CSCI 141

Lecture 16
String Manipulation

Announcements

Announcements

- Midterm grades are out, along with a long announcement with many details. Among them:

Announcements

- Midterm grades are out, along with a long announcement with many details. Among them:
    - Review your exam on Gradescope for 2 bonus points

Announcements

- Midterm grades are out, along with a long announcement with many details. Among them:
  - Review your exam on Gradescope for 2 bonus points
  - Grades are curved

Announcements

- Midterm grades are out, along with a long announcement with many details. Among them:
  - Review your exam on Gradescope for 2 bonus points
  - Grades are curved
  - If you do better on the final, it will replace your midterm grade.

Announcements

- Midterm grades are out, along with a long announcement with many details. Among them:
    - Review your exam on Gradescope for 2 bonus points
    - Grades are curved
    - If you do better on the final, it will replace your midterm grade.
- A4 is due Friday.

Announcements

- Midterm grades are out, along with a long announcement with many details. Among them:
    - Review your exam on Gradescope for 2 bonus points
    - Grades are curved
    - If you do better on the final, it will replace your midterm grade.
- A4 is due Friday.
    - I've updated the rubric - it's now worth 80 points.

Announcements

- Midterm grades are out, along with a long announcement with many details. Among them:
    - Review your exam on Gradescope for 2 bonus points
    - Grades are curved
    - If you do better on the final, it will replace your midterm grade.
- A4 is due Friday.
    - I’ve updated the rubric - it’s now worth 80 points.
    - If you haven’t started yet, start now.

Goals

- Review what we know already about strings:
  - the str type, + and * operators, len function

Last time…

- Returning from functions
- Using functions to wrap up complex things
- Function definition order
- Tuples:
    - packing, unpacking via the assignment operator
    - as return values and as parameters

A new data type: tuples

- A tuple is a sequence of values, optionally enclosed in parens.

  (of any types!)

```
(1, 4, “Mufasa”)
```

- You can “pack” and “unpack” them using assignment statements:

```
v = (1, 4, "Mufasa") # "packing"
(a, b, c) = v # "unpacking"
```

Docstrings, Preconditions and Postconditions

- Every function should have a docstring describing its behavior.
- When applicable, a docstring should include:
  - Preconditions: any assumptions the function must make to work.
  - Postconditions: things that are guaranteed to be true after the function finishes executing.

Reminder: Docstrings, Preconditions and Postconditions

Example. Suppose you wrote this function:

```
def split_bill(bill_amt, tip_pct, num_diners):
    """ Return the total owed by each diner for a
        restaurant bill of bill_amt, assuming a tip
        percent of tip_pct and splitting the bill
        evenly among num_diners people.

    """
    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners
```

Reminder: Docstrings, Preconditions and Postconditions

Example. Suppose you wrote this function:

```
def split_bill(bill_amt, tip_pct, num_diners):
    """ Return the total owed by each diner for a
        restaurant bill of bill_amt, assuming a tip
        percent of tip_pct and splitting the bill
        evenly among num_diners people.

    """
    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners
```

Reminder: Docstrings, Preconditions and Postconditions

Example. Suppose you wrote this function:

```
def split_bill(bill_amt, tip_pct, num_diners):
    """ Return the total owed by each diner for a
        restaurant bill of bill_amt, assuming a tip
        percent of tip_pct and splitting the bill
        evenly among num_diners people.

    """
    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners
```

```
>>> split_bill(34.78, 18.0, 0)
```

Reminder: Docstrings, Preconditions and Postconditions

Example. Suppose you wrote this function:

```
def split_bill(bill_amt, tip_pct, num_diners):
    """ Return the total owed by each diner for a
        restaurant bill of bill_amt, assuming a tip
        percent of tip_pct and splitting the bill
        evenly among num_diners people.

    """
    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners
```

```
>>> split_bill(34.78, 18.0, 0)
ZeroDivisionError: float division by zero
```

Reminder: Docstrings, Preconditions and Postconditions

Example. Suppose you wrote this function:

```
def split_bill(bill_amt, tip_pct, num_diners):
    """ Return the total owed by each diner for a
        restaurant bill of bill_amt, assuming a tip
        percent of tip_pct and splitting the bill
        evenly among num_diners people.

    """
    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners
```

```
>>> split_bill(34.78, 18.0, 0)
ZeroDivisionError: float division by zero
```

Bad news:

Reminder: Docstrings, Preconditions and Postconditions

Example. Suppose you wrote this function:

```python
def split_bill(bill_amt, tip_pct, num_diners):
    """ Return the total owed by each diner for a
        restaurant bill of bill_amt, assuming a tip
        percent of tip_pct and splitting the bill
        evenly among num_diners people.

    """
    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners
```

```
>>> split_bill(34.78, 18.0, 0)
ZeroDivisionError: float division by zero
```

Bad news: This is your fault.

