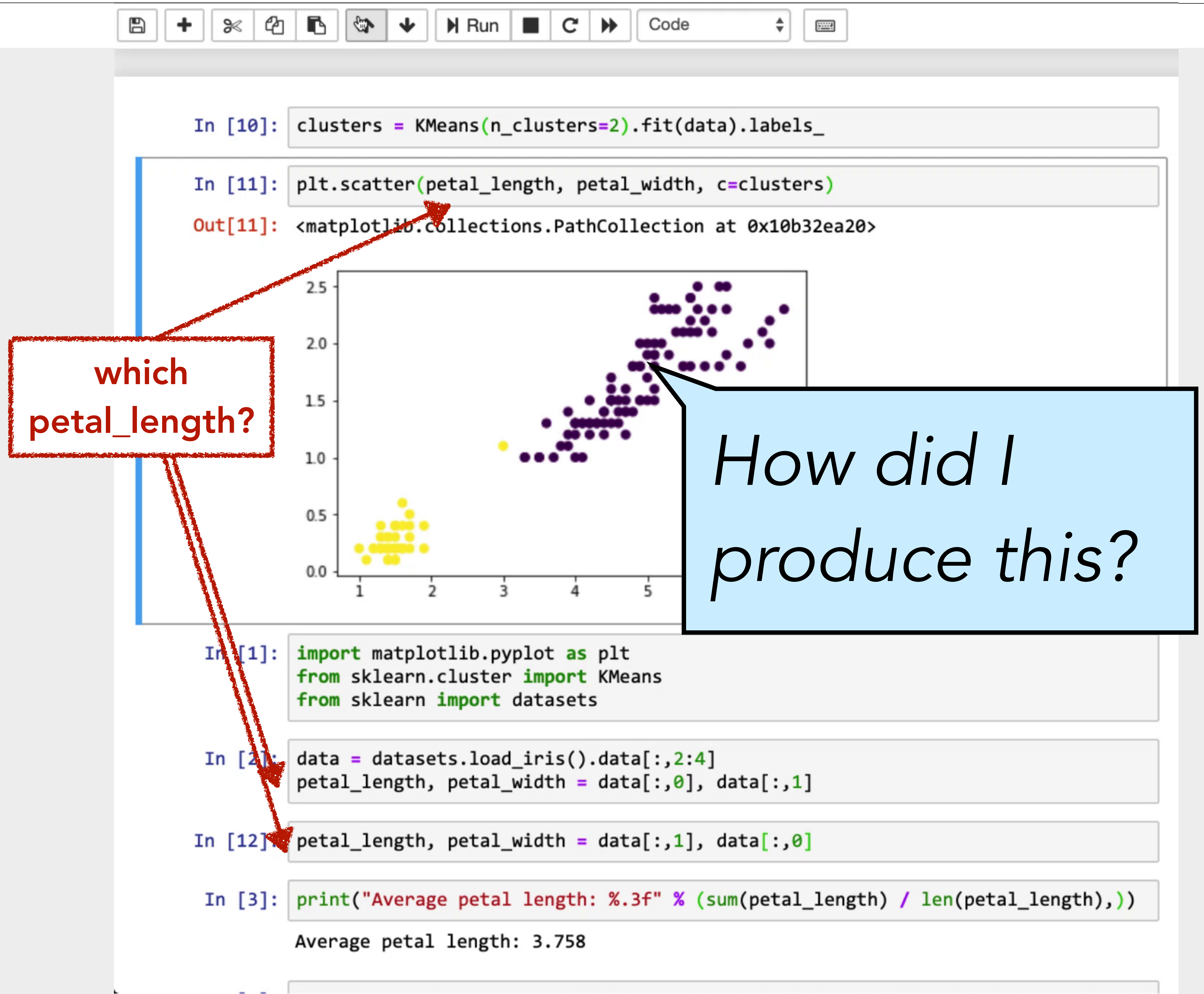


1. Incremental execution
2. In-situ output
3. Incremental changes
4. Control over layout

1 Week Later

1. How did I produce this result?

Notebook Model of Exploratory Programming

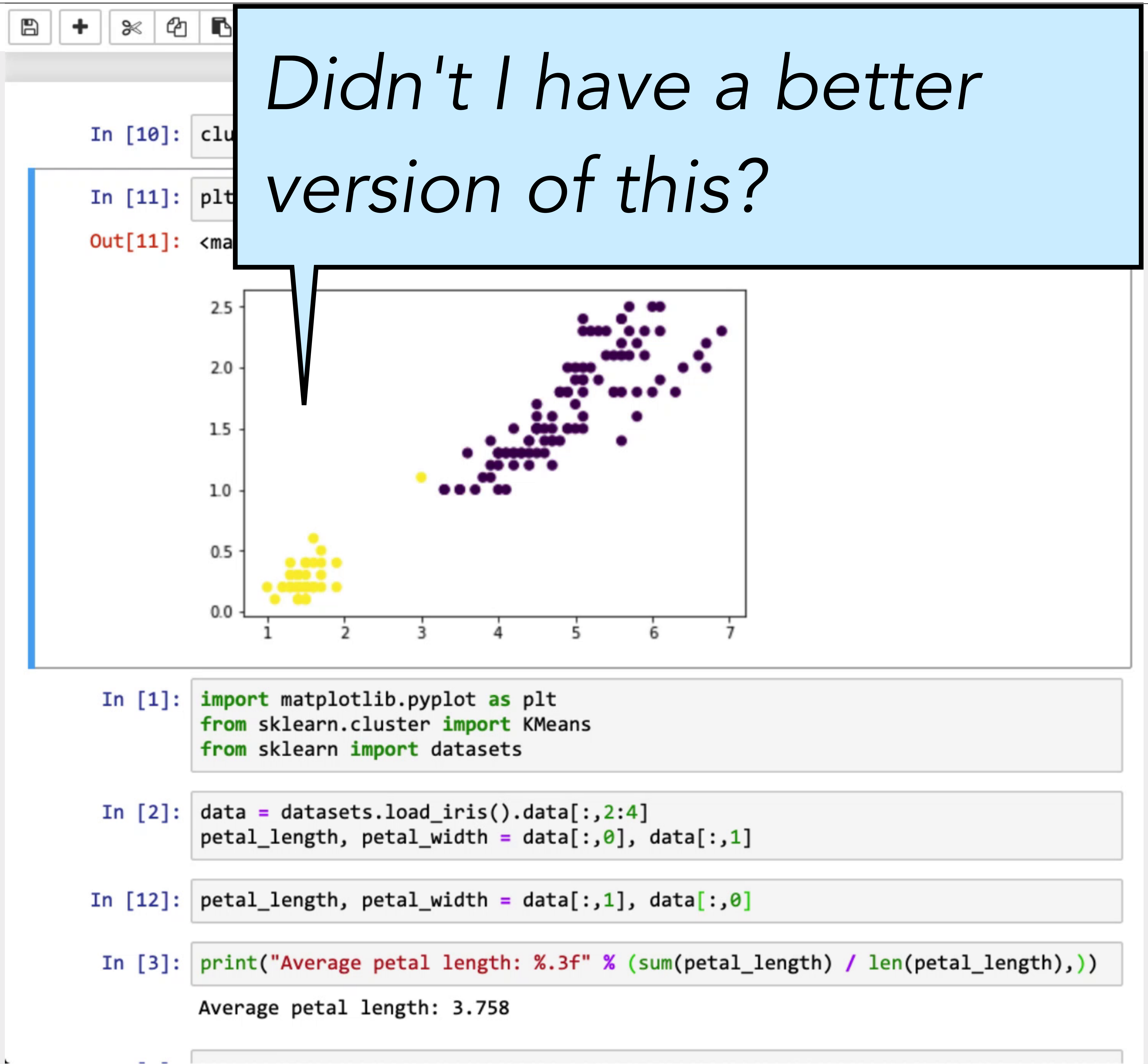


1. Incremental execution
2. In-situ output
3. Incremental changes
4. Control over layout

1 Week Later

1. How did I produce this result?

Notebook Model of Exploratory Programming

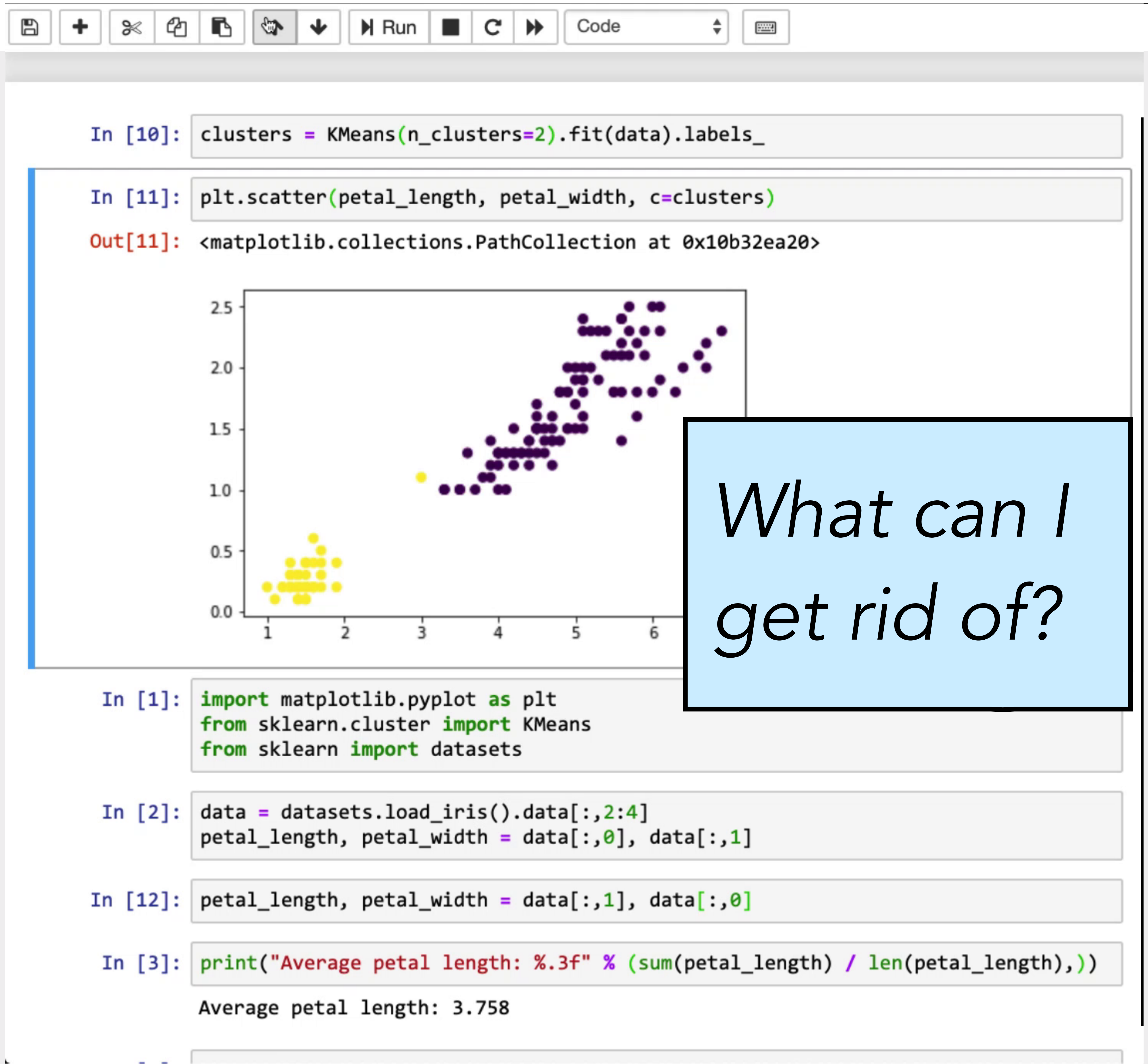


1. Incremental execution
2. In-situ output
3. Incremental changes
4. Control over layout

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

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?

Messes in Computational Notebooks

Disappearance

Deleted / overwritten code

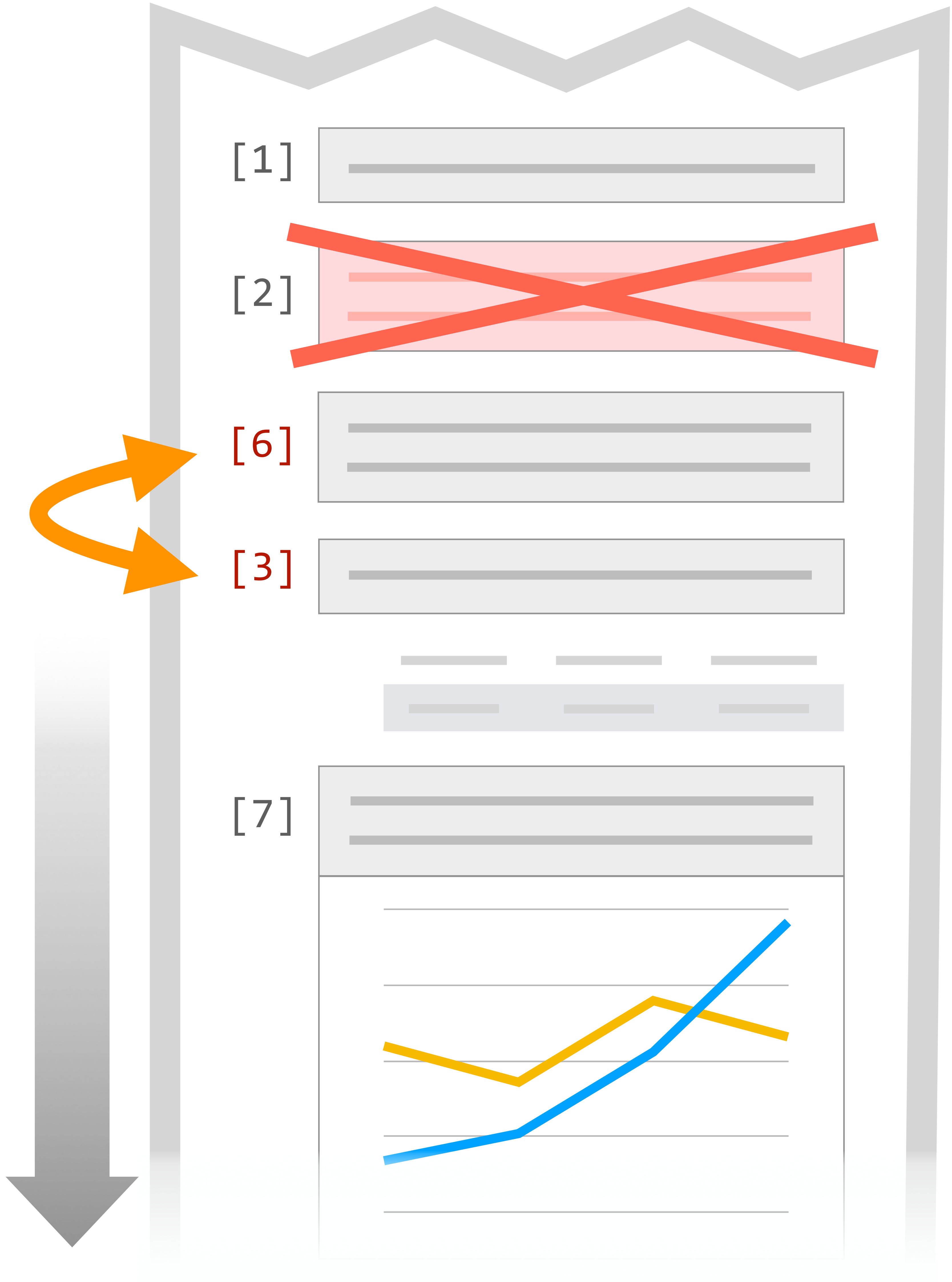
Disorder

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]

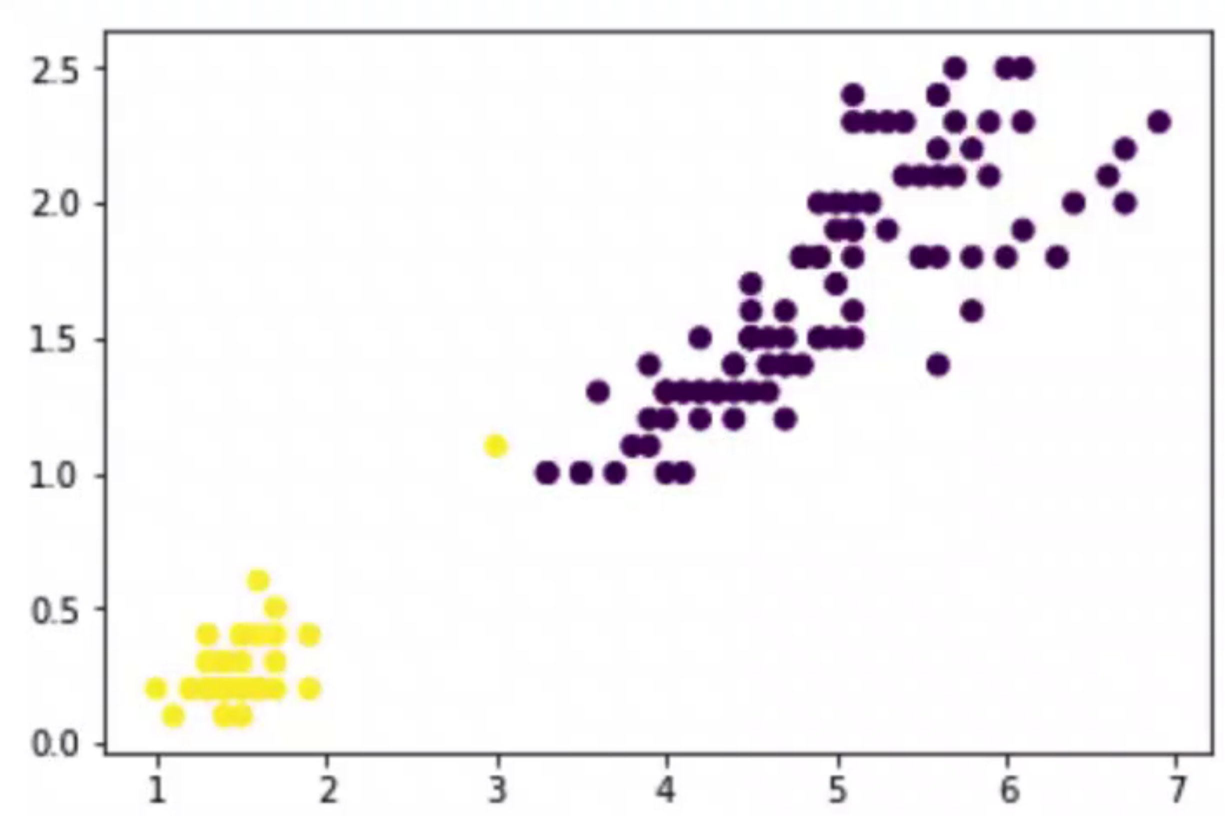
31 / 41 surveyed participants had trouble finding prior analyses [Kery et al. 2018]

