Unit “Naught”: Codes (part 2)

Dave Abel

April 4th, 2016

Outline

- A look at the rest of the term
- Revisiting codes
- Unit 8: Recursion

Schedule

- 4/4 - 4/10: Recursion!
  - Please do the reading this week! It's great!
- 4/11 - 4/17: Cryptography
- 4/18 - 4/24: Graphics, Guest Lecturer on Astrophysics! (Ian Dell'antonio), Comp. Biology!
- 4/25 - 4/29: Conclusions and Review

Other Things to Come

- **Python**
  - Advanced Workshop: next week! (survey in email)
  - Python “I want to remember the first workshop”: I will email the class with some practice exercises, can come to his or the HTA’s office hours to talk and review python.
  - Project Rubric will be put on the website this week (also in upcoming email)

Other Things to Come

- **New Office Hours:** Wednesday at 11am.
- **The Last Homework Assignment** (applications unit)
  - Out 4/18, Due 5/10
  - Discuss how CS has affected a topic of interest to you
  - Read some articles and write a short reflection (800-1200 words) summarizing and analyzing your chosen topic (full rubric to be released around 4/18)

Main Idea

En**code** information in a particular way, to make the information more:

(1) Secure
(2) Safe from error
(3) Compressed

Main Idea

En**code** information in a particular way, to make the information more:

(1) Secure
(2) Safe from error
(3) Compressed

Why Do We Need Codes?

Why Do We Need Codes?

Why Do We Need Codes?

Why Do We Need Codes?

ERRORS

Cello

Why Do We Need Codes?

ERRORS

Cello

Reason One: Come up with a *code* that helps handle errors!

Why *Else* Might We Need Codes?

Why *Else* Might We Need Codes?

Why *Else* Might We Need Codes?

Why *Else* Might We Need Codes?

Why *Else* Might We Need Codes?

Why *Else* Might We Need Codes?

Why Do We Need Codes?

Reason One: Communication that is robust to errors!

Reason Two: Send information in *compressed form!*

Why Do We Need Codes?

Reason One: Communication that is robust to errors!

(Reason Three: keep messages secure)

Reason Two: Send information in *compressed form!*

Error Correcting Codes

- Goal One: what if our messages could *tell us* when there was a mistake?

Error Correcting Codes

- Goal One: what if our messages could *tell us* when there was a mistake?

Suppose I send you our zip code

02906

Error Correcting Codes

- Goal One: what if our messages could *tell us* when there was a mistake?

Suppose I send you our zip code

Error Correcting Codes

- Goal One: what if our messages could *tell us* when there was a mistake?

Suppose I send you our zip code

02906

First Idea: Repetition

- Send it twice! (or more)
- If they're the same, great!
- If they're different, ask to be sent the message again

Suppose I send you our zip code

02906

02906

First Idea: Repetition

- Send it twice! (or more)
- If they're the same, great!
- If they're different, ask to be sent the message again

Suppose I send you our zip code

First Idea: Repetition

- Send it twice! (or more)
- If they’re the same, great!
- If they’re different, ask to be sent the message again

Suppose I send you our zip code

02905
02906

First Idea: Repetition

- Send it twice! (or more)
- If they're the same, great!
- If they're different, ask to be sent the message again
- **Problem: may have to send messages *many times***