Docstrings, Preconditions and Postconditions

Example. Suppose you wrote this function:

```
def split_bill(bill_amt, tip_pct, num_diners):
    """ Return the total owed by each diner for a
        restaurant bill of bill_amt, assuming a tip
        percent of tip_pct and splitting the bill
        evenly among num_diners people.
        Precondition: num_diners > 0
    """
    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners
```

Docstrings, Preconditions and Postconditions

Example. Suppose you wrote this function:

```
def split_bill(bill_amt, tip_pct, num_diners):
    """ Return the total owed by each diner for a
        restaurant bill of bill_amt, assuming a tip
        percent of tip_pct and splitting the bill
        evenly among num_diners people.
        Precondition: num_diners > 0
    """
    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners
```

Docstrings, Preconditions and Postconditions

Example. Suppose you wrote this function:

```
def split_bill(bill_amt, tip_pct, num_diners):
    """ Return the total owed by each diner for a
        restaurant bill of bill_amt, assuming a tip
        percent of tip_pct and splitting the bill
        evenly among num_diners people.
        Precondition: num_diners > 0
    """
    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners
```

```
>>> split_bill(34.78, 18.0, 0)
```

Docstrings, Preconditions and Postconditions

Example. Suppose you wrote this function:

```
def split_bill(bill_amt, tip_pct, num_diners):
    """ Return the total owed by each diner for a
        restaurant bill of bill_amt, assuming a tip
        percent of tip_pct and splitting the bill
        evenly among num_diners people.
        Precondition: num_diners > 0
    """
    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners
```

```
>>> split_bill(34.78, 18.0, 0)
ZeroDivisionError: float division by zero
```

Docstrings, Preconditions and Postconditions

Example. Suppose you wrote this function:

```
def split_bill(bill_amt, tip_pct, num_diners):
    """ Return the total owed by each diner for a
        restaurant bill of bill_amt, assuming a tip
        percent of tip_pct and splitting the bill
        evenly among num_diners people.
        Precondition: num_diners > 0
    """
    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners
```

```
>>> split_bill(34.78, 18.0, 0)
ZeroDivisionError: float division by zero
```

This is my fault.

Tuples are sequences,

so they can be used in for loops just like lists and ranges.

These two loops do the same thing:

```
for number in [1, 3]:
    print(number, ">", sep=“<", end="")

for number in (1, 3):
    print(number, ">", sep="<", end="")
```

Tuples are sequences,

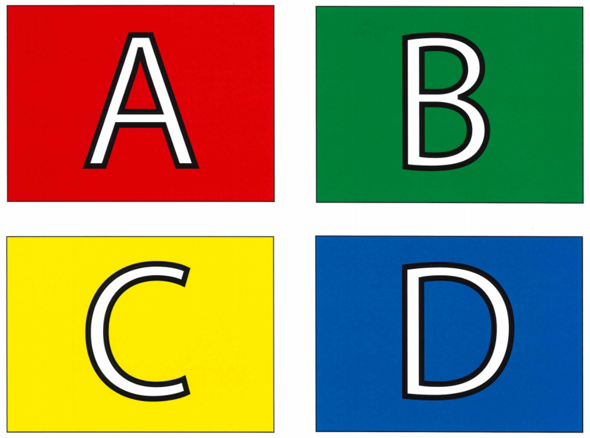
so they can be used in for loops just like lists and ranges.

These two loops do the same thing:

```
for number in [1, 3]:
    print(number, ">", sep=“<", end="")

for number in (1, 3):
    print(number, ">", sep="<", end="")
```

What do they print?



```
A. <1>        C. 1<>3
   <3>


B. 1><3       D. 1<>3<>
```

Today’s Quiz

- 3 minutes

Today’s Quiz

- 3 minutes
- Working with a neighbor: do your answers agree? (2 minutes)

Today: Strings

Don’t we already know about strings?

Today: Strings

Don’t we already know about strings?

```
type("hello")
```

Today: Strings

Don’t we already know about strings?

```
type("hello")    # => <class ‘str’>
```

Today: Strings

Don’t we already know about strings?

```
type("hello")    # => <class ‘str’>

print("Hello")
```

Today: Strings

Don’t we already know about strings?

```
type("hello")    # => <class ‘str’>

print("Hello")    # prints Hello to the console
```

Today: Strings

Don’t we already know about strings?

```
type("hello")    # => <class ‘str’>

print("Hello")    # prints Hello to the console

"Hello" + “World"
```

Today: Strings

Don’t we already know about strings?

```
type("hello")    # => <class ‘str’>

print("Hello")    # prints Hello to the console

"Hello" + “World"  # => “HelloWorld”
```

Today: Strings

Don’t we already know about strings?

```
type("hello")    # => <class ‘str’>

print("Hello")    # prints Hello to the console

"Hello" + “World"  # => “HelloWorld”

len("abc")
```

Today: Strings

Don’t we already know about strings?

```
type("hello")    # => <class ‘str’>

print("Hello")    # prints Hello to the console

"Hello" + “World"  # => “HelloWorld”

len("abc")             # => 3
```

Today: Strings

Don’t we already know about strings?

```
type("hello")    # => <class ‘str’>

print("Hello")    # prints Hello to the console

"Hello" + “World"  # => “HelloWorld”

len("abc")           # => 3

"na" * 16 + " Batman!"
```

Today: Strings

Don’t we already know about strings?

```
type("hello")    # => <class ‘str’>

print("Hello")    # prints Hello to the console

"Hello" + “World"  # => “HelloWorld”

len("abc")              # => 3

"na" * 16 + " Batman!"

# => …
```

Today: Strings

Don’t we already know about strings?

```
type("hello")    # => <class ‘str’>

print("Hello")    # prints Hello to the console

"Hello" + “World"  # => “HelloWorld”

len("abc")          # => 3

"na" * 16 + " Batman!"
# => …"nananananananananananananananana Batman!"
```

Strings: What else is there?