Code Gathering Tools Demo

Code Gathering Tools Demo

Gather to: Clipboard | Notebook | Revisions | Clear

```
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 0x108eacba8>
```

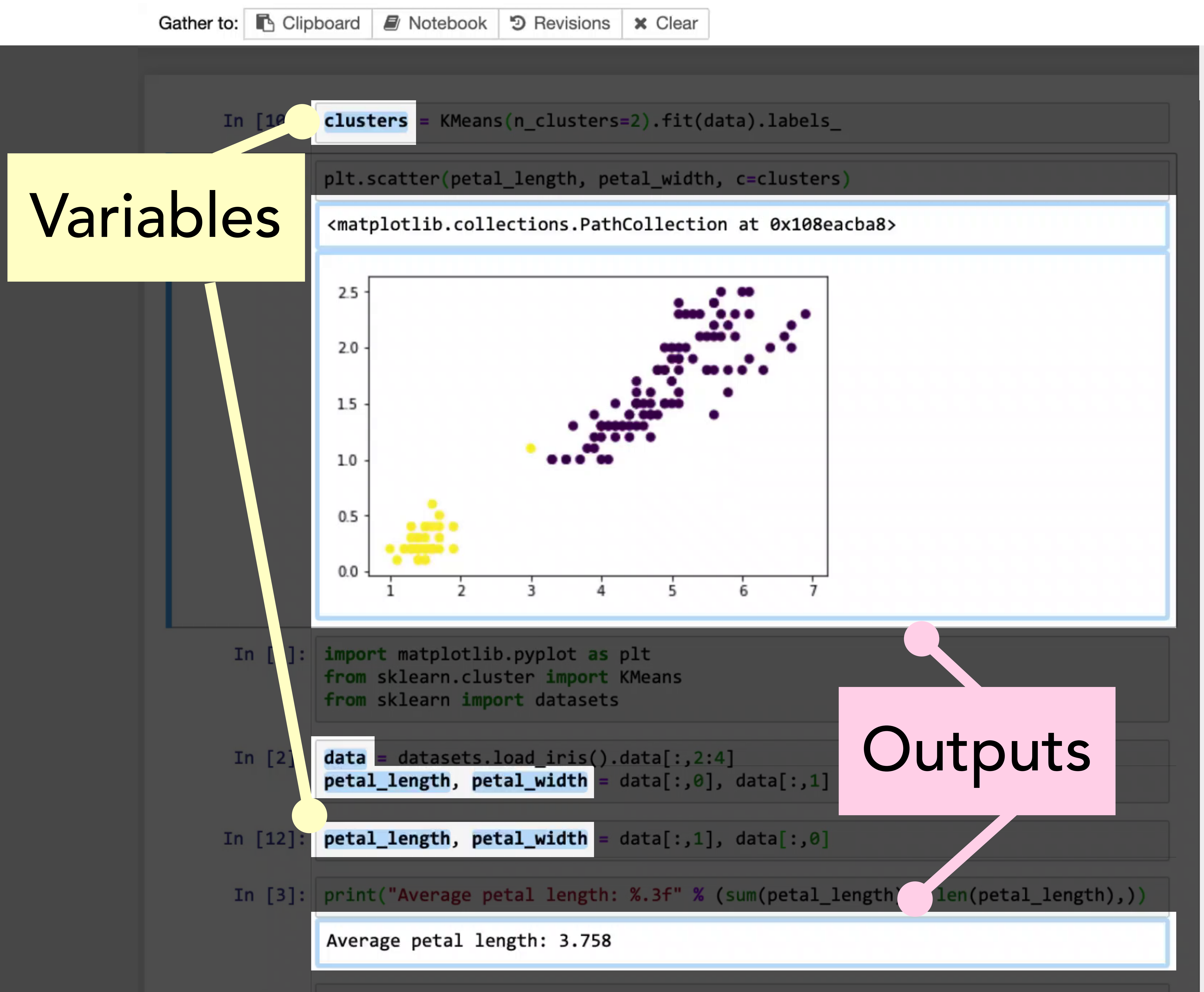



```
In [1]: import matplotlib.pyplot as plt
        from sklearn.cluster import KMeans
        from sklearn import datasets

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?

Code Gathering Tools Demo



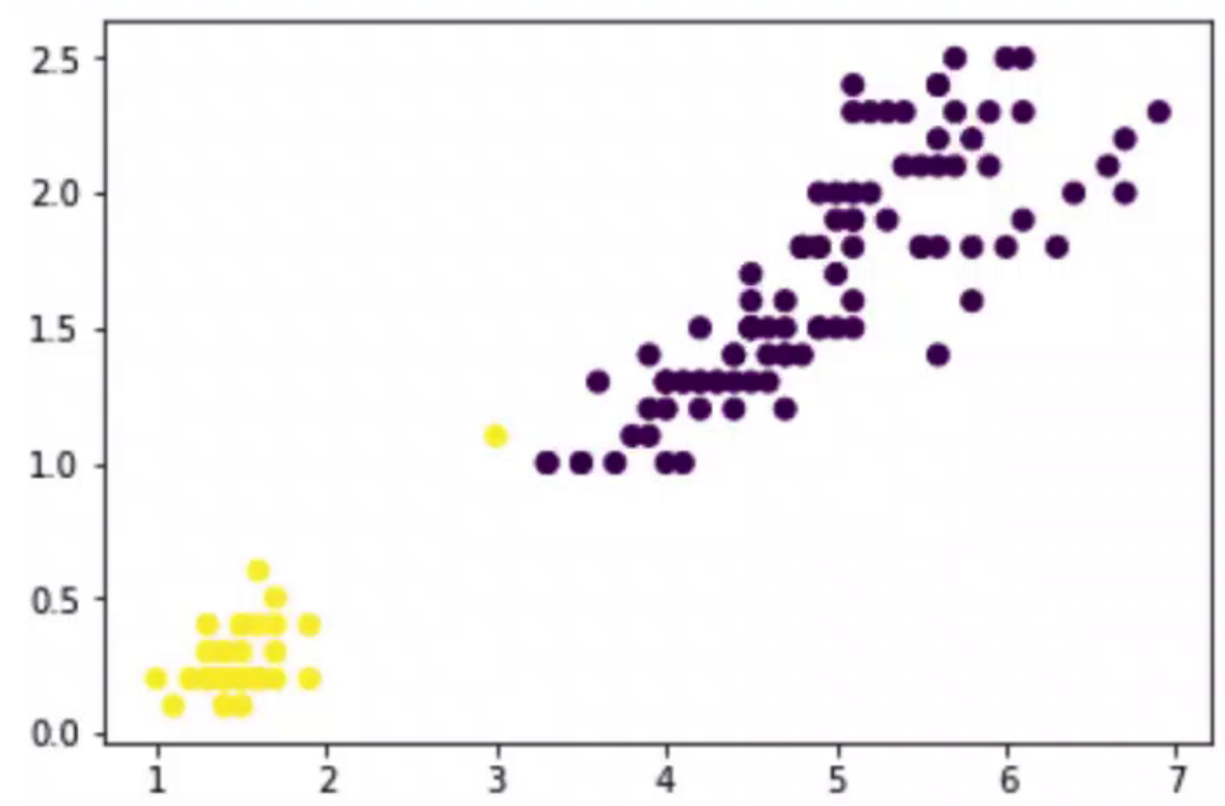
Task 1: Recovering Code

How did I produce this?

Code Gathering Tools Demo

Gather to: Clipboard | Notebook | Revisions | Clear

```
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 0x108eacba8>
```




```
In [1]: import matplotlib.pyplot as plt
        from sklearn.cluster import KMeans
        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
```

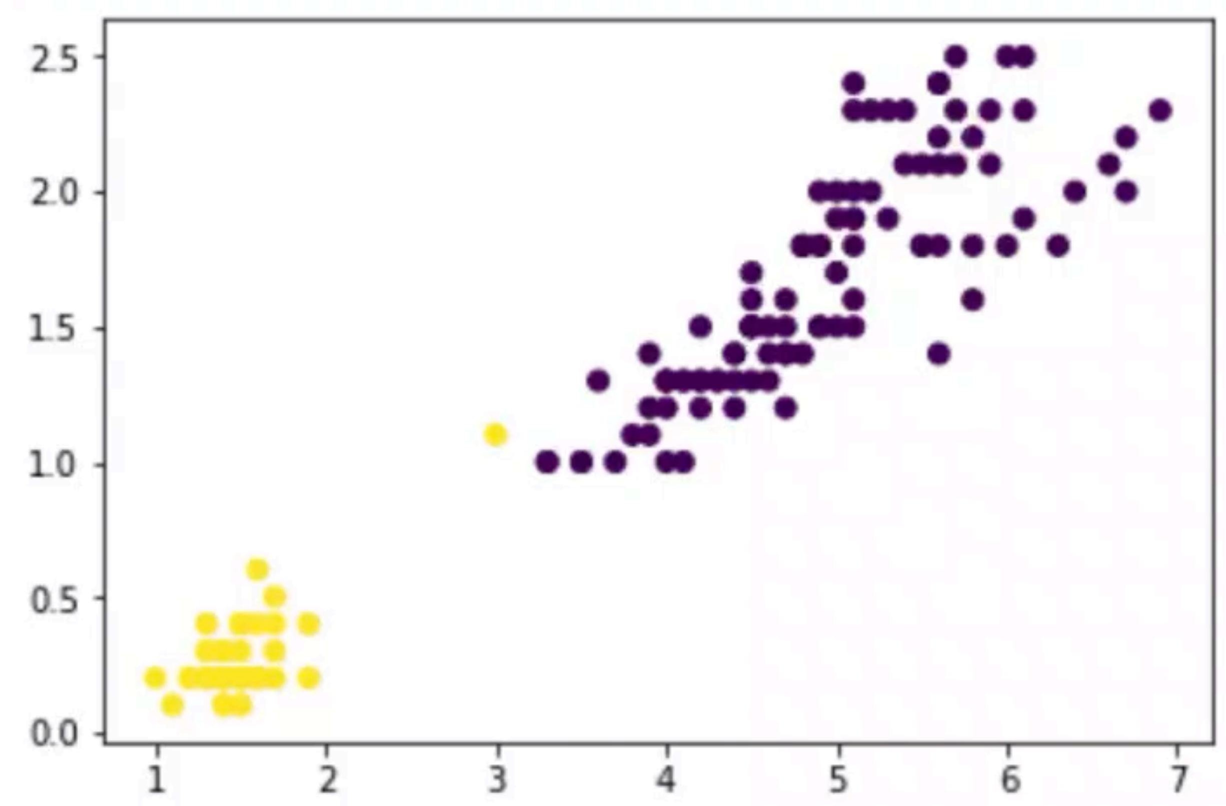
Task 1: Recovering Code

How did I produce this?

Code Gathering Tools Demo

Gather to: Clipboard | Notebook | Revisions | Clear

```
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 0x108eacba8>
```




```
In [1]: import matplotlib.pyplot as plt
        from sklearn.cluster import KMeans
        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

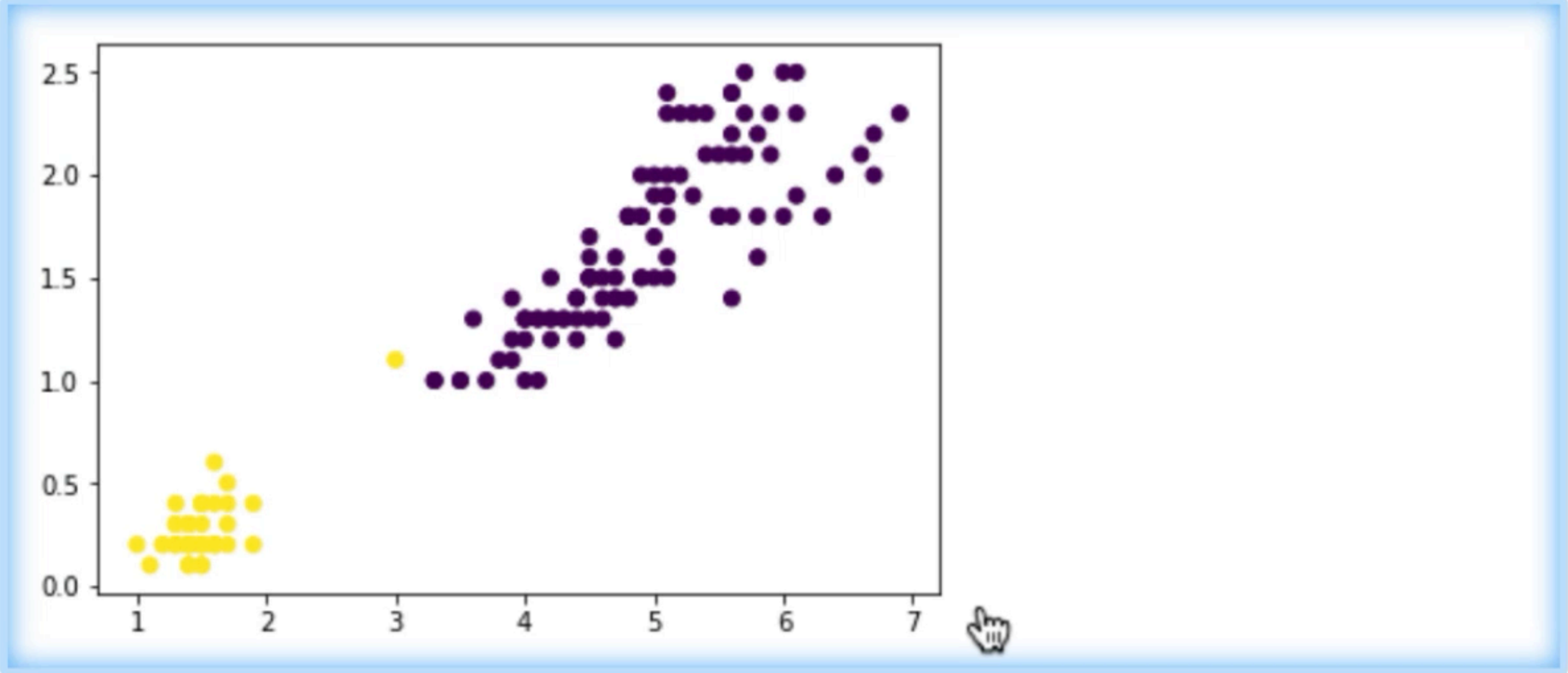
How did I produce this?

Request cell subset that produced the result.

Code Gathering Tools Demo

Gather to: Clipboard Notebook Revisions ✖ Clear

```
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 0x108eacba8>
```




```
In [1]: import matplotlib.pyplot as plt
        from sklearn.cluster import KMeans
        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
```

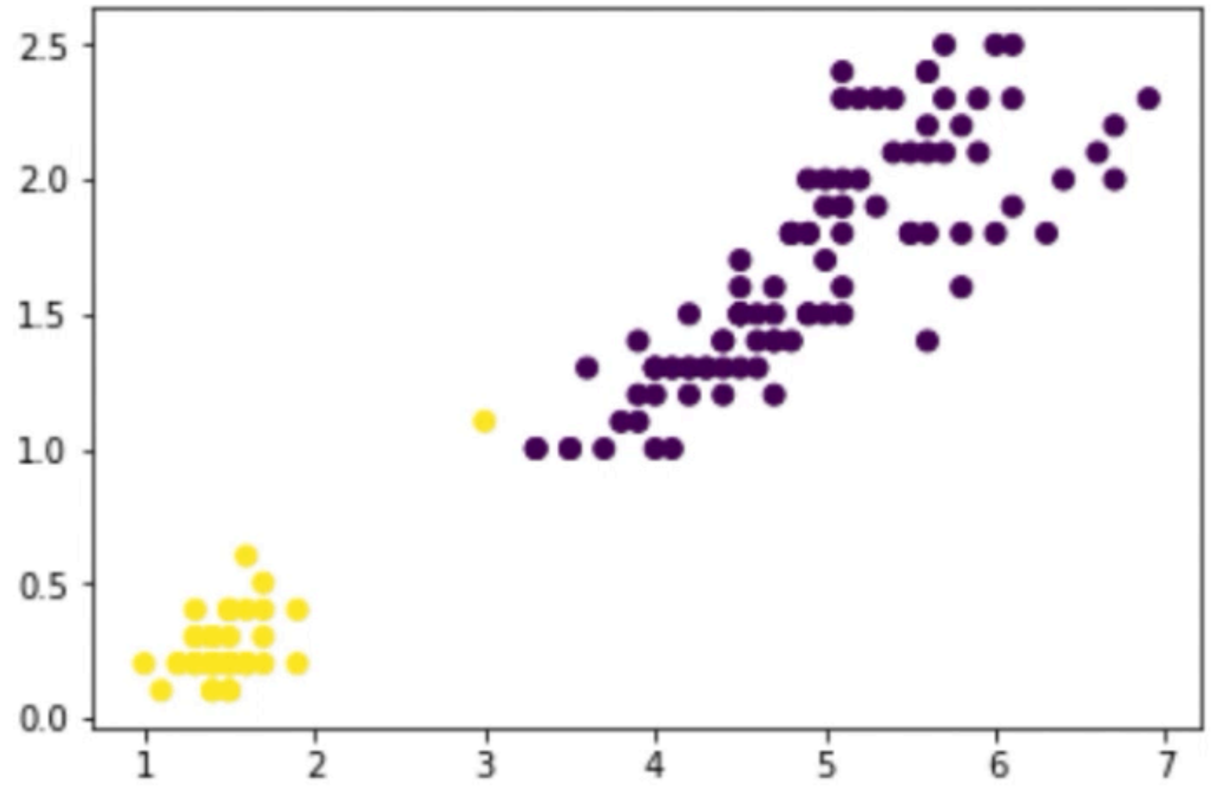
Task 1: Recovering Code

How did I produce this?