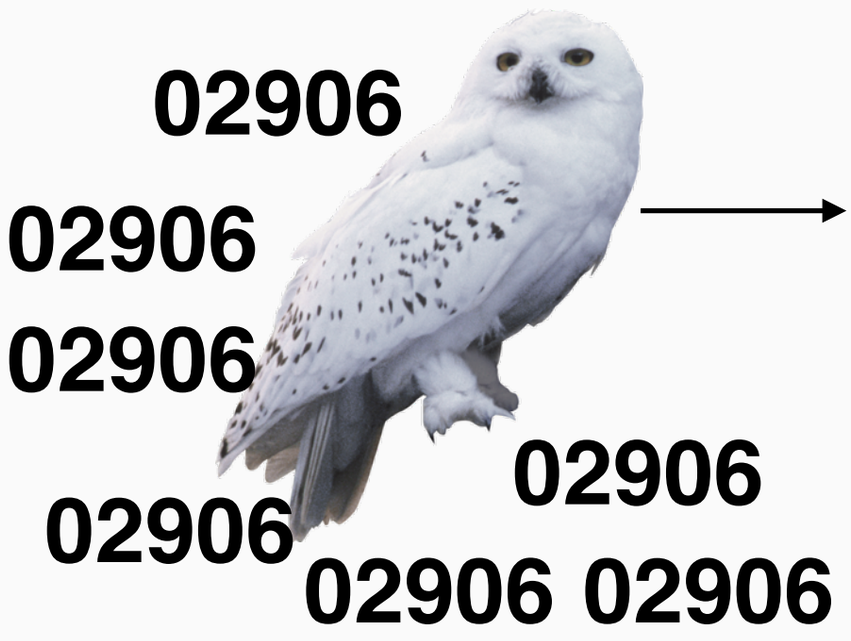
First Idea: Repetition

- Send it twice! (or more)
- If they're the same, great!
- If they're different, ask to be sent the message again
- Problem: may have to send messages *many times*

Solution: can use multiple copies to recreate the original!

First Idea: Repetition

Suppose I send
you our zip code



Solution: can use multiple copies to recreate the original!

First Idea: Repetition

Suppose I send
you our zip code

02906

12906

02906

02905

02906

03906

02906

Solution: can use multiple copies to recreate the original!

First Idea: Repetition

Suppose I send you our zip code

Q: How can we recover the original from these?

02906

12906

02906

02905

02906

03906

02906

Solution: can use multiple copies to recreate the original!

First Idea: Repetition

Q: How can we recover the original from these?

02906
12906
02906
02905
02906
03906
02906

Solution: can use multiple copies to recreate the original!

First Idea: Repetition

Q: How can we recover the original from these?

Idea: most common digit is correct

02906
12906
02906
02905
02906
03906
02906

Solution: can use multiple copies to recreate the original!

First Idea: Repetition

Q: How can we recover the original from these?

Idea: most common digit is correct

02906

Solution: can use multiple copies to recreate the original!

First Idea: Repetition

- Send it twice! (or more)
- If they're the same, great!
- If they're different, ask to be sent the message again
- Problem: may have to send messages *many times*
- Solution: can use multiple copies to recreate the original! Don't need to resend over and over again.
- **New Problem:** now we have to send way more stuff!

Second Idea: Checksum

- Goal: Send *less* information, but still get error-robust codes
- Idea: add some information to the message that can help detect/fix errors!

Second Idea: Checksum

- Goal: Send *less* information, but still get error-robust codes
- Idea: add some information to the message that can help detect/fix errors!
- Checksum: add the sum of the message *mod 10* to the end!

Second Idea: Checksum

- Checksum: add the sum of the message *mod 10* to the end!

- **Example**: 02906, the sum is 0 + 2 + 9 + 0 + 6 = 17

Second Idea: Checksum

- Checksum: add the sum of the message *mod 10* to the end!

- **Example**: 02906, the sum is 0 + 2 + 9 + 0 + 6 = 17

- So we add 17 *mod 10* to the end, which is **029067**.

- Then, upon receiving a message, we check to make sure the checksum is still correct! If it's not, we ask the sender to send another message.

Clicker Question!

- Idea: add some information to the message that can help detect errors!
- Checksum: add the sum of the message *mod 10* to the end!

Q: Is this scheme perfect? Will you *always* catch errors?

[A] Yes [B] No [C] I'm Confused

Clicker Answer!

- Idea: add some information to the message that can help detect errors!
- Checksum: add the sum of the message *mod 10* to the end!

Q: Is this scheme perfect? Will you *always* catch errors?