Strings: What else is there?

```
def house_number(address_line):
```

Strings: What else is there?

```
def house_number(address_line):
    """ Return the house number portion of
```

Strings: What else is there?

```
def house_number(address_line):
    """ Return the house number portion of
        the given address line.
```

Strings: What else is there?

```
def house_number(address_line):
    """ Return the house number portion of
        the given address line.
        Examples:
```

Strings: What else is there?

```
def house_number(address_line):
    """ Return the house number portion of
        the given address line.
        Examples:
            house_number("1600 Pennsylvania Ave")
```

Strings: What else is there?

```
def house_number(address_line):
    """ Return the house number portion of
        the given address line.
        Examples:
            house_number("1600 Pennsylvania Ave")
              => 1600
```

Strings: What else is there?

```
def house_number(address_line):
    """ Return the house number portion of
        the given address line.
        Examples:
            house_number("1600 Pennsylvania Ave")
              => 1600
            house_number("221B Baker St")
```

Strings: What else is there?

```
def house_number(address_line):
    """ Return the house number portion of
        the given address line.
        Examples:
            house_number("1600 Pennsylvania Ave")
              => 1600
            house_number("221B Baker St")
              => 221
```

Strings: What else is there?

```
def house_number(address_line):
    """ Return the house number portion of
        the given address line.
        Examples:
            house_number("1600 Pennsylvania Ave")
              => 1600
            house_number("221B Baker St")
              => 221
    """
```

Strings: What else is there?

```
def house_number(address_line):
    """ Return the house number portion of
        the given address line.
        Examples:
            house_number("1600 Pennsylvania Ave")
              => 1600
            house_number("221B Baker St")
              => 221
    """
    # ????
```

Strings: What else is there?

```
def house_number(address_line):
    """ Return the house number portion of
        the given address line.
        Examples:
            house_number("1600 Pennsylvania Ave")
              => 1600
            house_number("221B Baker St")
              => 221
    """
    # ????
    return result
```

Strings: What else is there?

```
def house_number(address_line):
    """ Return the house number portion of
        the given address line.
        Examples:
            house_number("1600 Pennsylvania Ave")
              => 1600
            house_number("221B Baker St")
              => 221
    """
    # ????
    return result
```

Strings: What else is there?

Strings: What else is there?

```
def ignore_comments(line_of_code):
```

Strings: What else is there?

```
def ignore_comments(line_of_code):
    """ Return a line of code with any comments
```

Strings: What else is there?

```python
def ignore_comments(line_of_code):
    """ Return a line of code with any comments
        after a # sign removed.
```

Strings: What else is there?

```
def ignore_comments(line_of_code):
    """ Return a line of code with any comments
        after a # sign removed.
    """
```

Strings: What else is there?

```
def ignore_comments(line_of_code):
    """ Return a line of code with any comments
        after a # sign removed.
    """
    # ????
```

Strings: What else is there?

```
def ignore_comments(line_of_code):
    """ Return a line of code with any comments
        after a # sign removed.
    """
    # ????
    return result
```

Strings: What else is there?

```
def ignore_comments(line_of_code):
    """ Return a line of code with any comments
        after a # sign removed.
    """
    # ????
    return result
```

Strings are sequences,

so they can be used in for loops just like lists and ranges.

Check this out:

```
for letter in "Bellingham":
    print(letter, "-", sep="", end="")
```

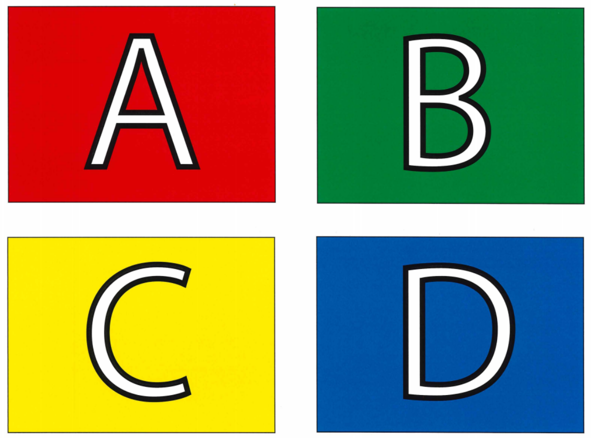
Strings are sequences,

so they can be used in for loops just like lists and ranges.

Check this out:

```
for letter in "Bellingham":
    print(letter, "-", sep="", end="")
```

What does this print?



```
A. Bellingham
B. B-e-l-l-i-n-g-h-a-m
C. -B-e-l-l-i-n-g-h-a-m
D. B-e-l-l-i-n-g-h-a-m-
```

Exercise (not collected)

Write a function that **prints** a string with all vowels removed.

```python
def remove_vowels(string):
    """ Print string, but with no vowels.
        Don't count y as a vowel. """
```

Modification: Return the modified string instead of printing it.

Indexing into Strings

(just smaller strings!)

Strings are collections of individual characters.
We can get access to an individual character by index.

```
outlook = “Summer is near”
```

How is this stored in memory?