Request cell subset that produced the result.

Code Gathering Tools Demo

```
In [ ]: import matplotlib.pyplot as plt
        from sklearn.cluster import KMeans
        from sklearn import datasets

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

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

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



The gathered code is...

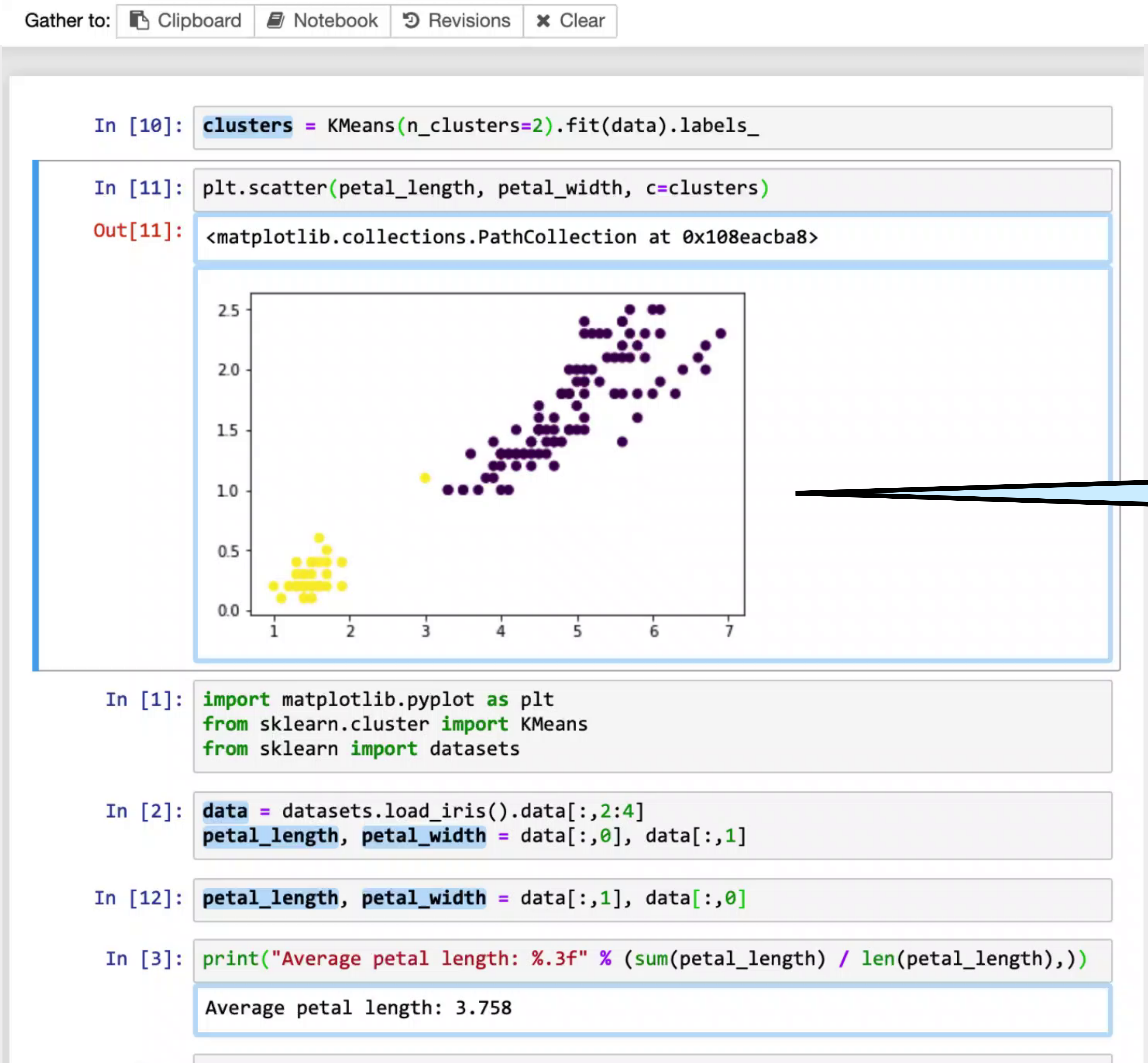
- reduced
- ordered
- complete

Task 1: Recovering Code

How did I produce this?

Request cell subset that produced the result.

Code Gathering Tools Demo



Task 1: Recovering Code

Request cell subset that produced the result.

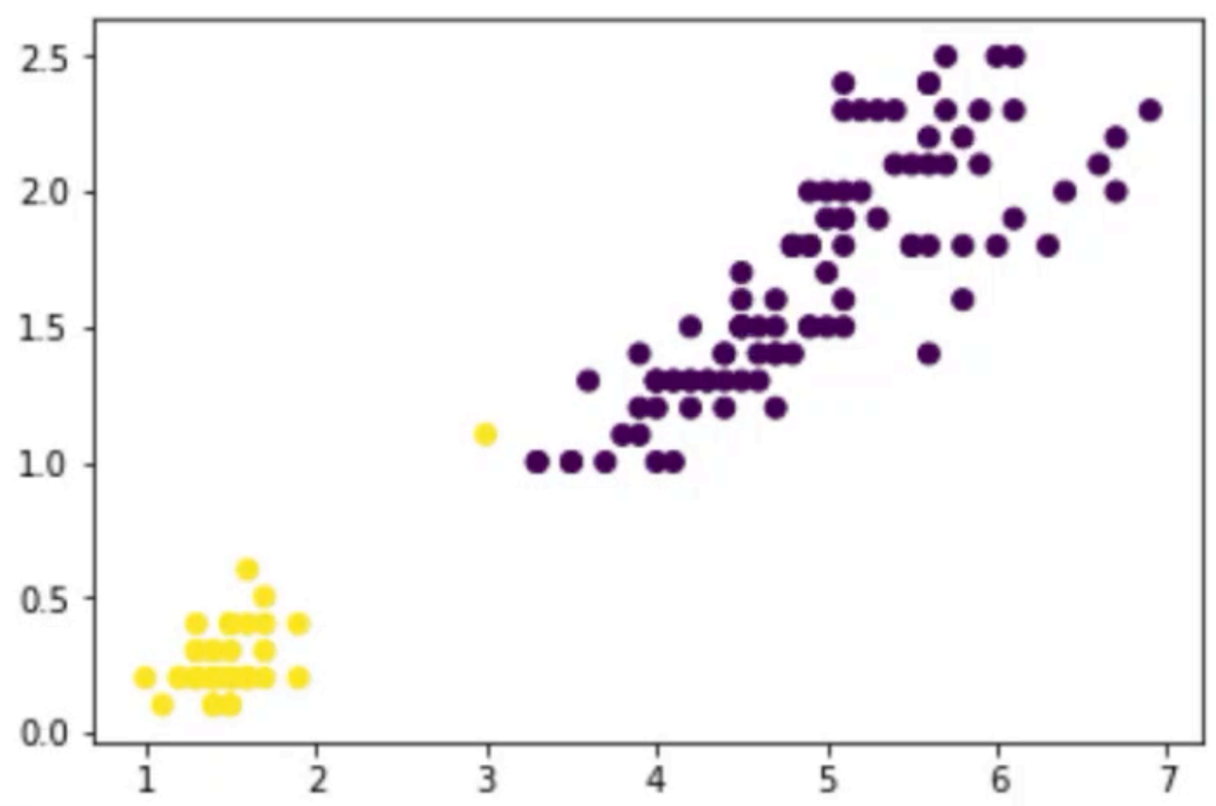
Task 2: Comparing Versions

Didn't I have a better version of this?

Code Gathering Tools Demo

Gather to: Clipboard | Notebook | Revisions | Clear

```
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 0x108eacba8>
```



```
In [1]: import matplotlib.pyplot as plt
        from sklearn.cluster import KMeans
        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

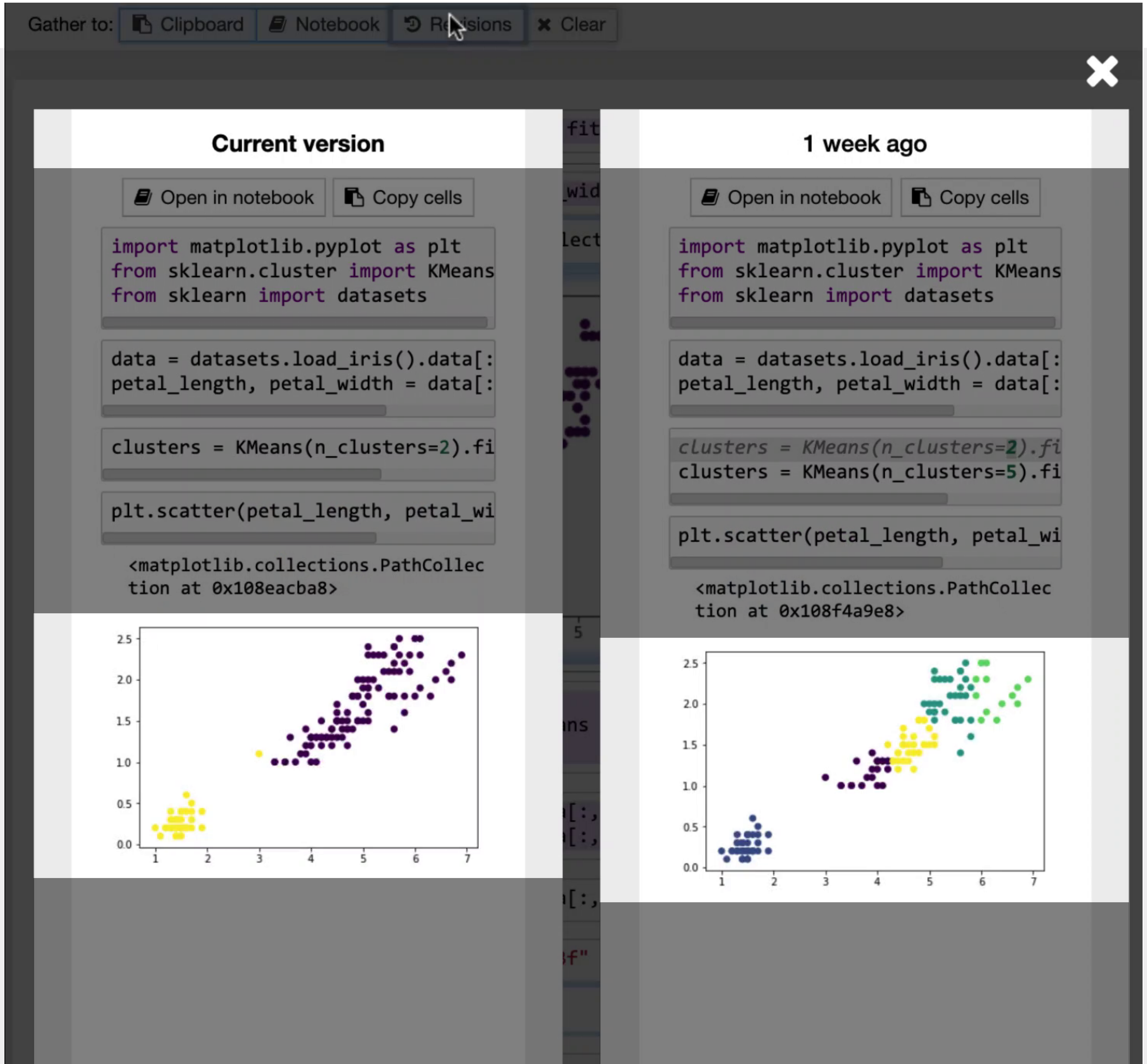
Request cell subset that produced the result.

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

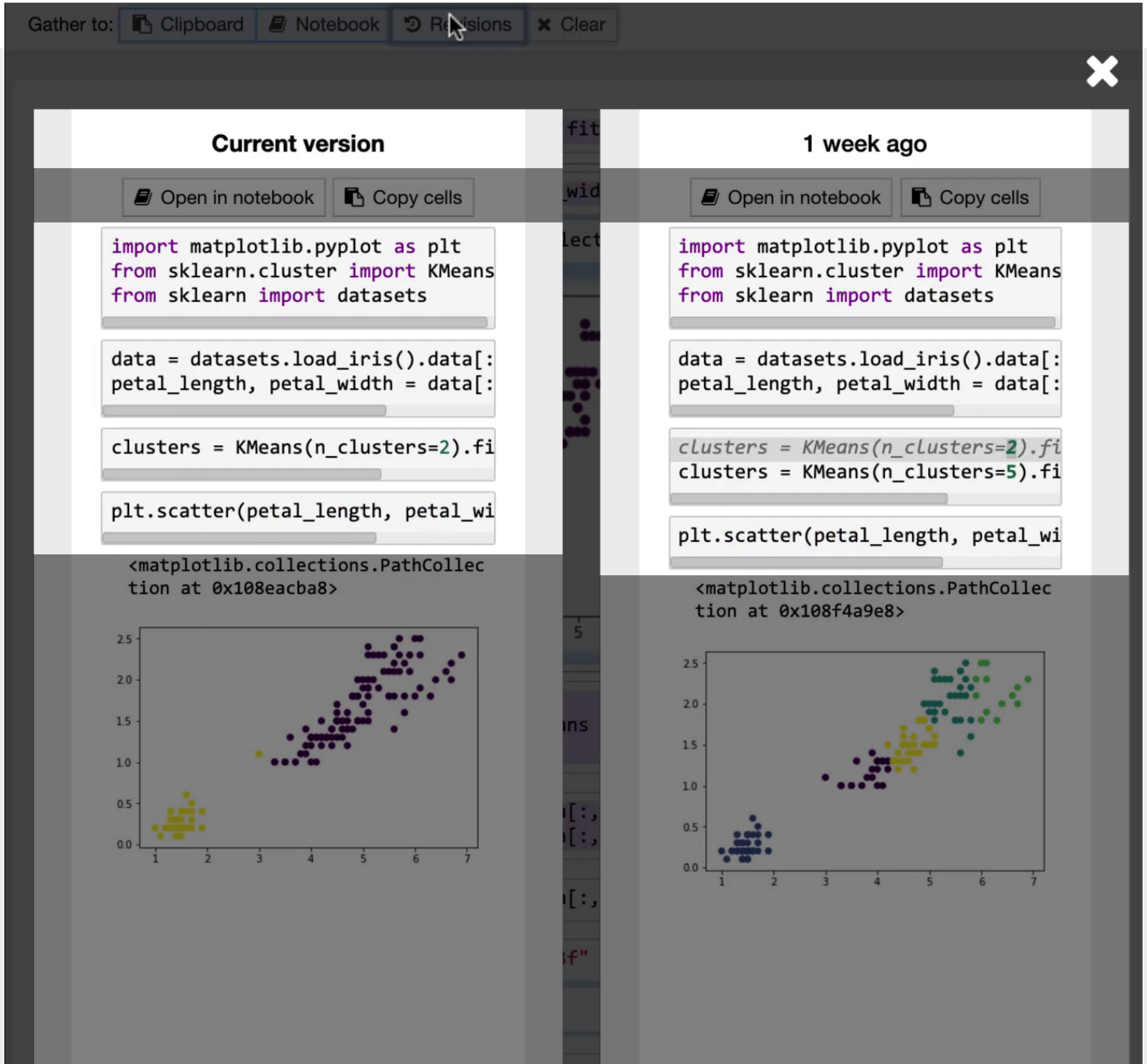
Request cell subset that produced the result.

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

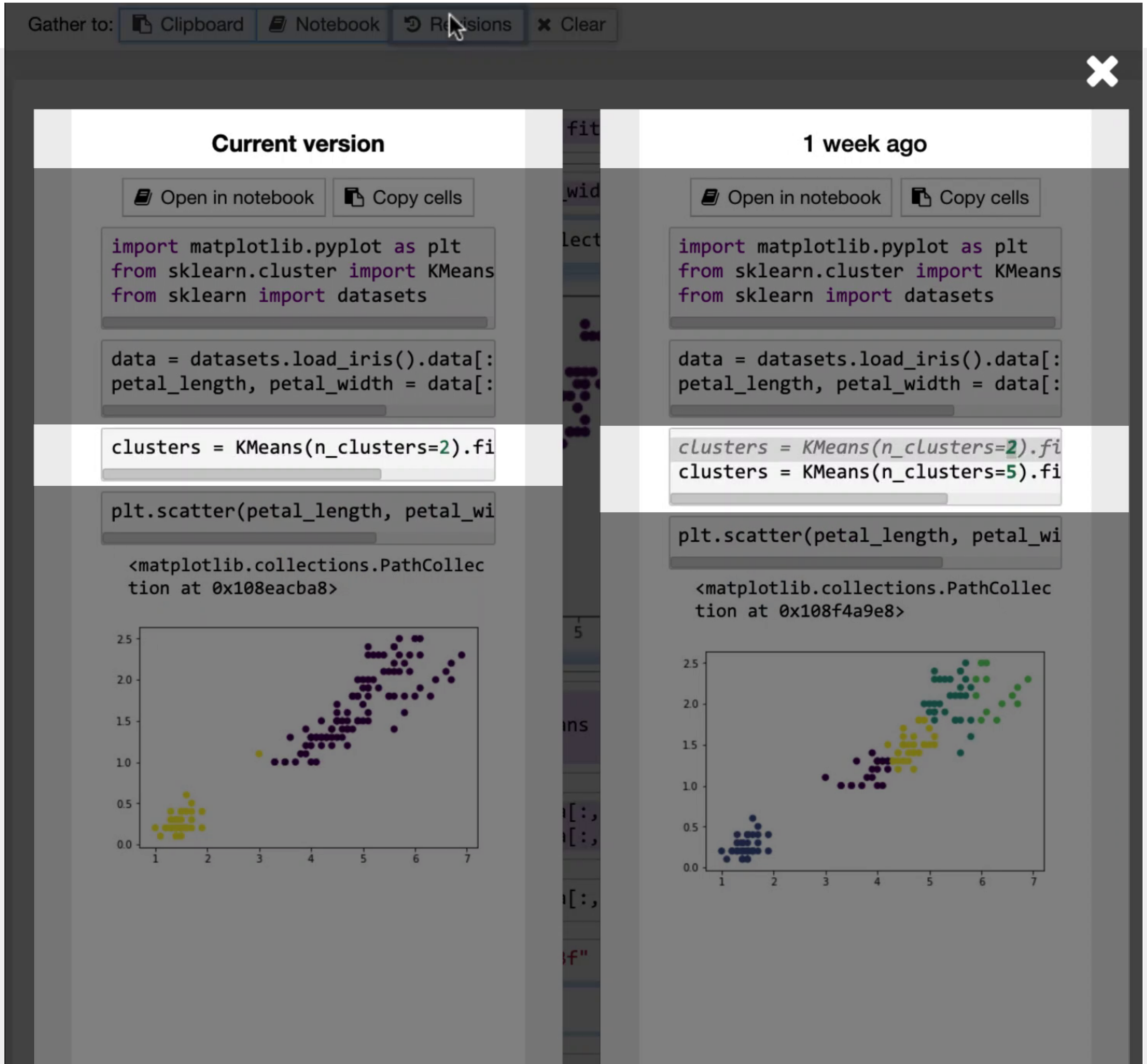
Request cell subset that produced the result.