[A] Yes **[B] No** [C] I'm Confused

Clicker Answer!

5040

Q: Is this scheme perfect? Will you *always* catch errors?

[A] Yes **[B] No** [C] I'm Confused

Clicker Answer!

5040 ⟶ 50409

Q: Is this scheme perfect? Will you *always* catch errors?

[A] Yes **[B] No** [C] I'm Confused

Clicker Answer!

5040 ⟶ 50409 ⟶

Q: Is this scheme perfect? Will you *always* catch errors?

[A] Yes **[B] No** [C] I'm Confused

Clicker Answer!

5040 ⟶ 50409 ⟶  ⟶ 50308

Q: Is this scheme perfect? Will you *always* catch errors?

[A] Yes **[B] No** [C] I'm Confused

Error Correcting Codes

- In a computer, communication errors happen regularly!
  - Errors in transmitting to and from the internet
  - Errors reading/writing from our computers memory
    - Old news: Errors reading/writing to CDs, floppies.
- We want to know when errors happen so we can fix them.
- Solutions:
  - Send copies! (But too much space)
  - Send a checksum!

Compression

- Idea: We can turn *big* things into *small* things that effectively *preserve the main information*.

Compression

- Idea: We can turn *big* things into *small* things that effectively *preserve the main information*.

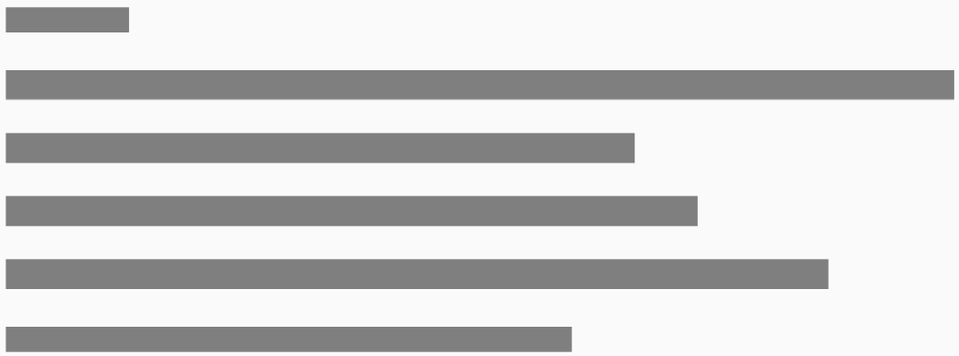
Compression

- Idea: We can turn *big* things into *small* things that effectively *preserve the main information*.

, but way bigger

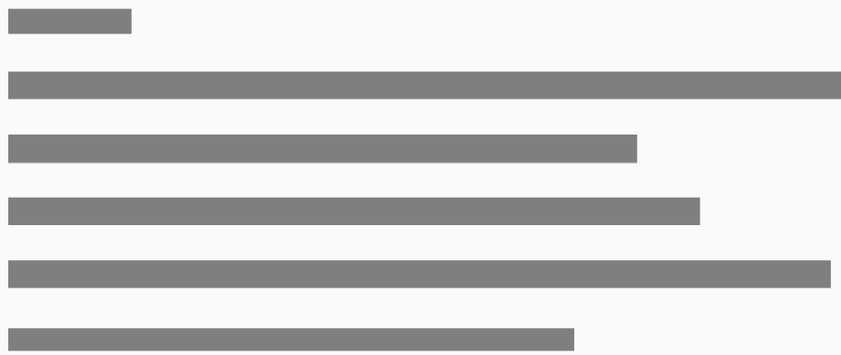
Simple Explanations

Challenge 5:



Simple Explanations

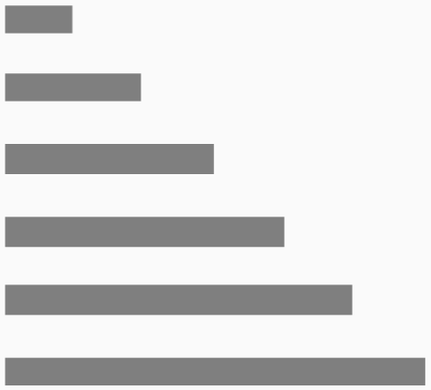
Challenge 5:



Shortest description: six bar histogram switch 2 and 6

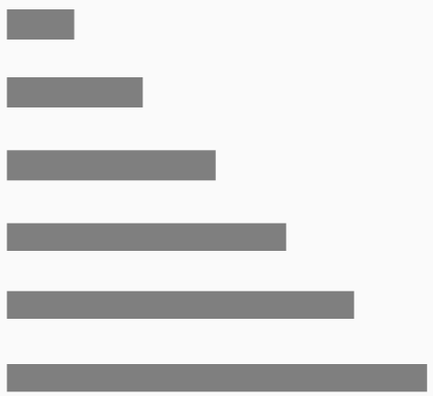
Simple Explanations

Challenge 6:



Simple Explanations

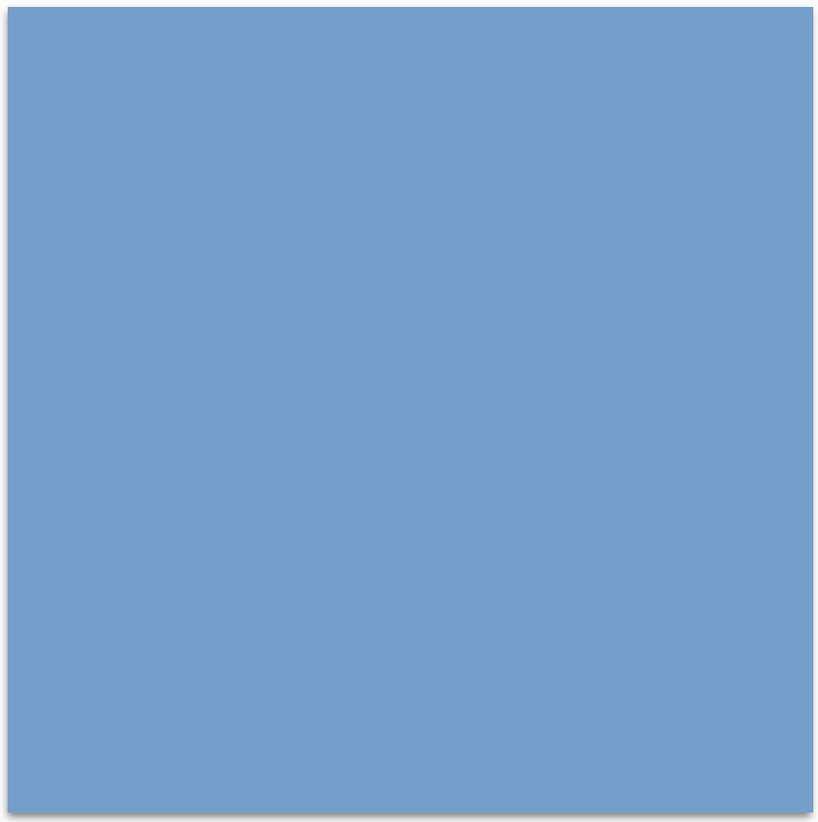
Challenge 6:

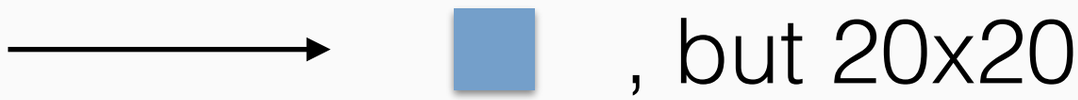


Shortest description: bars ordered

Compression

- Q: How can we make an object *as small as possible*, but still preserve what the object is?