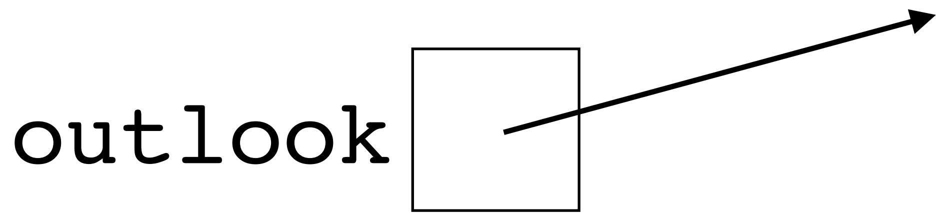
Indexing into Strings

(just smaller strings!)

Strings are collections of individual characters.
We can get access to an individual character by index.

```
outlook = “Summer is near”
```

How is this stored in memory?



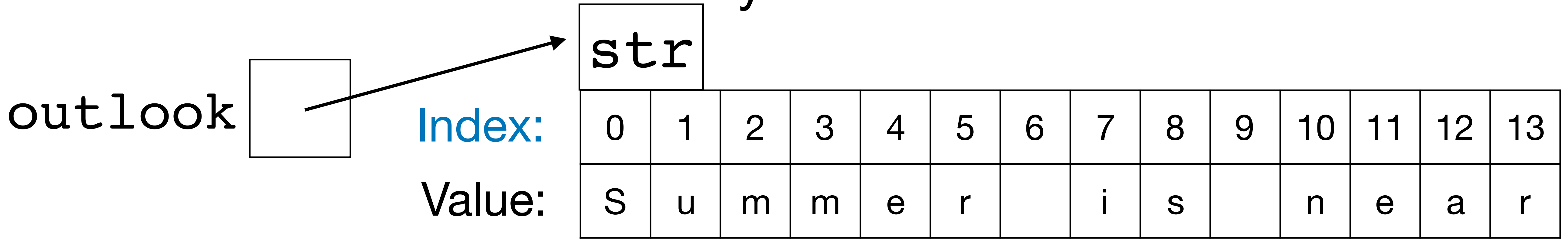
Indexing into Strings

(just smaller strings!)

Strings are collections of individual characters.
We can get access to an individual character by index.

```
outlook = “Summer is near”
```

How is this stored in memory?



Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Value:	S	u	m	m	e	r		i	s		n	e	a	r

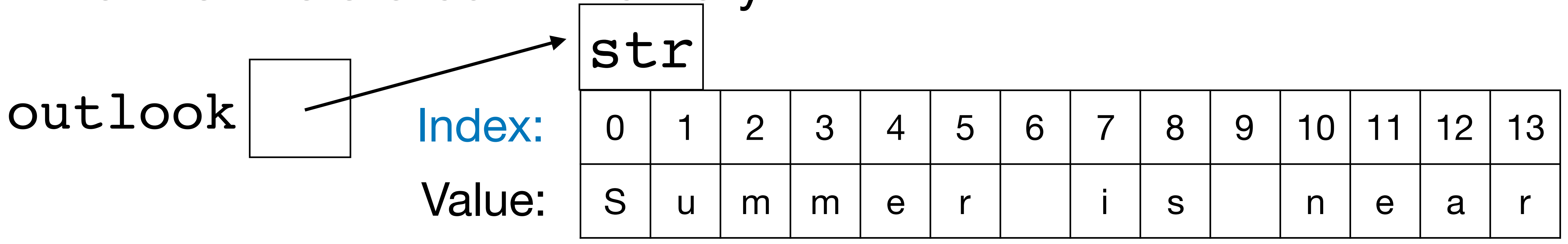
Indexing into Strings

(just smaller strings!)

Strings are collections of individual characters.
We can get access to an individual character by index.

```
outlook = “Summer is near”
```

How is this stored in memory?



Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Value:	S	u	m	m	e	r		i	s		n	e	a	r

Indices in Python begin at 0.

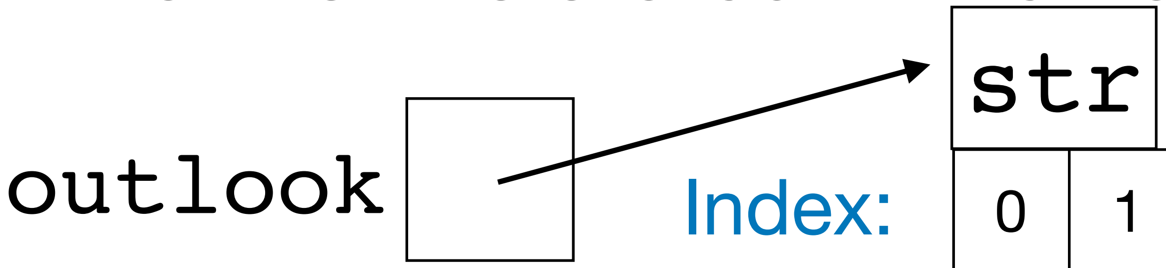
Indexing into Strings

(just smaller strings!)

Strings are collections of individual characters.
We can get access to an individual character by index.

```
outlook = “Summer is near”
```

How is this stored in memory?



Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Value:	S	u	m	m	e	r		i	s		n	e	a	r

Indices in Python begin at 0.