Task 2: Comparing Versions

Didn't I have a better version of this?

Open a version browser for a result.

Code Gathering Tools Demo



Task 1: Recovering Code

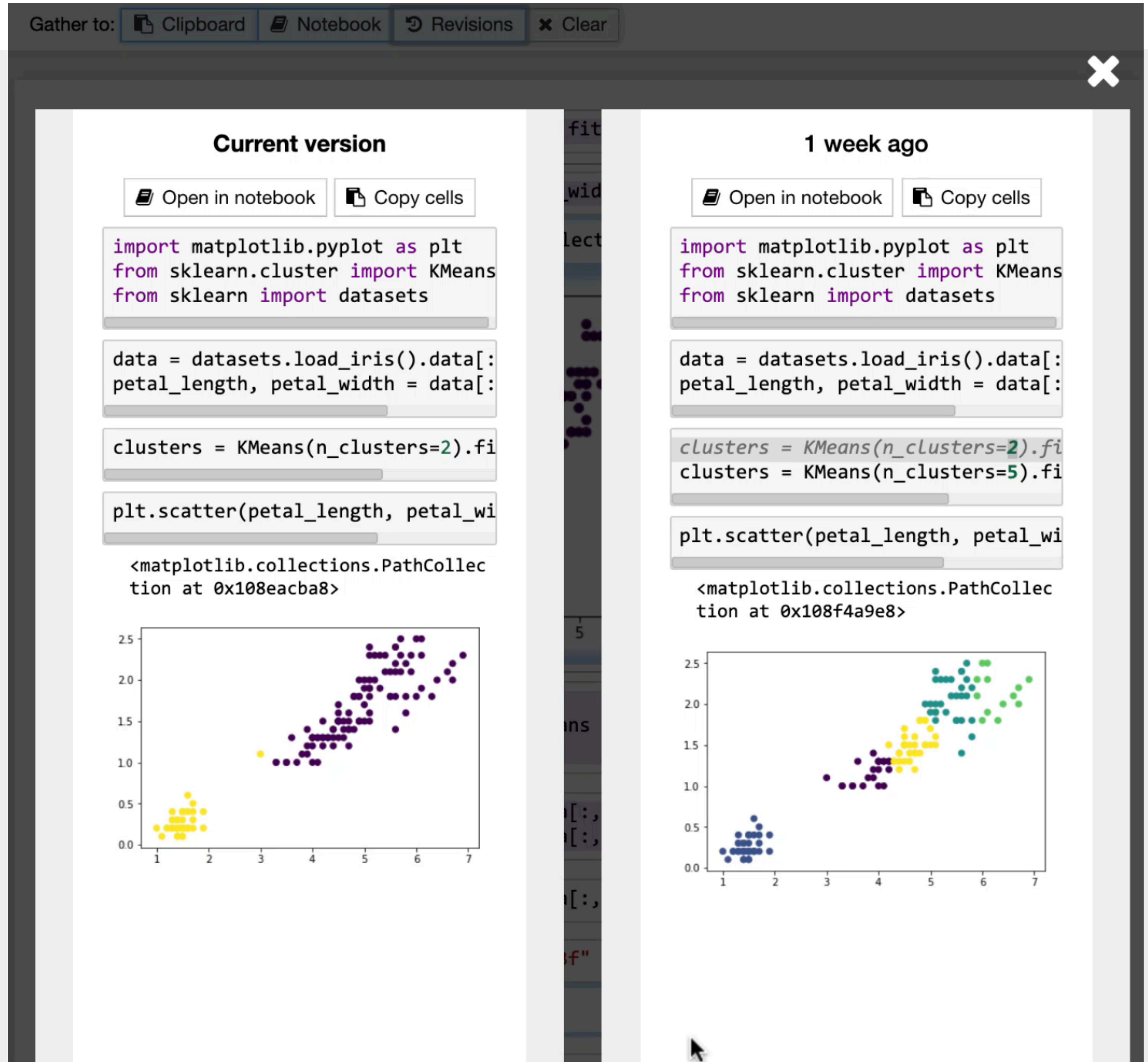
Request cell subset that produced the result.

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

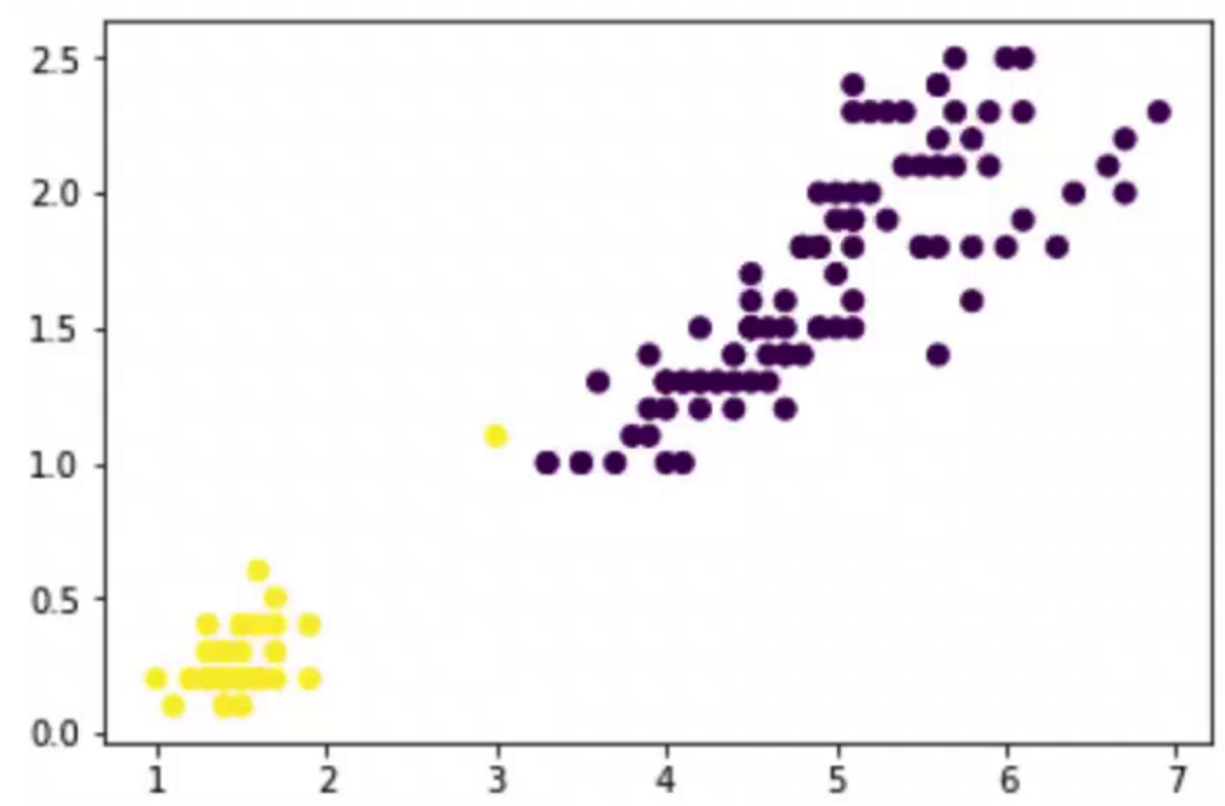
Didn't I have a better version of this?

Open a version browser for a result.

Code Gathering Tools Demo

Gather to: Clipboard | Notebook | Revisions | Clear

```
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 0x108eacba8>
```




```
In [1]: import matplotlib.pyplot as plt
        from sklearn.cluster import KMeans
        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

Request cell subset that produced the result.

Task 2: Comparing Versions

Open a version browser for a result.

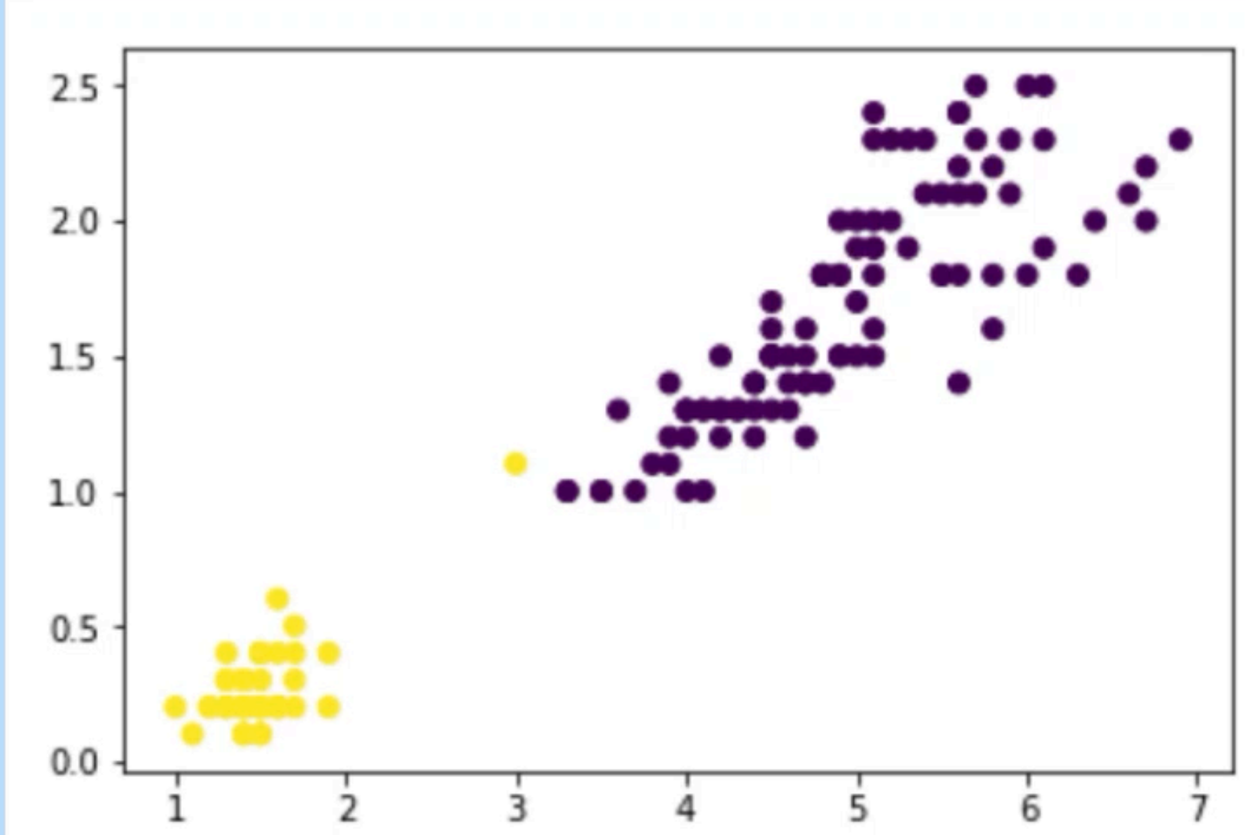
Task 3: Cleaning Notebook

What code can I get rid of?

Code Gathering Tools Demo

Gather to: Clipboard | Notebook | Revisions | Clear

```
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 0x108eacba8>
```




```
In [1]: import matplotlib.pyplot as plt
        from sklearn.cluster import KMeans
        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

Request cell subset that produced the result.

Task 2: Comparing Versions

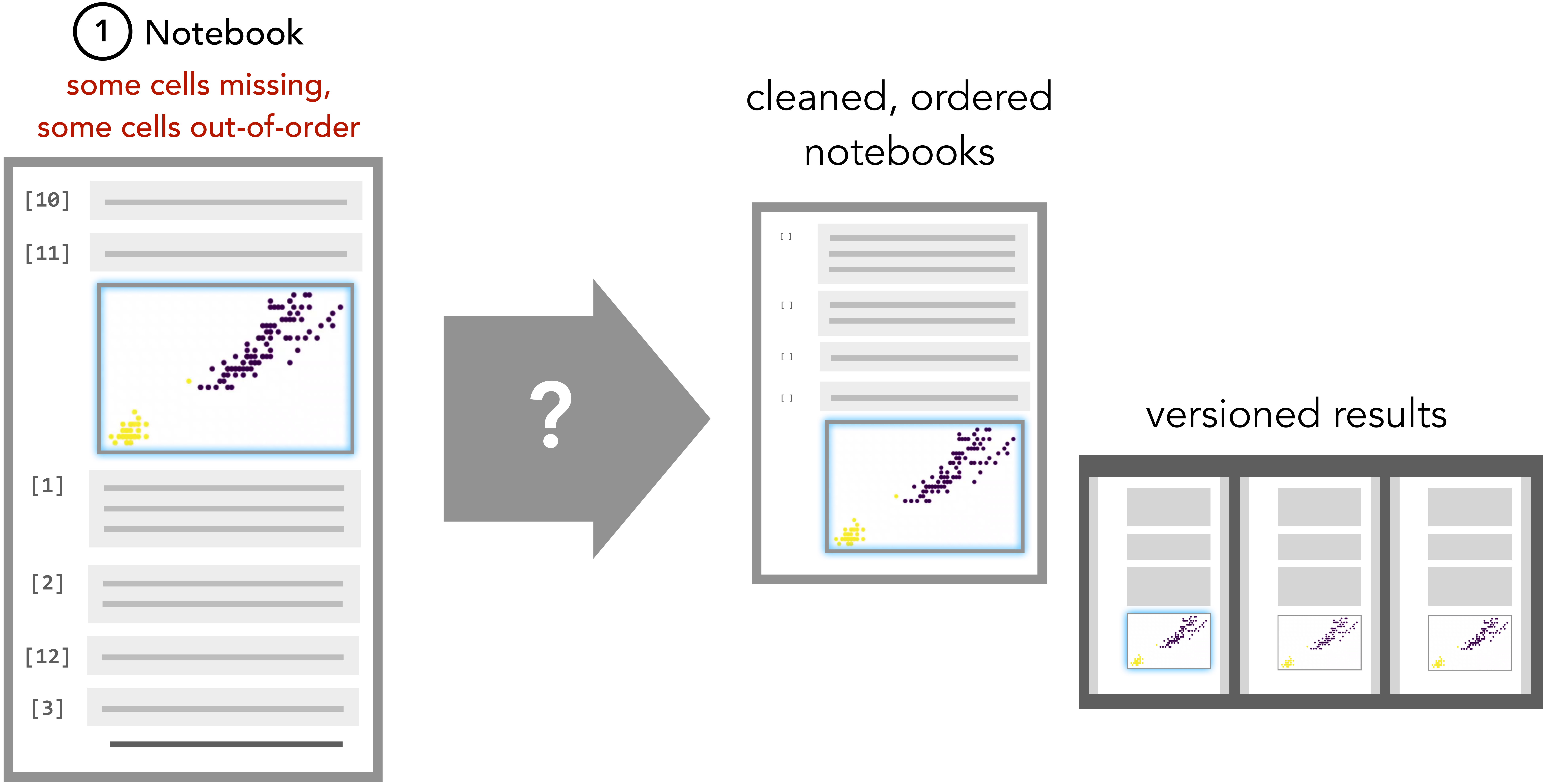
Open a version browser for a result.

Task 3: Cleaning Notebook

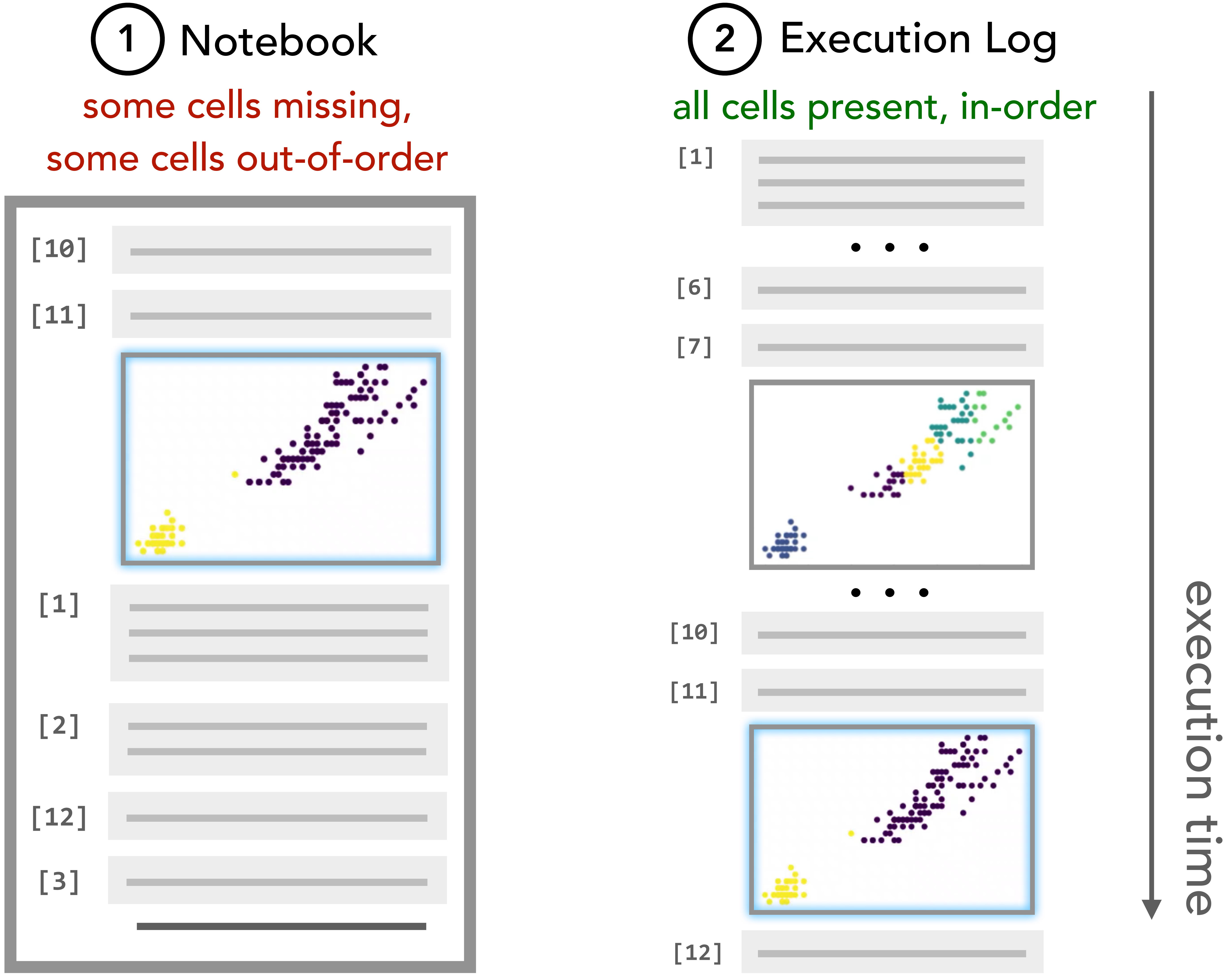
What code can I get rid of?