Syntax:

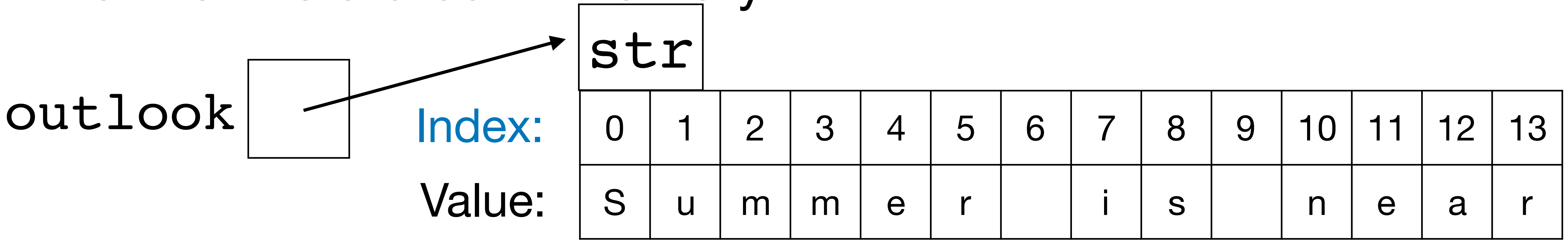
Indexing into Strings

(just smaller strings!)

Strings are collections of individual characters.
We can get access to an individual character by index.

```
outlook = “Summer is near”
```

How is this stored in memory?



Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Value:	S	u	m	m	e	r		i	s		n	e	a	r

Indices in Python begin at 0.

Syntax:

```
    outlook[0] # => "S"
```

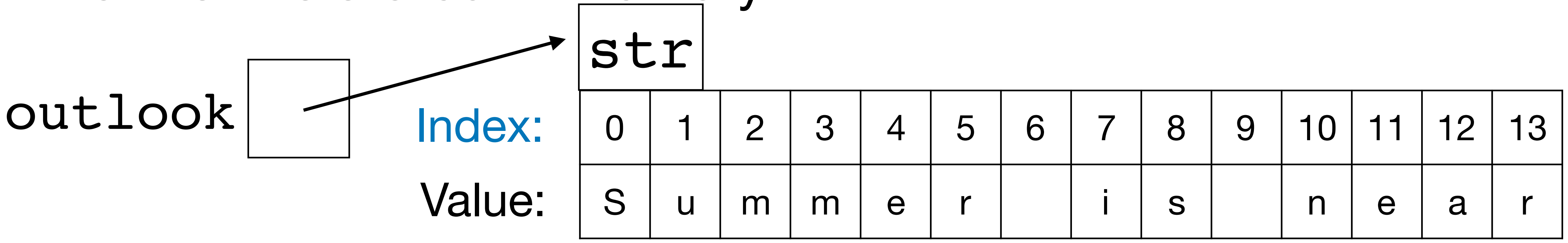
Indexing into Strings

(just smaller strings!)

Strings are collections of individual characters.
We can get access to an individual character by index.

```
outlook = “Summer is near”
```

How is this stored in memory?



Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Value:	S	u	m	m	e	r		i	s		n	e	a	r

Indices in Python begin at 0.

Syntax:

```
    outlook[0] # => "S"
    outlook[4] # => "e"
```

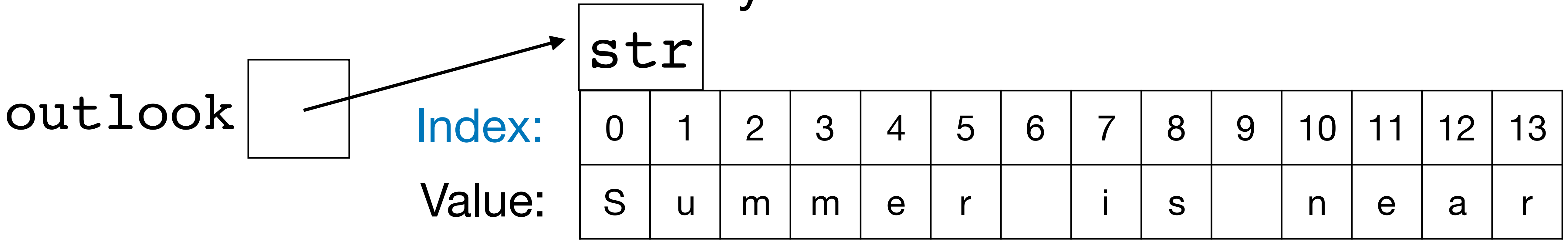
Indexing into Strings

(just smaller strings!)

Strings are collections of individual characters.
We can get access to an individual character by index.

```
outlook = “Summer is near”
```

How is this stored in memory?



Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Value:	S	u	m	m	e	r		i	s		n	e	a	r

Indices in Python begin at 0.

Spaces are characters too!