... Request cell subset that produced the result.

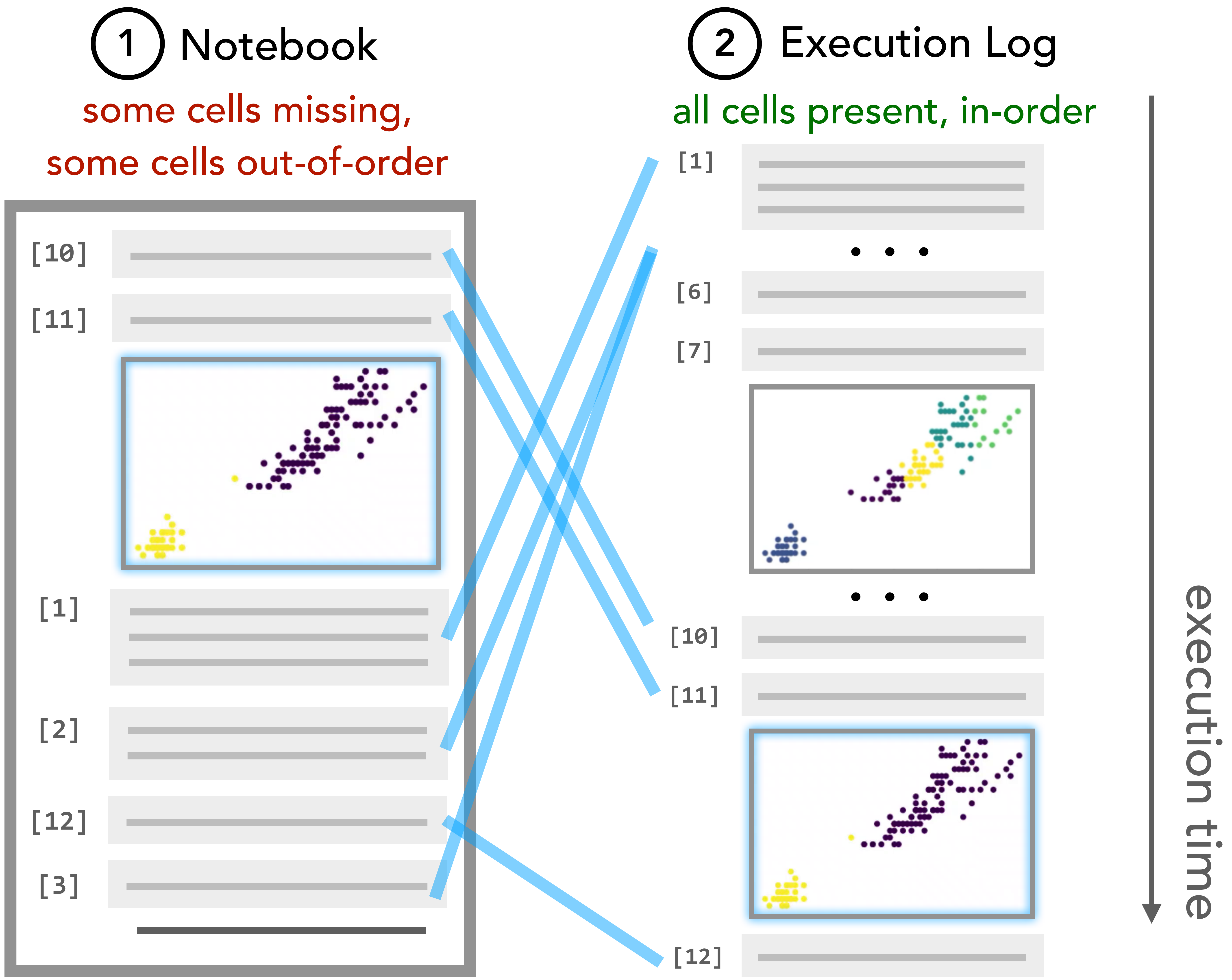
Program Analysis: Slicing Notebooks



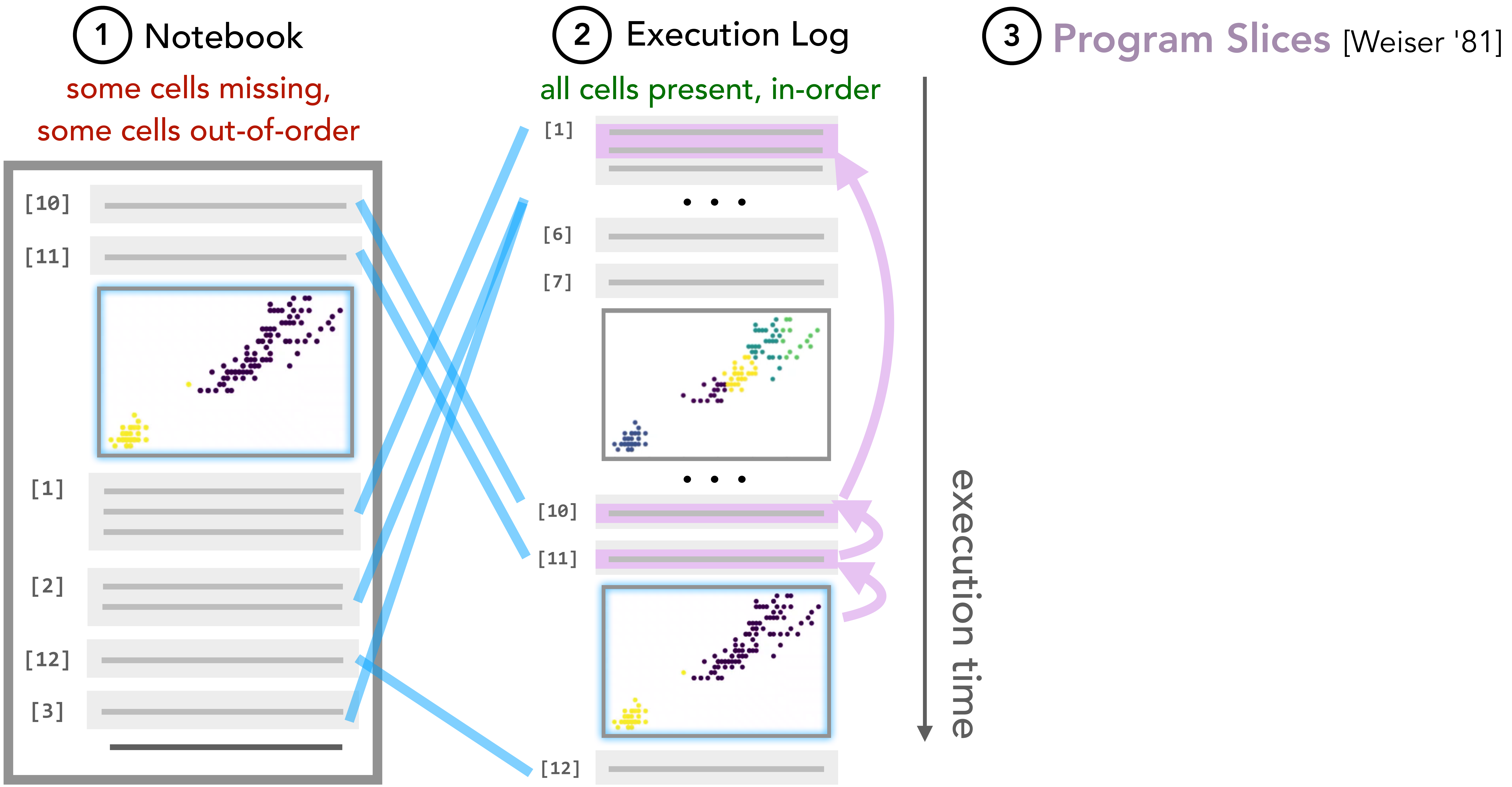
Program Analysis: Slicing Notebooks



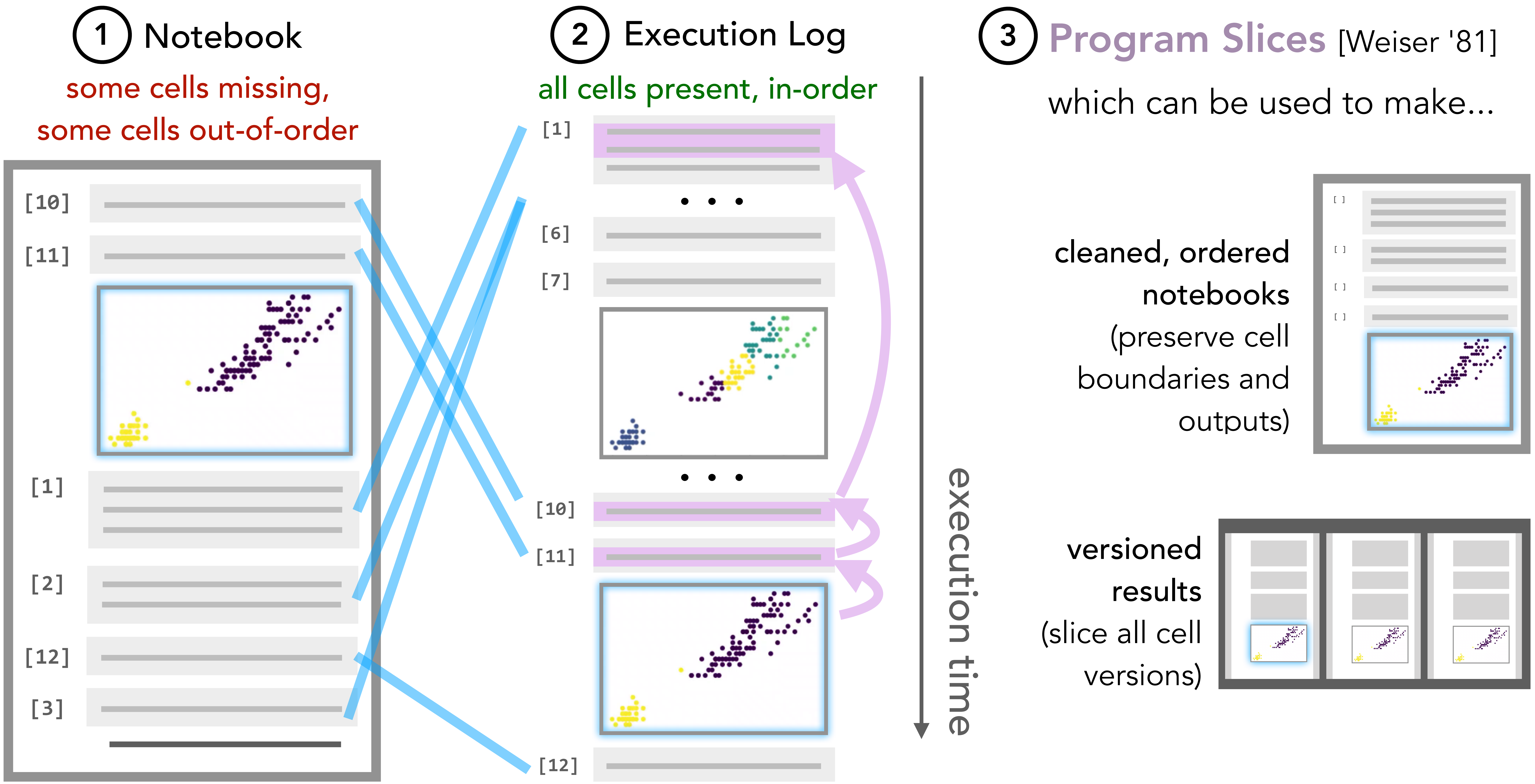
Program Analysis: Slicing Notebooks



Program Analysis: Slicing Notebooks



Program Analysis: Slicing Notebooks



Evaluating Code Gathering Tools

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

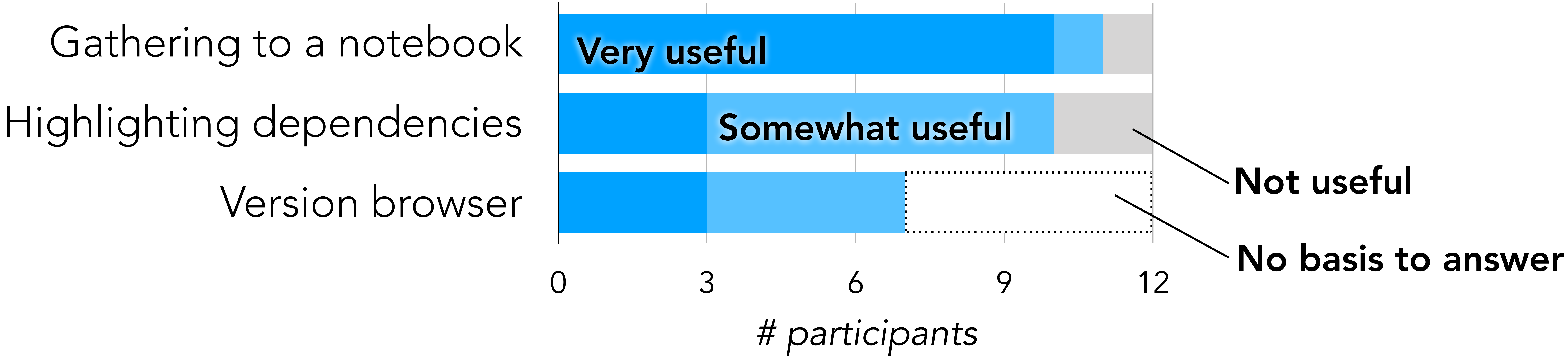
A Qualitative Study of Gathering

Participants: *N = 12* professional data analysts

Cleaning Task × 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.

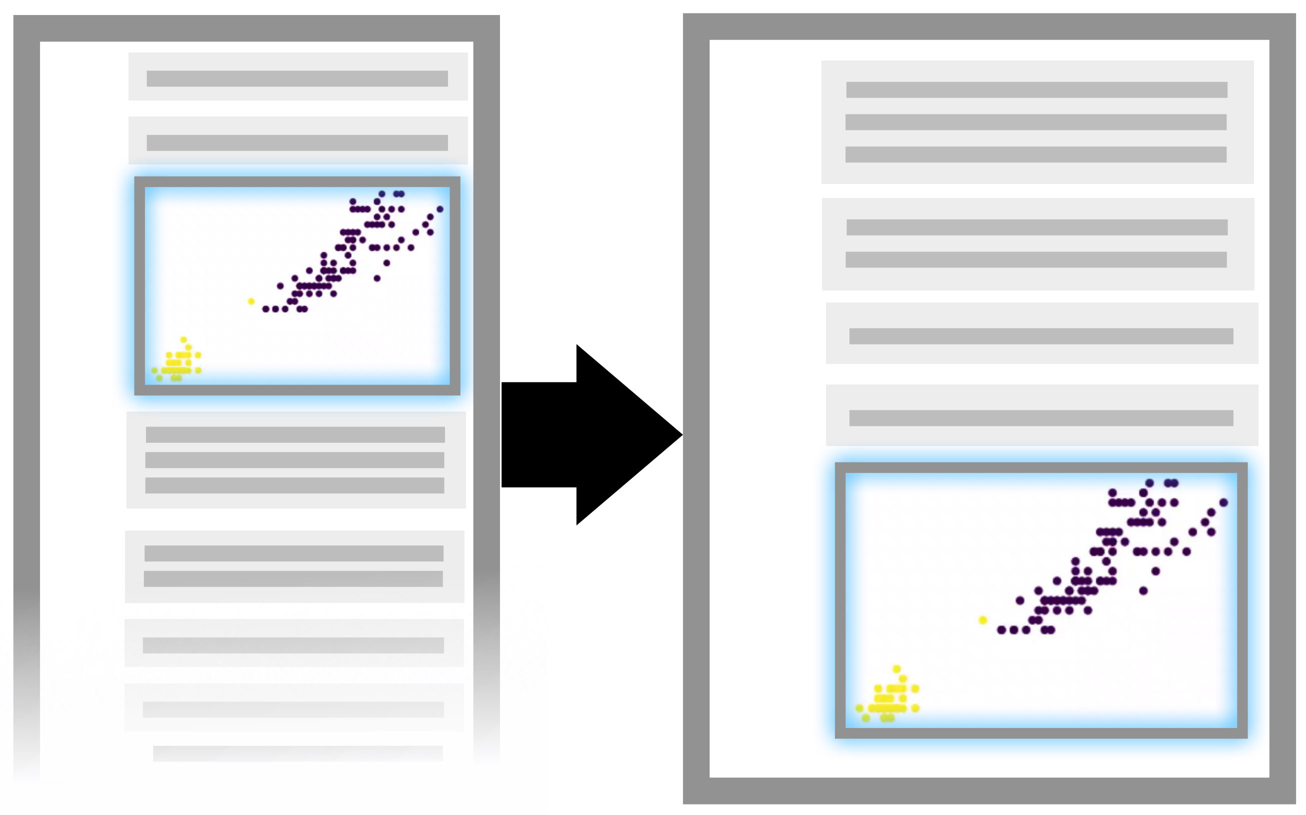
Q. How do data analysts distill code during the process of exploratory data analysis?