Syntax:

```
    outlook[0] # => "S"
    outlook[4] # => "e"
```

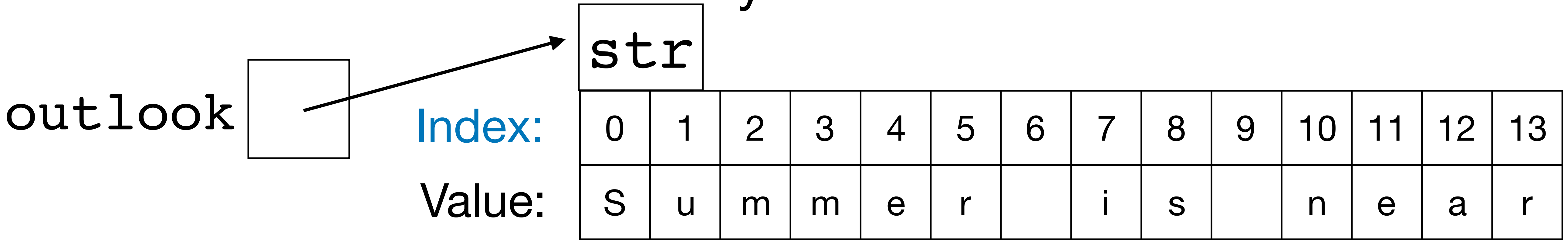
Indexing into Strings

(just smaller strings!)

Strings are collections of individual characters.
We can get access to an individual character by index.

```
outlook = “Summer is near”
```

How is this stored in memory?



Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Value:	S	u	m	m	e	r		i	s		n	e	a	r

Syntax:

```
outlook[0] # => "S"
outlook[4] # => "e"
```

Indices in Python begin at 0.

Spaces are characters too!

```
outlook[6] # => " "
```

Indexing into Strings

Strings are collections of individual characters.
We can get access to an individual character by index.

Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Value:	S	u	m	m	e	r		i	s		n	e	a	r

Problem: Return a string with any text after and including the # symbol removed.

```
def remove_comments(string):
    """ Remove all characters starting
        with a # symbol from string, and
        return the result. """
```

Indexing into Strings

(just smaller strings!)

Strings are collections of individual characters.
We can get access to an individual character by index.

Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Value:	S	u	m	m	e	r		i	s		n	e	a	r

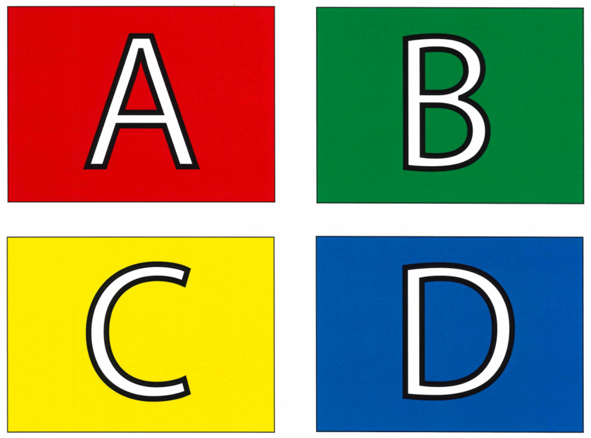
Problem: Return a string with any text after and including the # symbol removed.

```
def remove_comments(string):
    """ Remove all characters starting
        with a # symbol from string, and
        return the result. """
```

Indexing into Strings

Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Value:	S	u	m	m	e	r		i	s		n	e	a	r

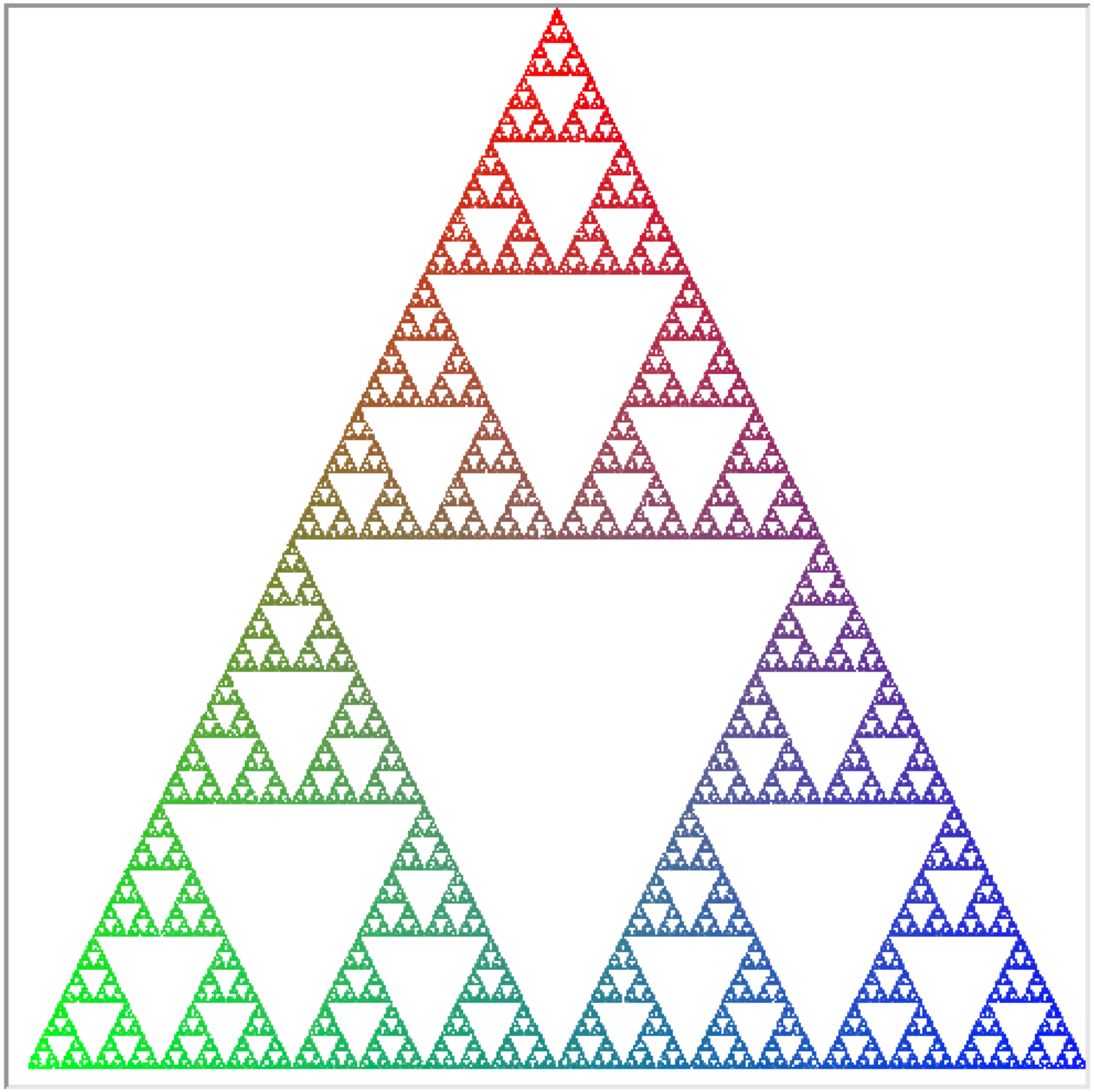
ABCD: What is the index of the last character of a string `s`?



```
A. len(s)
B. len(s - 1)
C. len(s + 1)
D. 42
```