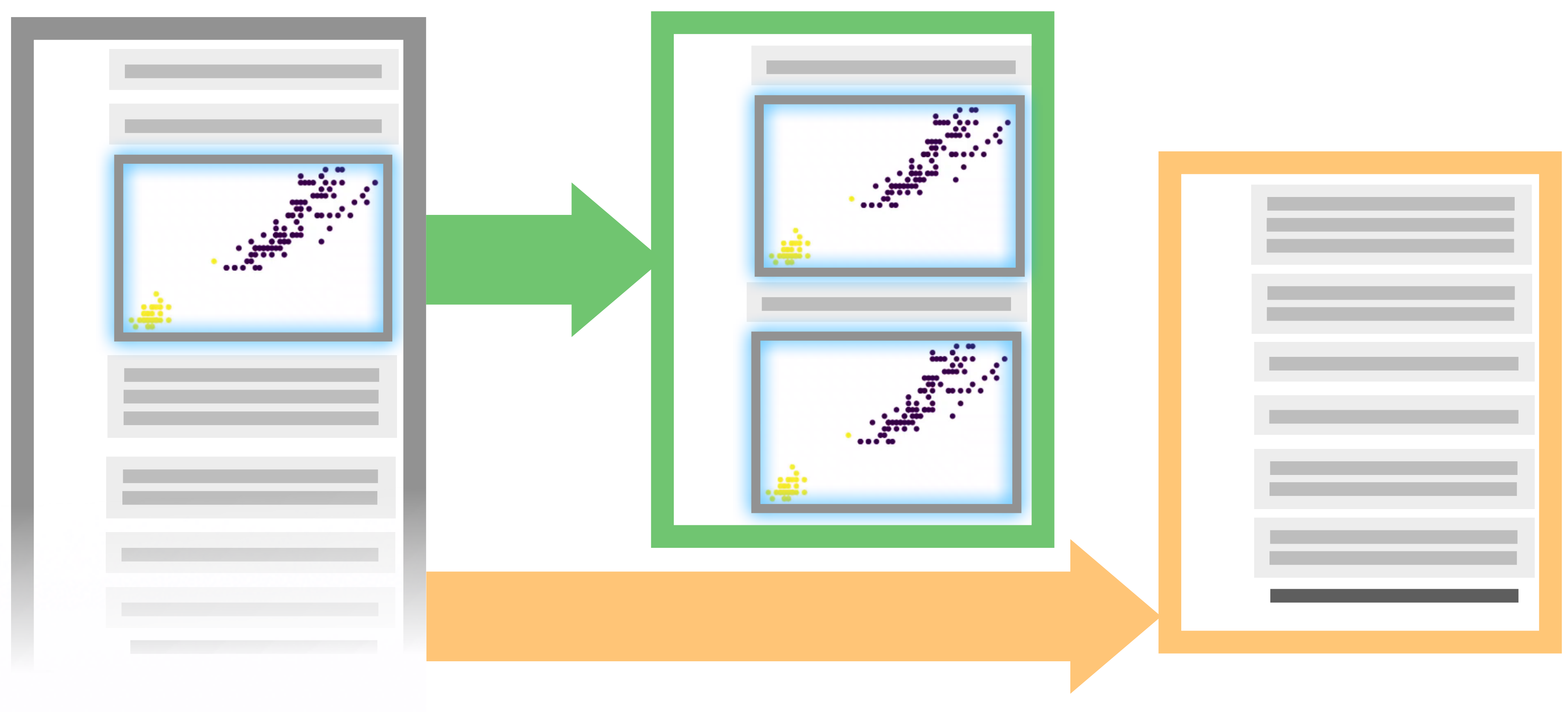
Participants described gathering to a notebook as "beautiful" and "amazing": it "hits the nail on the head."

Some Observed Uses of Gathering Tools

"Finishing moves"



Gathering for multiple audiences



Lightweight branching

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:

writing documentation [P1, P5, P7, P10, P11]

merging cells [P11]