A4 (Revisited, briefly)

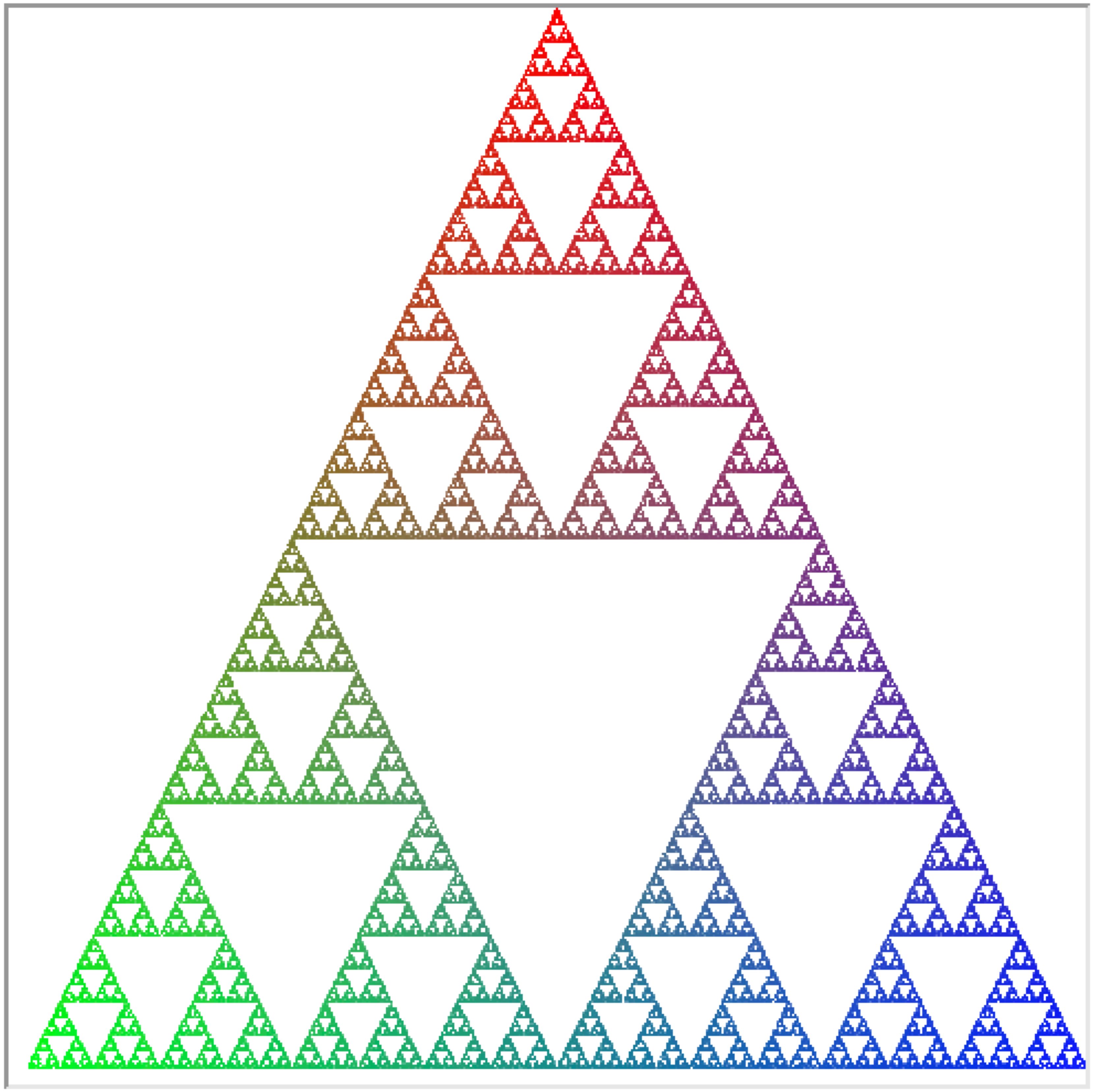
Your task:
Draw this.



A4 (Revisited, briefly)

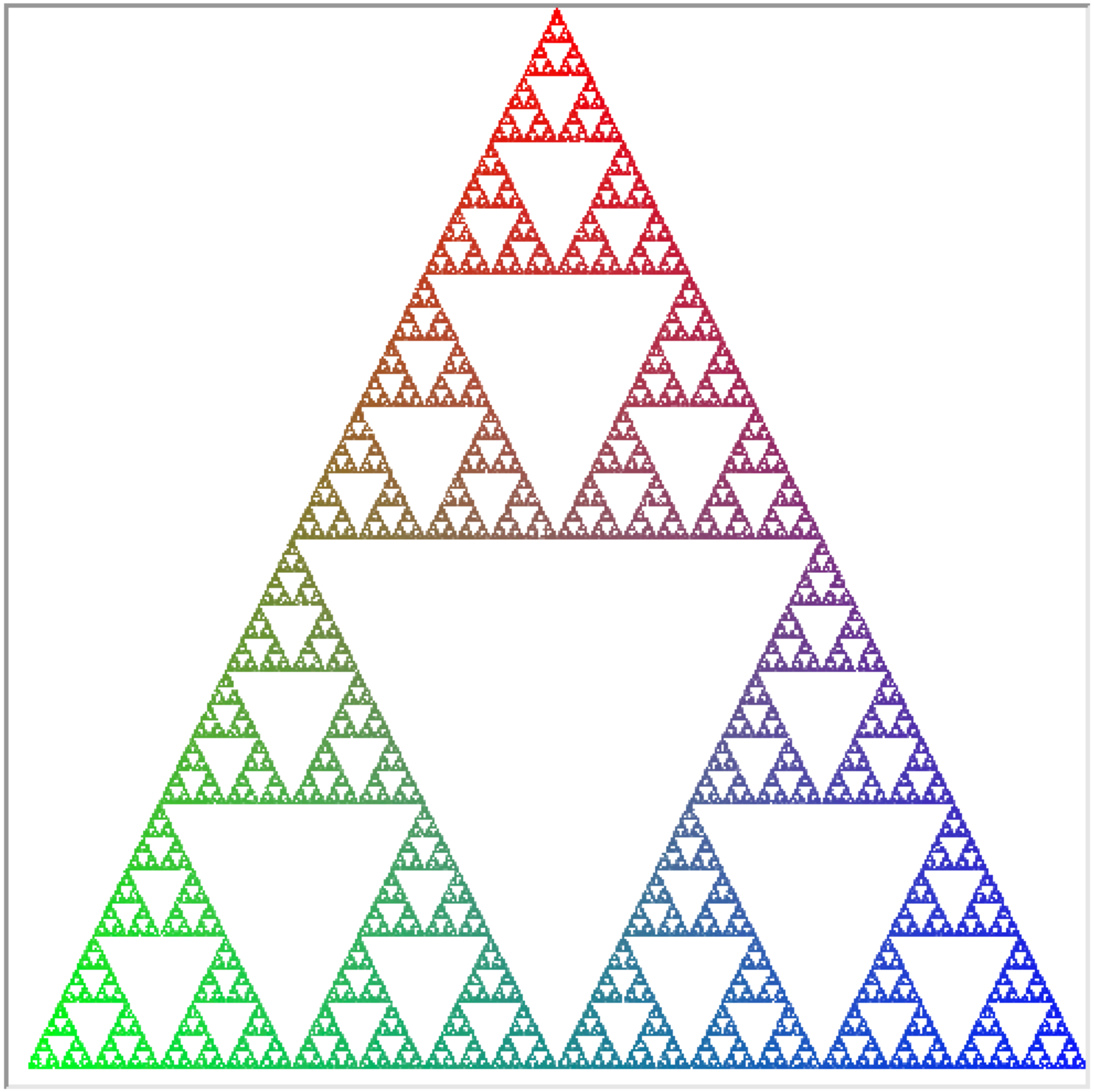
Your task:
Draw this.

Sounds
simple,
right?



A4 (Revisited, briefly)

Your task:
Draw this.



Sounds simple, right?

No.

A4: Pseudocode

```
# Let p be a random point in the window
# loop 10000 times:
#     c = a random corner of the triangle
#     m = the midpoint between p and c
#     choose a color for m
#     color the pixel at m
#     p=m
```

A4: Demo

```
# Let p be a random point in the window
# loop 10000 times:
#     c = a random corner of the triangle
#     m = the midpoint between p and c
#     choose a color for m
#     color the pixel at m
#     p=m
```

A4: Demo

```
# Let p be a random point in the window
# loop 10000 times:
#       c = a random corner of the triangle
#       m = the midpoint between p and c
#       choose a color for m
#       color the pixel at m
#       p=m
```

Demo:

- solution in action
- making up function names