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
```

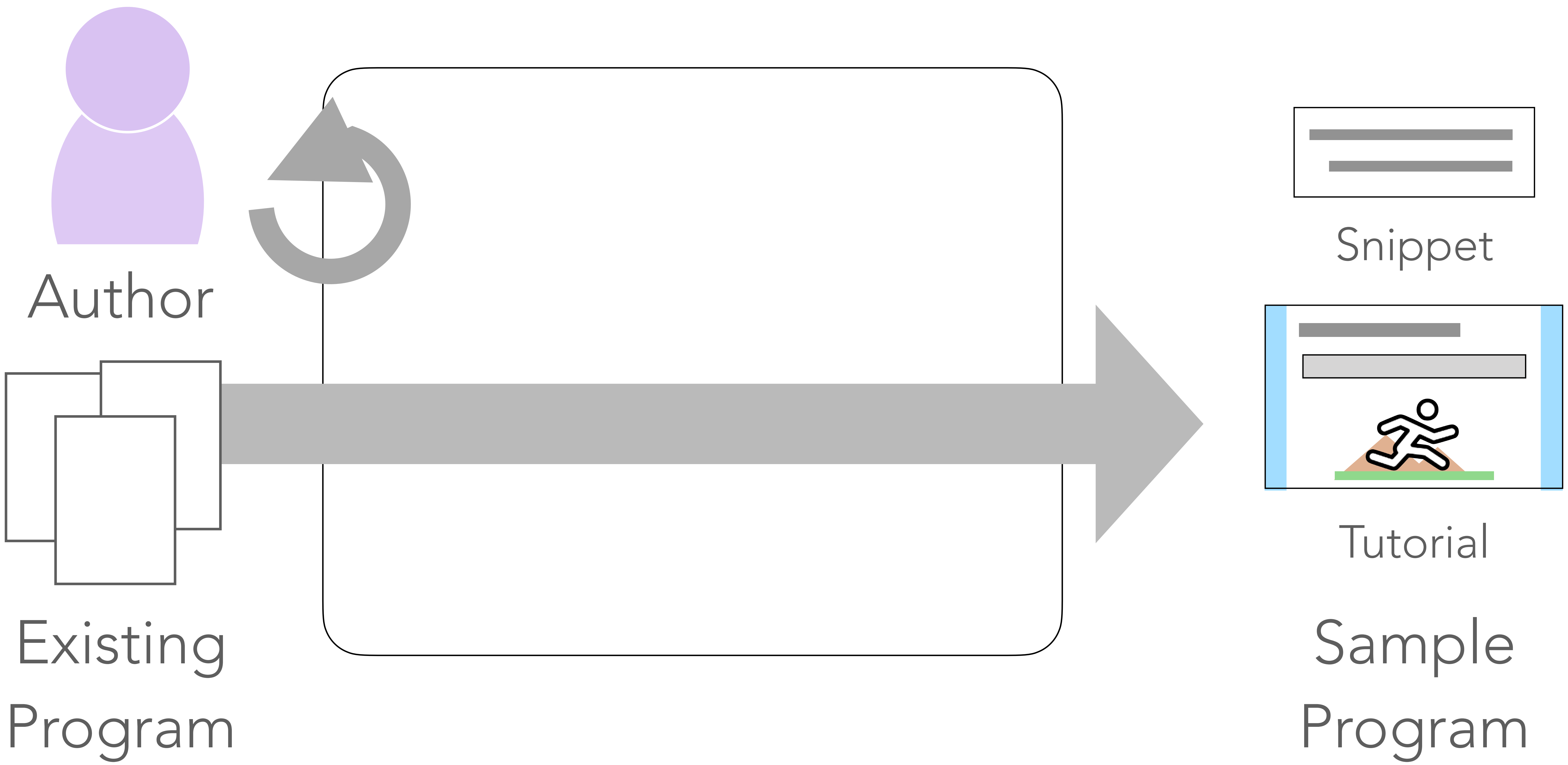
```
$ jupyter labextension install nbgather
```

Also implemented as an experimental feature in VS Code

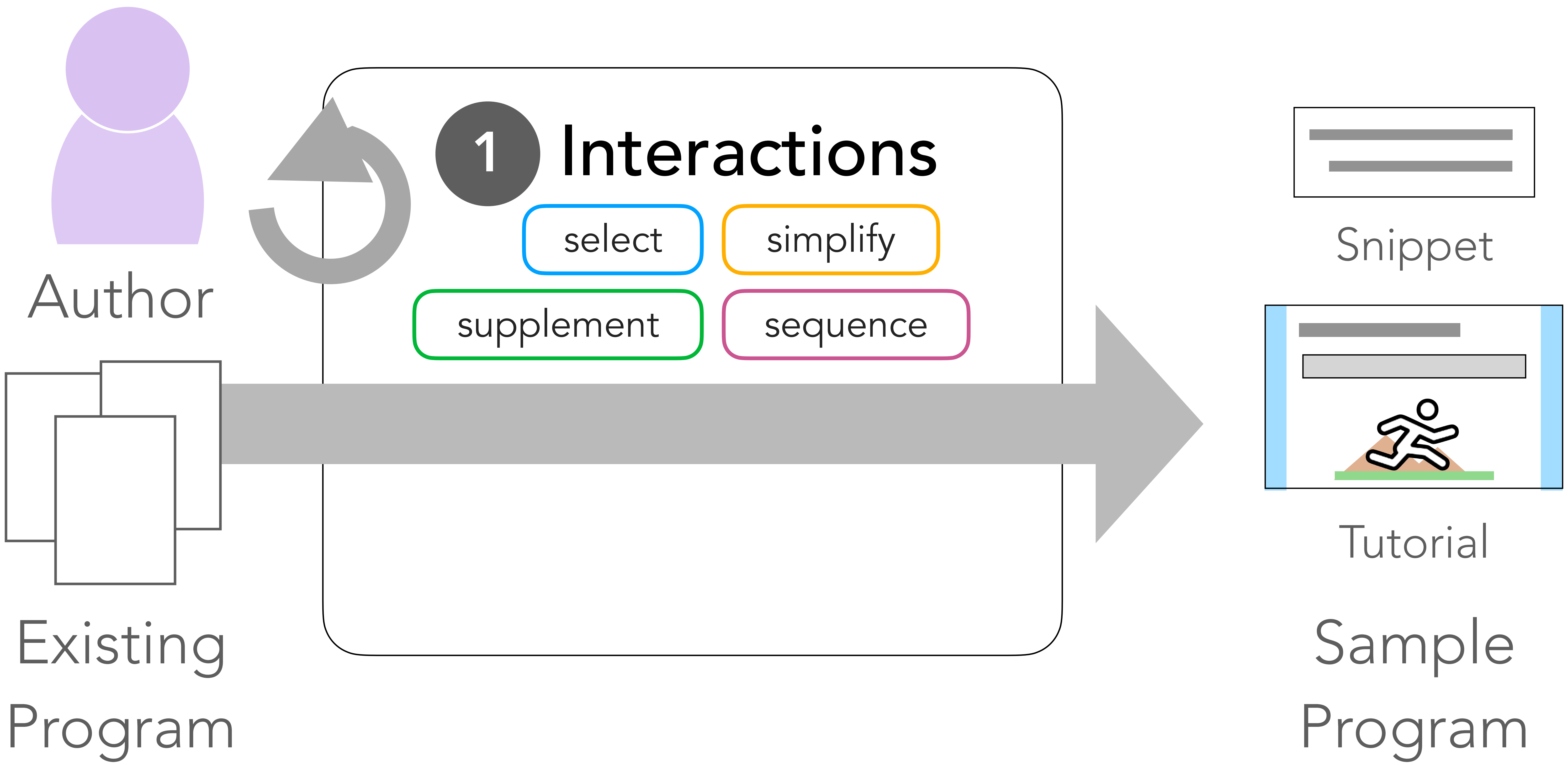
Thesis

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

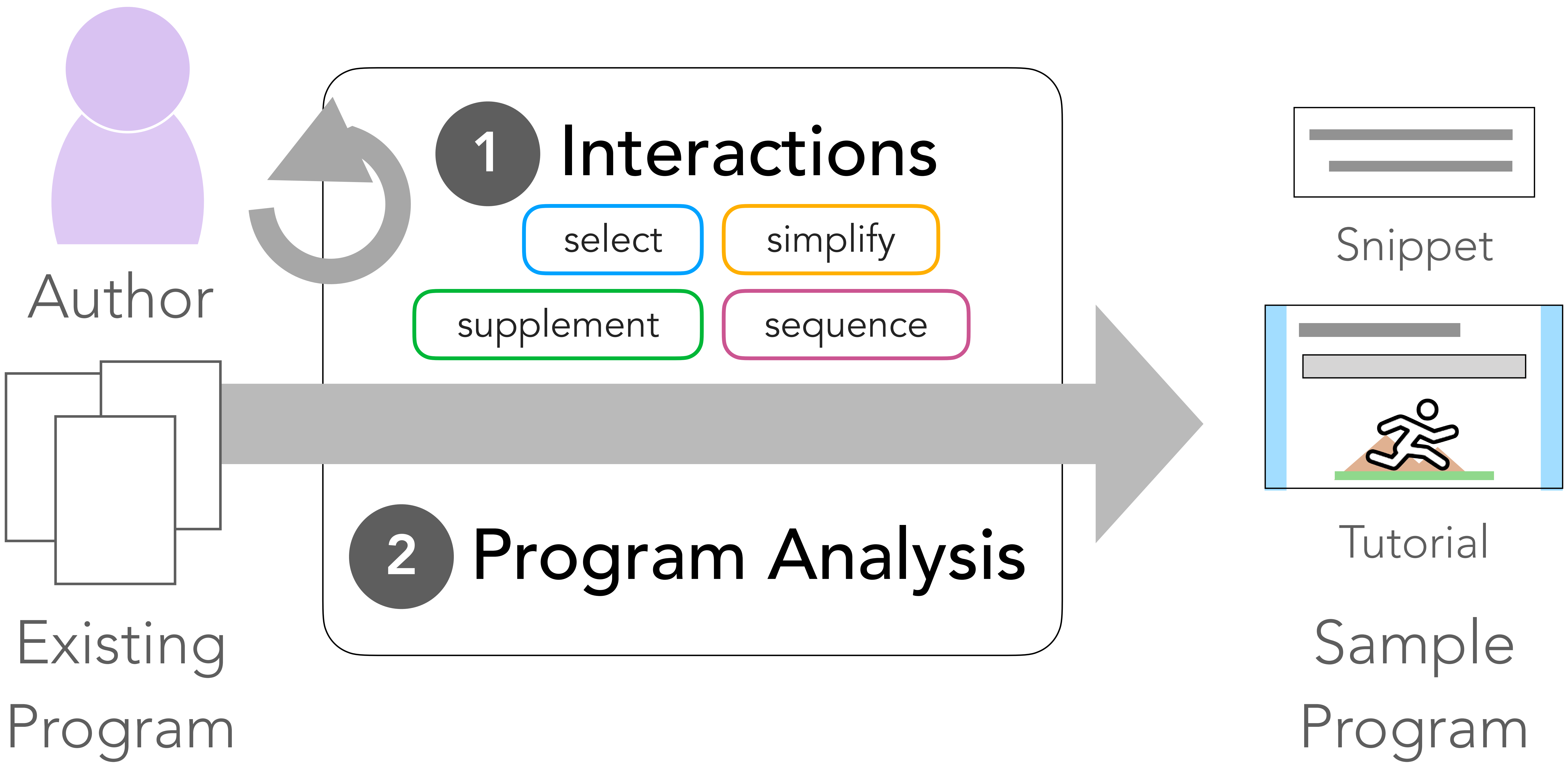
The Essence of Distillation Tools



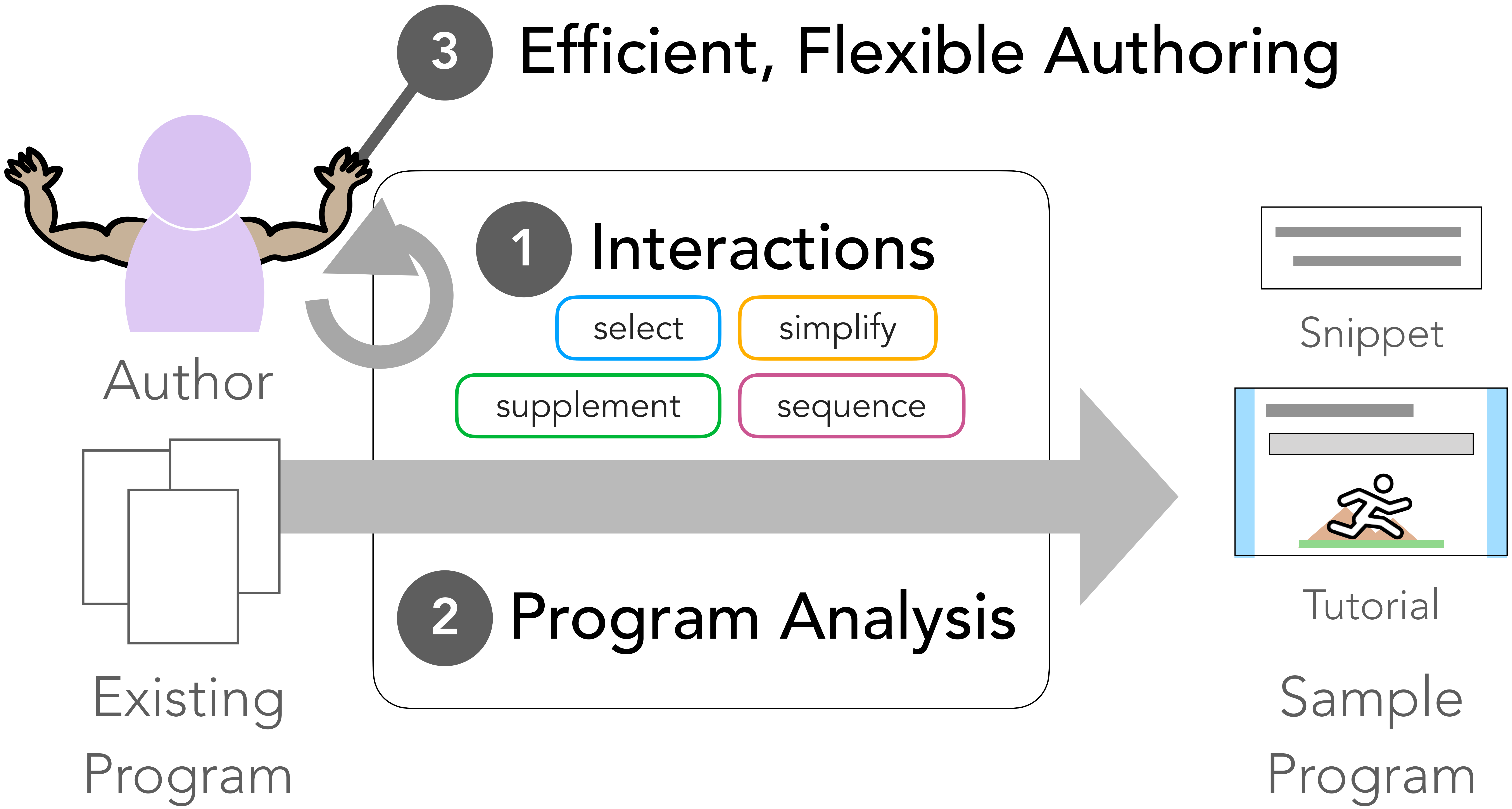
The Essence of Distillation Tools



The Essence of Distillation Tools

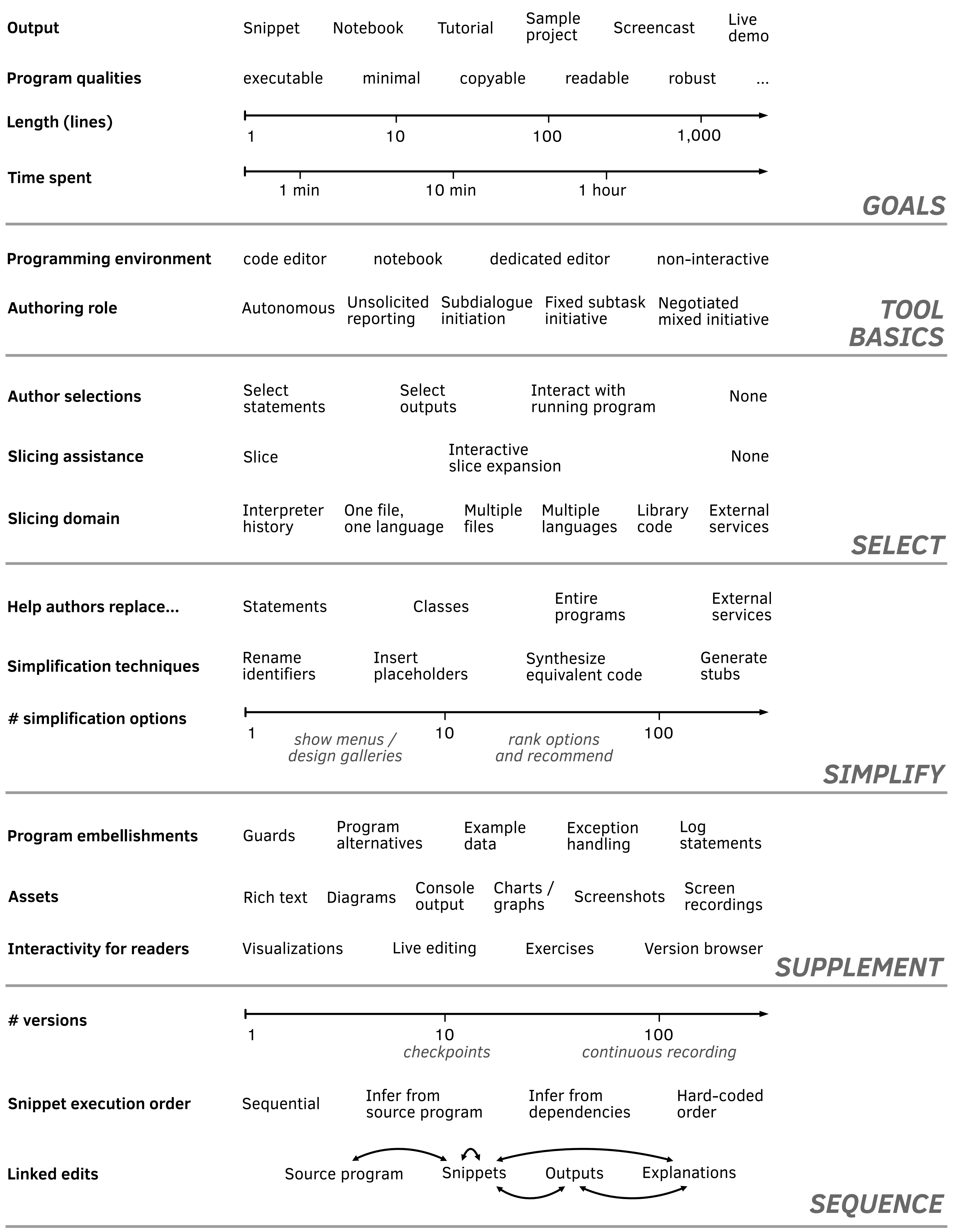


The Essence of Distillation Tools



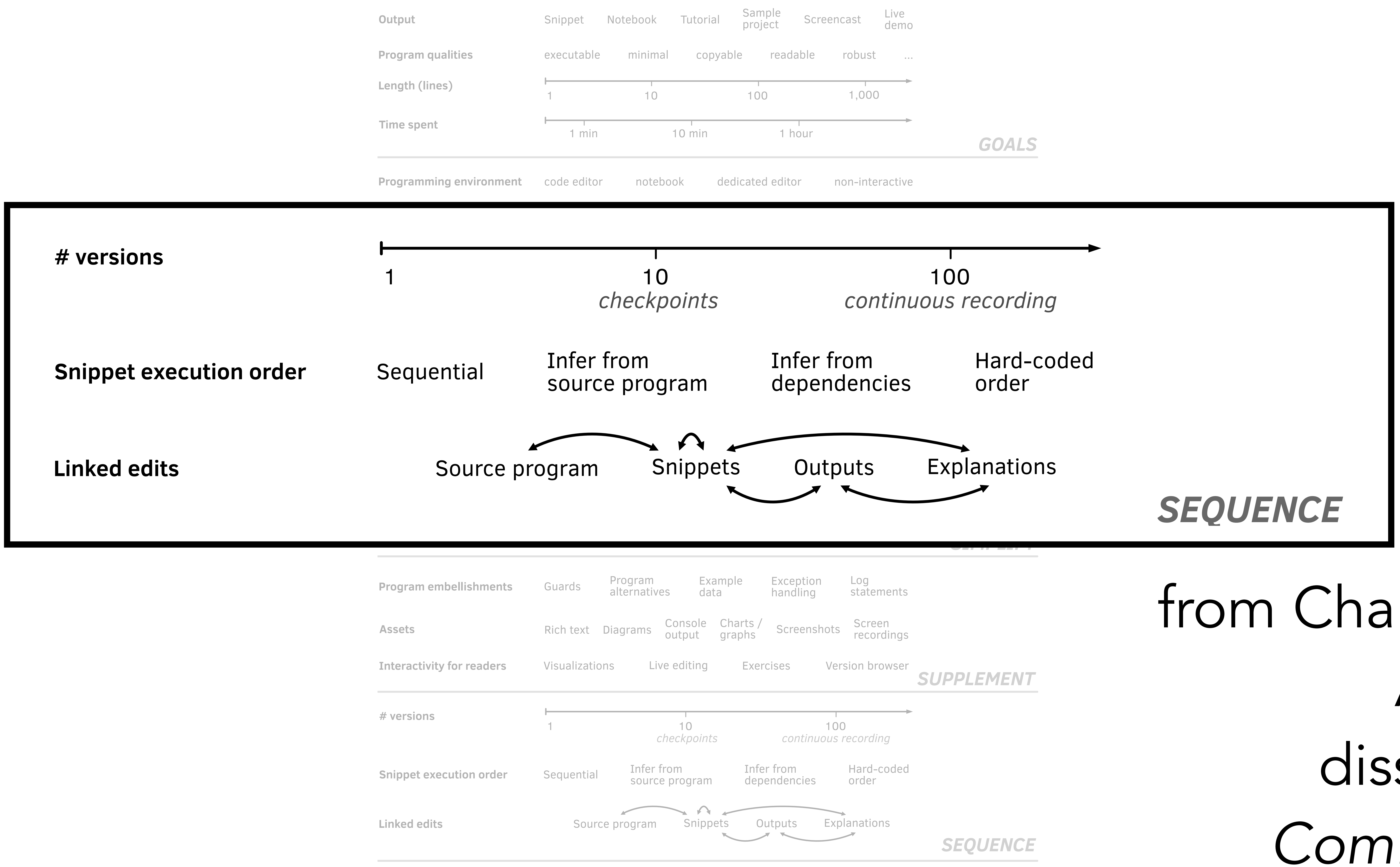
How do we help authors create more effective instructions?

A design space of distillation tools



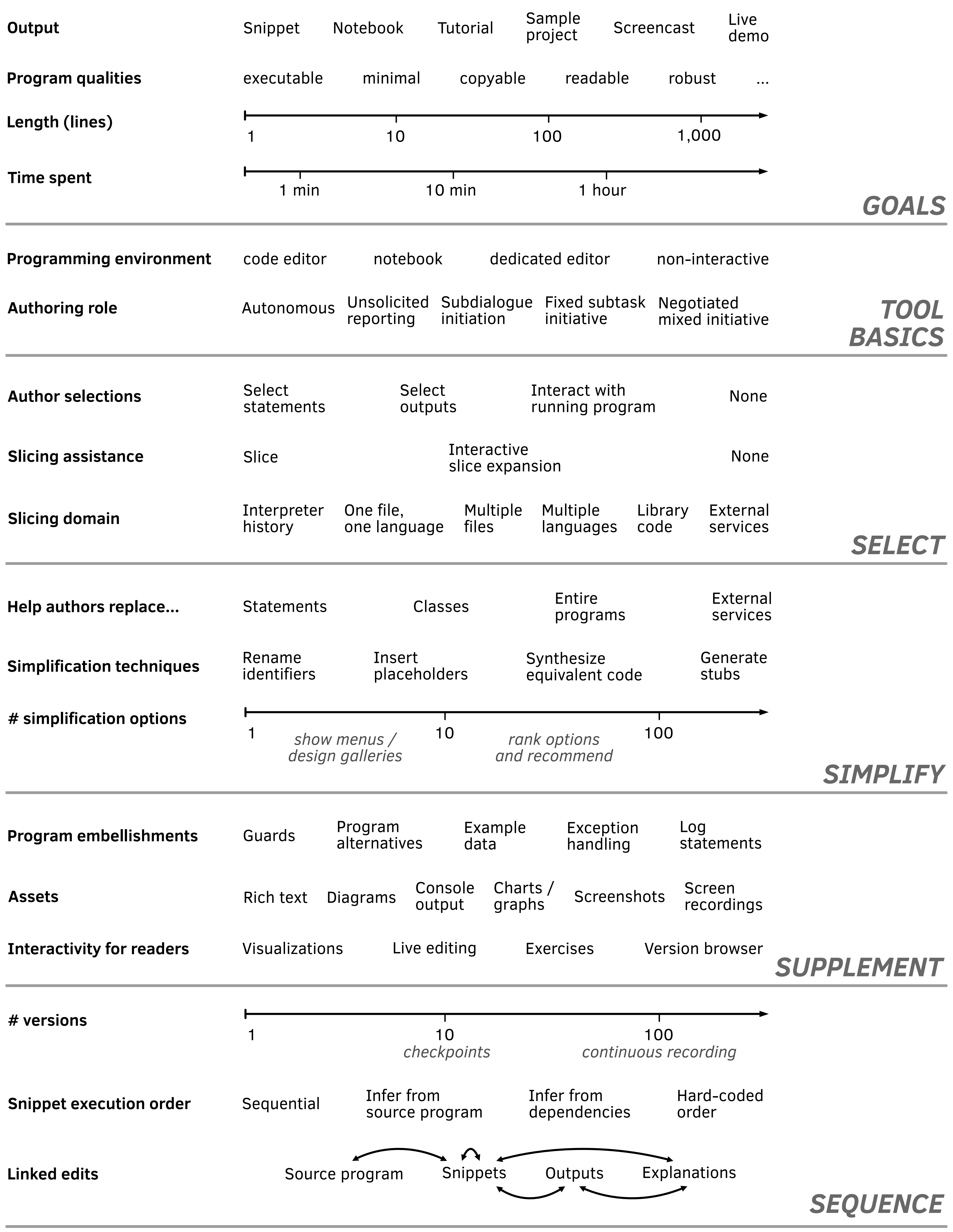
from Chapter 3 of Andrew's dissertation. *Coming soon!*

A design space of distillation tools



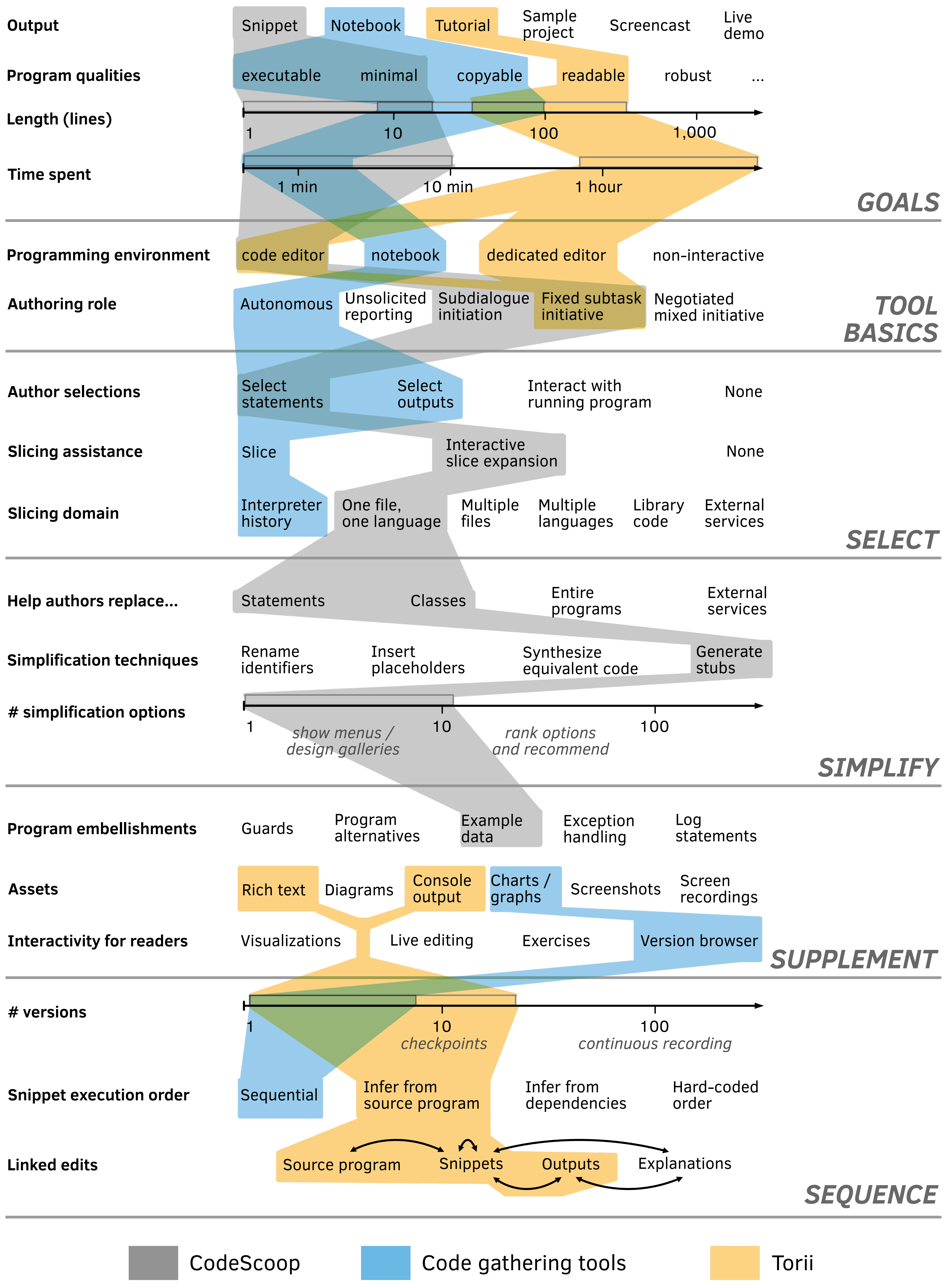
from Chapter 3 of Andrew's dissertation.
Coming soon!

A design space of distillation tools



from Chapter 3 of Andrew's dissertation. *Coming soon!*

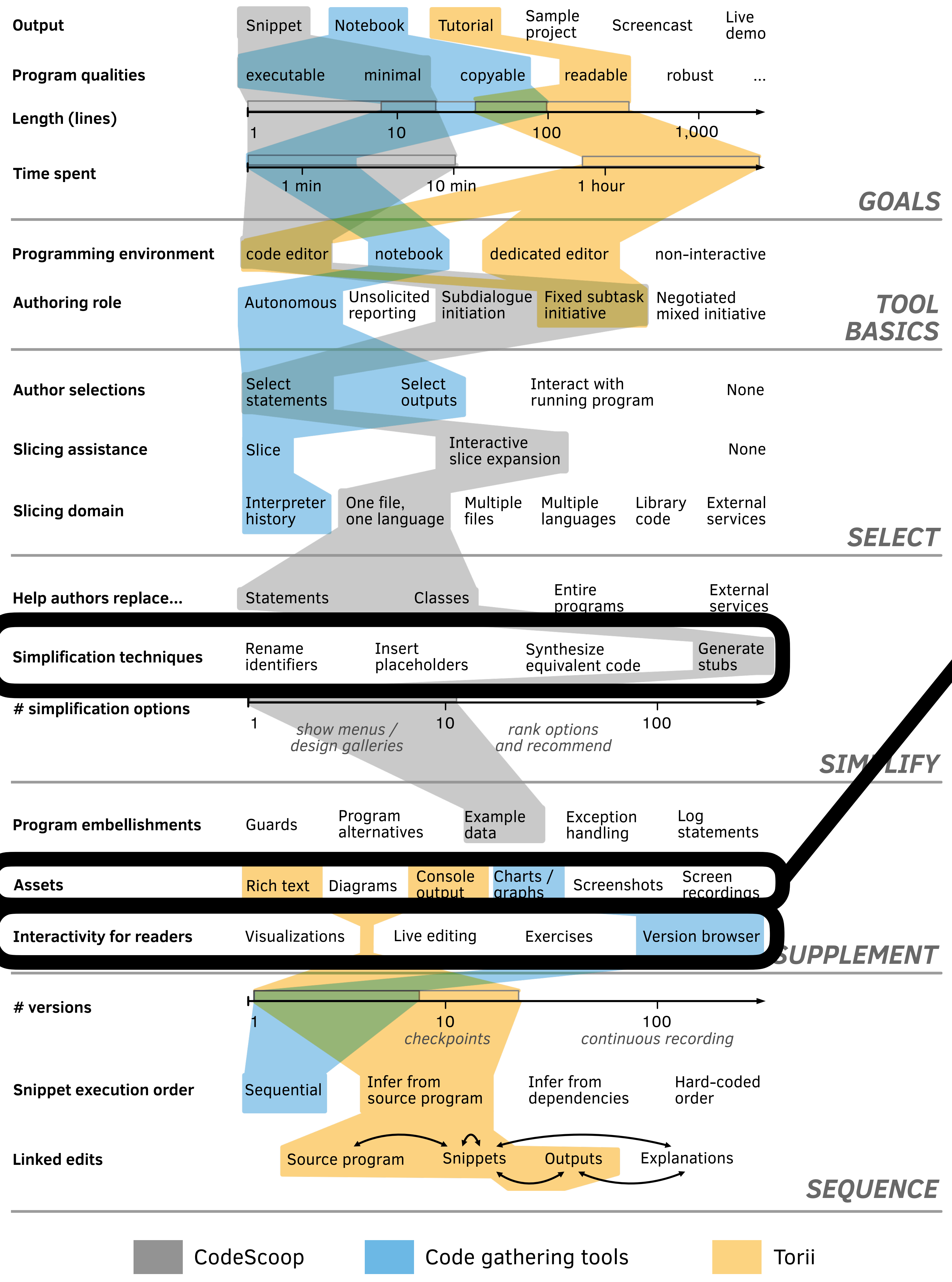
A design space of distillation tools



from Chapter 3 of Andrew's dissertation. *Coming soon!*

A design space of distillation tools

Mixed-initiative program synthesis



Natural language generation

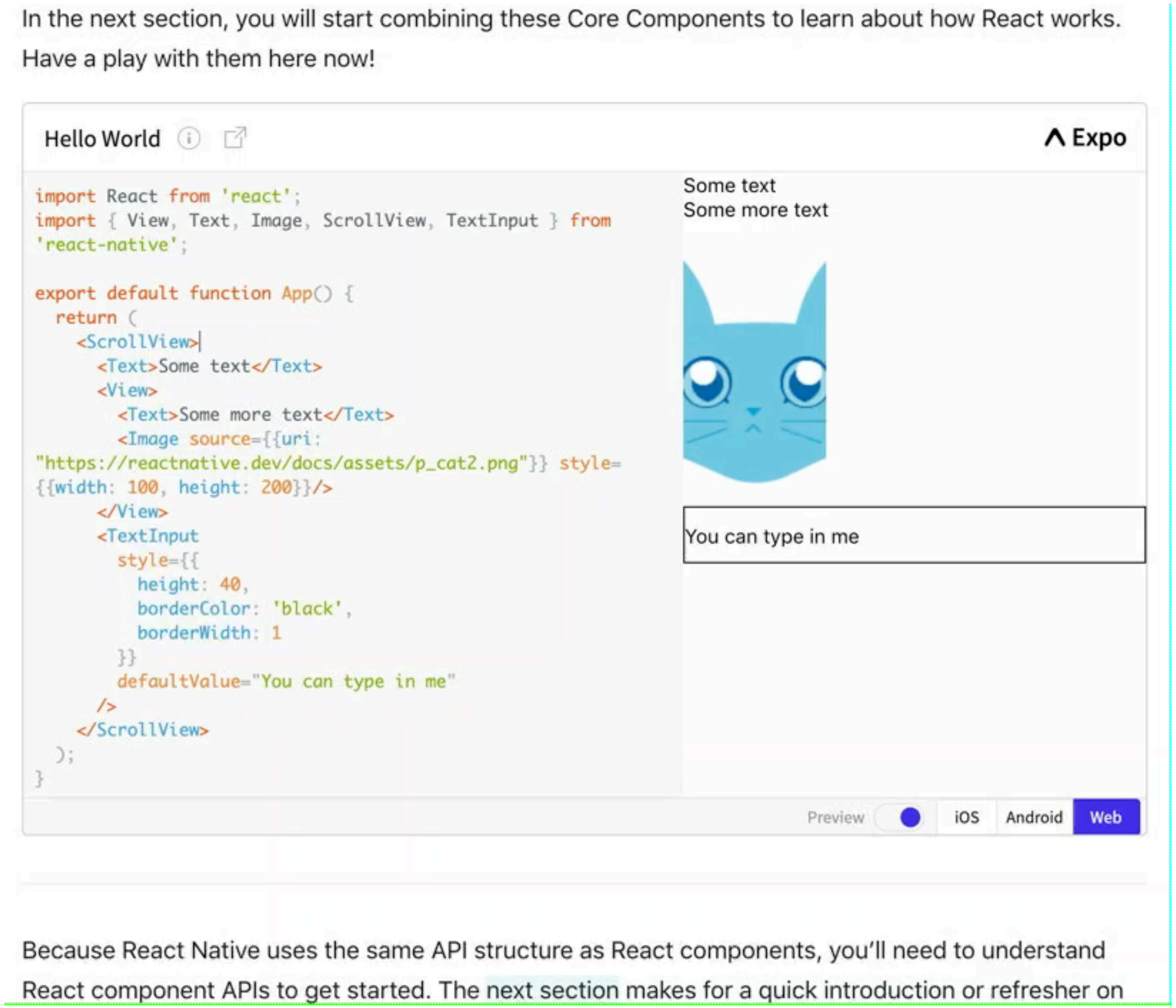
The design of explorable tutorials

from Chapter 3 of Andrew's dissertation. *Coming soon!*

The distillation of explorable tutorials

Empirical Questions

What are effective patterns for creating explroables?


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

Hello World ^ Expo

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

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

Some text
Some more text

You can type in me

Preview iOS Android Web

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

Technical Questions

How can tools help authors distill programs into instructive explorables?

The distillation of scientific discourse

ken t_k given the history $(t_1, ..., t_{k-1})$:

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

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 $\mathbf{x}_k^{LM}$ (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 $\overrightarrow{\mathbf{h}}_{k,j}^{LM}$ where $j = 1, \ldots, L$. The top layer LSTM output, $\overrightarrow{\mathbf{h}}_{k,L}^{LM}$, 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:

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

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

$$\begin{aligned} R_k &= \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} \\ &= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\}, \end{aligned}$$

where $\mathbf{h}_{k,0}^{LM}$ is the token layer and $\mathbf{h}_{k,j}^{LM} = [\overrightarrow{\mathbf{h}}_{k,j}^{LM}; \overleftarrow{\mathbf{h}}_{k,j}^{LM}]$, for each biLSTM layer.

For inclusion in a downstream model, ELMo collapses all layers in R into a single vector, $\mathbf{ELMo}_k = E(R_k; \mathbf{\Theta}_e)$. In the simplest case, ELMo just selects the top layer, $E(R_k) = \mathbf{h}_{k,L}^{LM}$, 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:

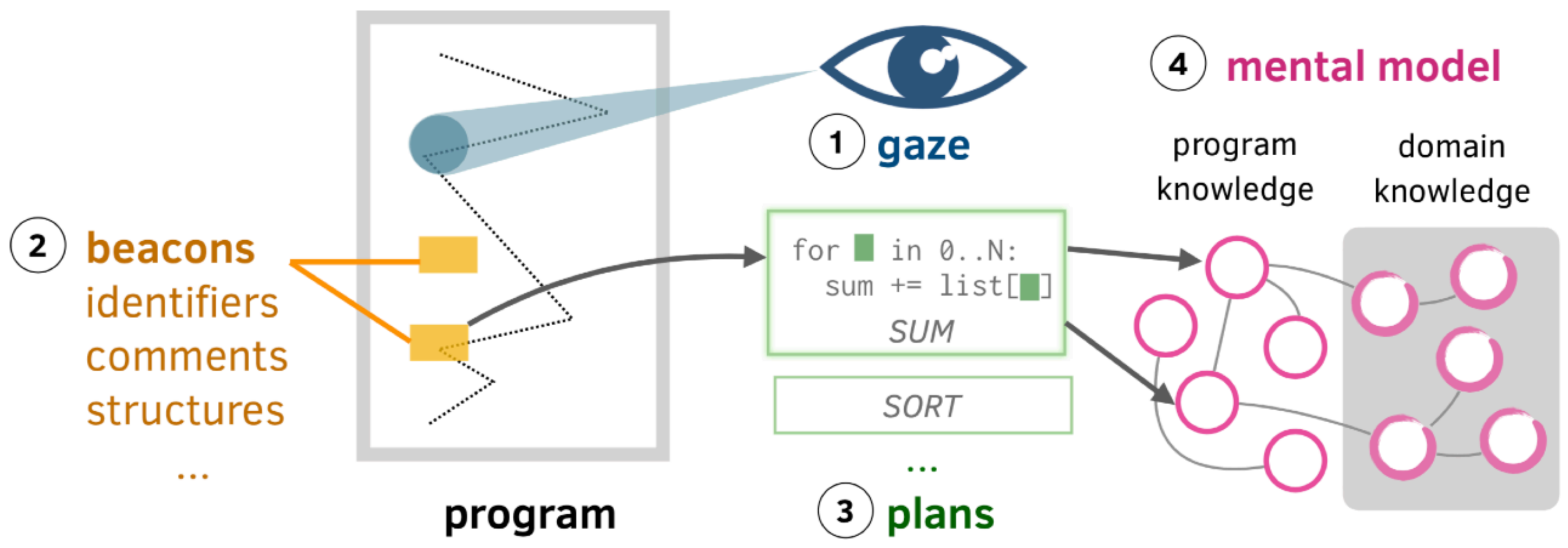
$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}. \quad (1)$$

In (1), $\mathbf{s}^{task}$ are softmax-normalized weights and the scalar parameter γ^{task} allows the task model to

layer normalization (Ba et al., 2016) to each biLM

an epilogue to the Ph.D. with Berkeley UNIVERSITY OF CALIFORNIA and Ai2 .

A plug for my dissertation



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

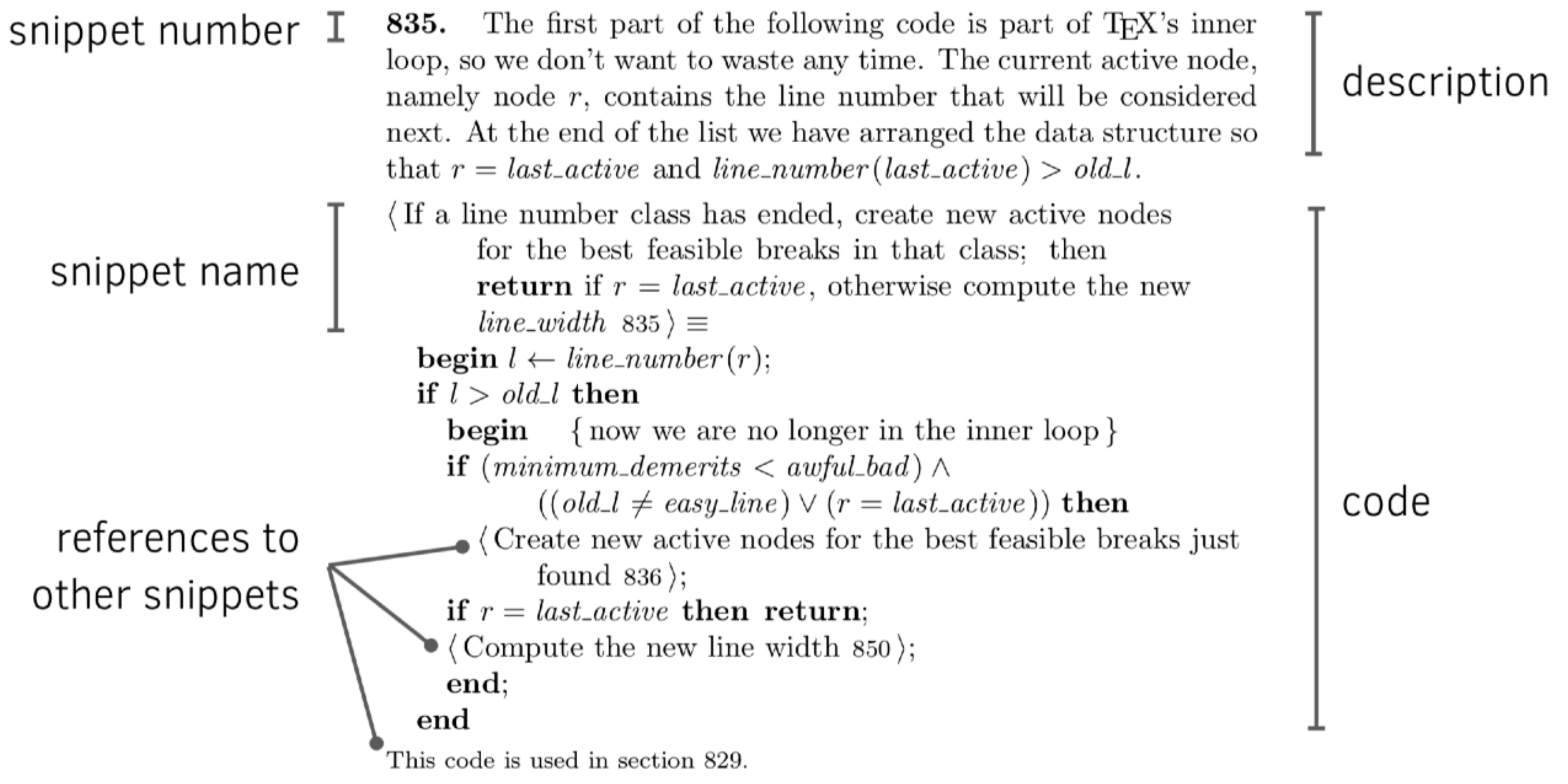


Figure 0.1: A snippet of the TEX program (Knuth 1986)

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

Björn

Marti

Thank you to my mentors and collaborators!

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







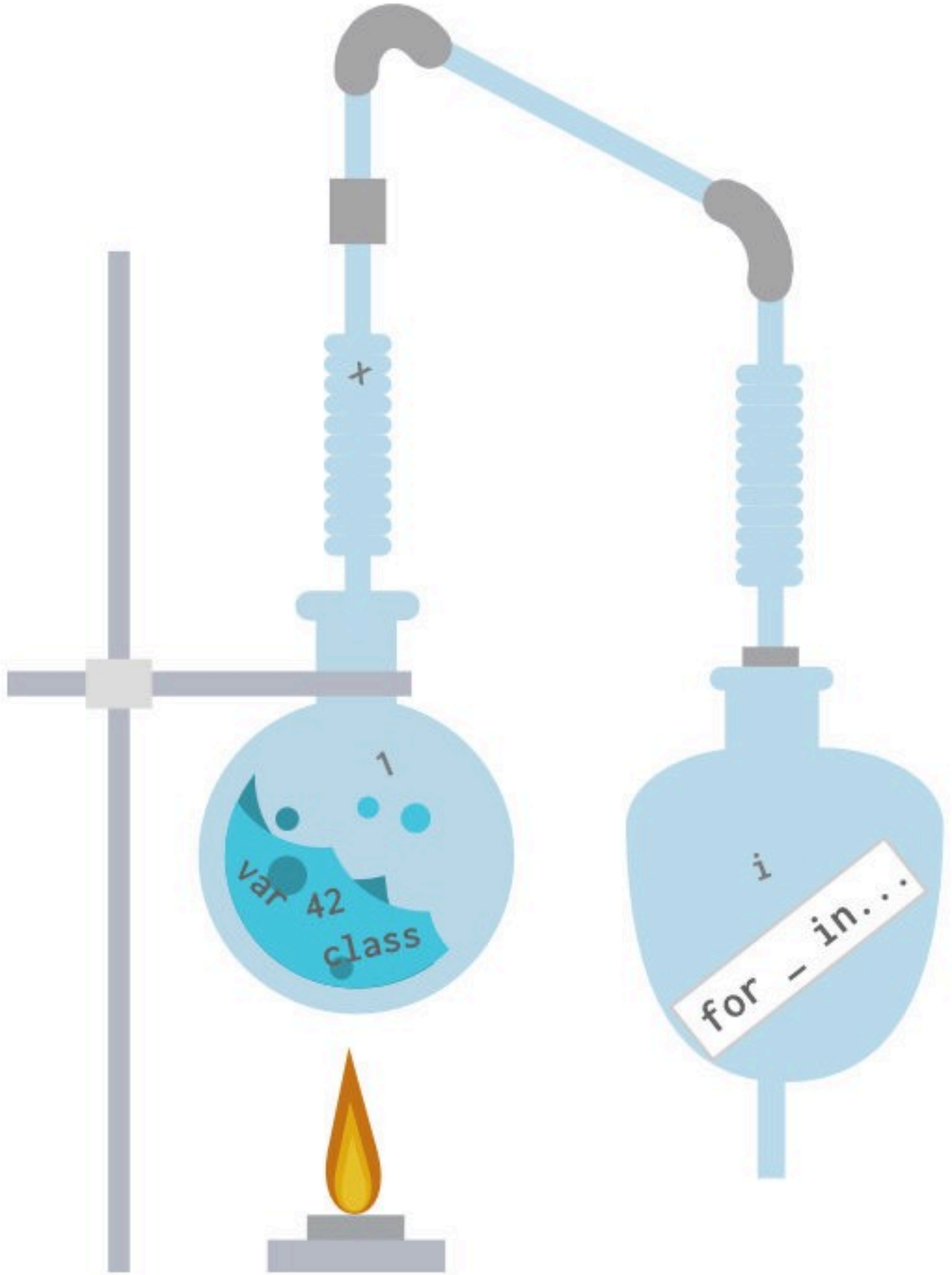


Thank you to my mentors and collaborators!

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."

Tony Hoare, *Hints on Programming Language Design*, 1974