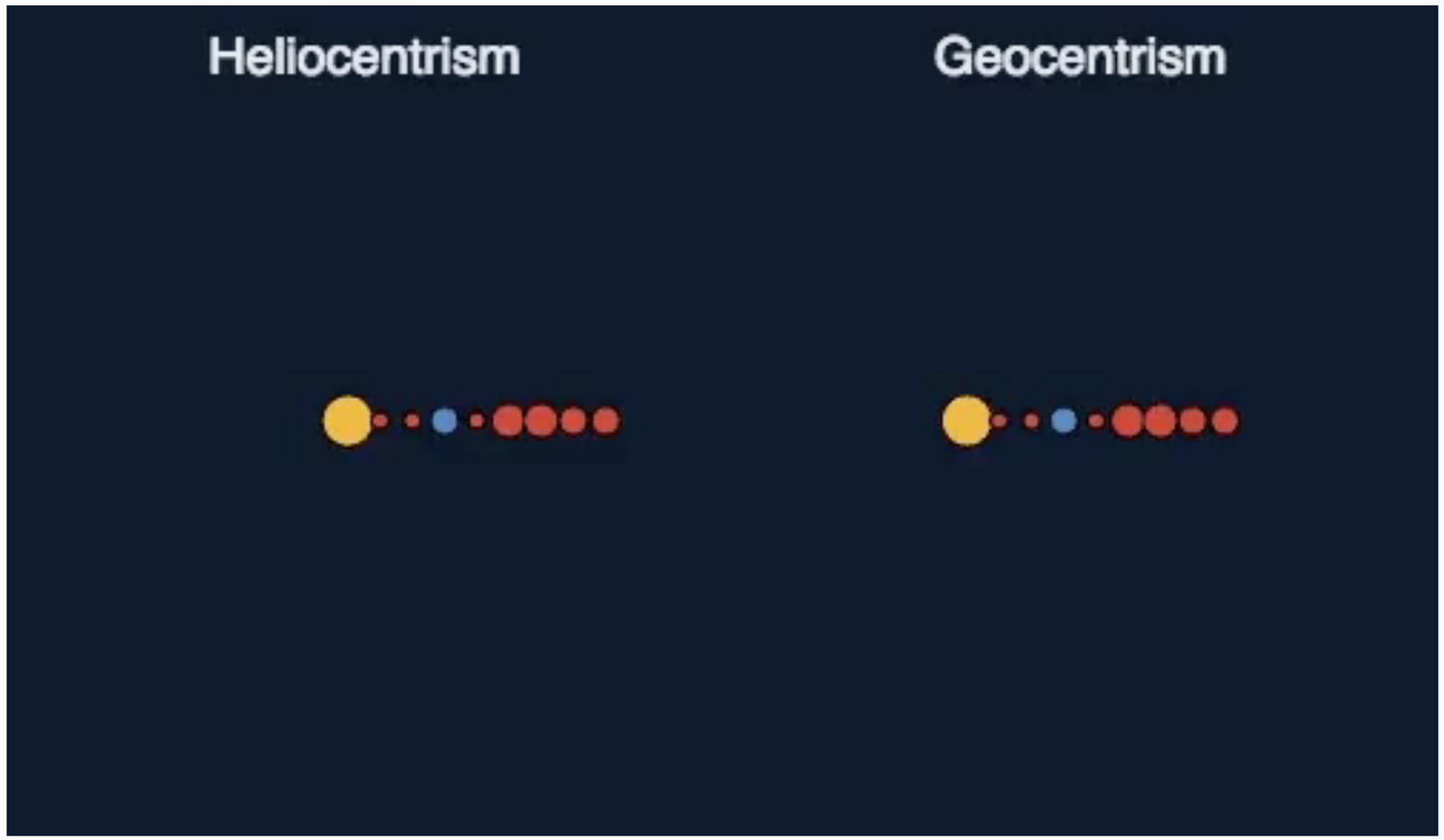
, but 20x20

Compression

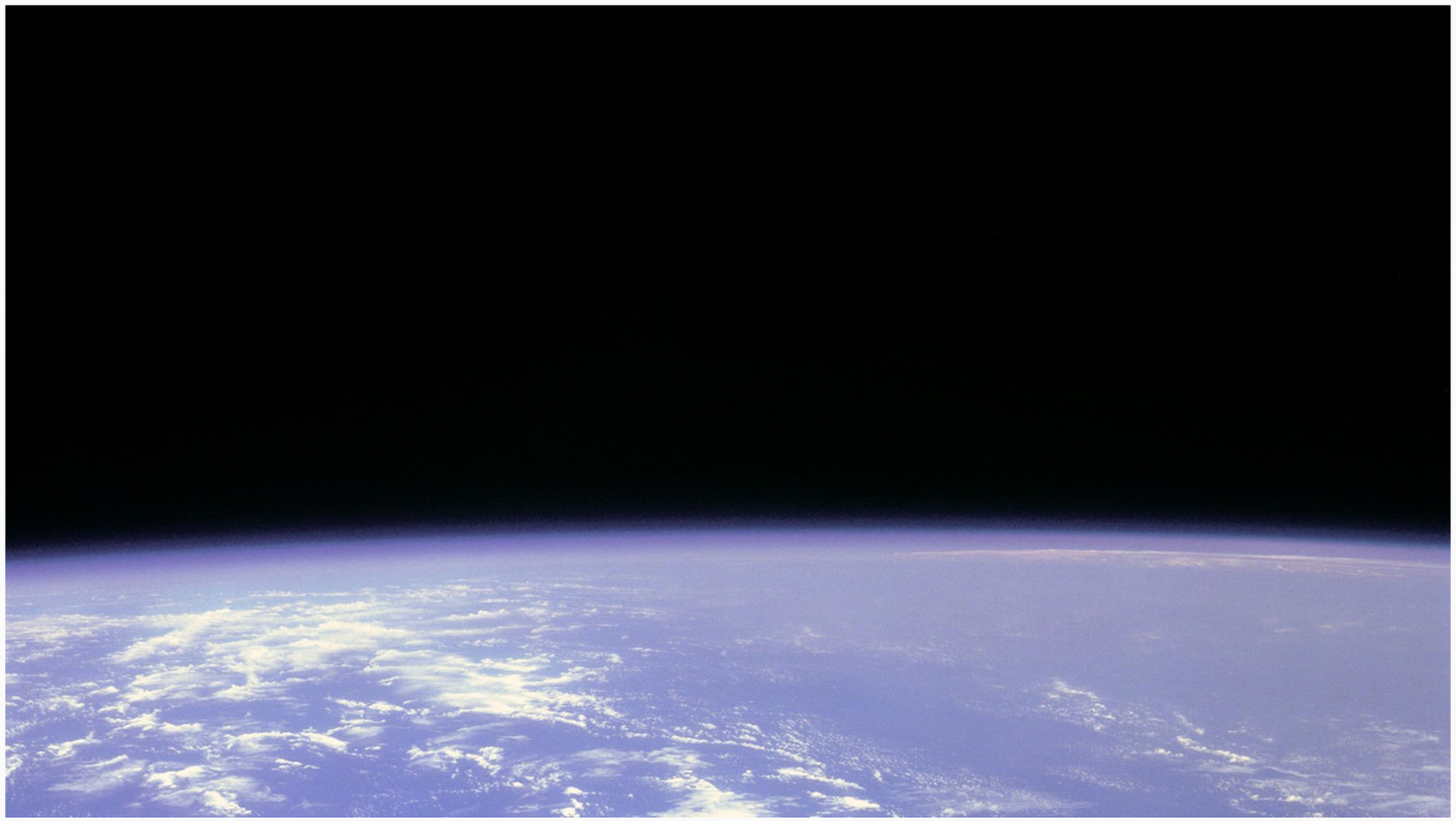
- Q: How can we make an object *as small as possible*, but still preserve what the object is?

Compression

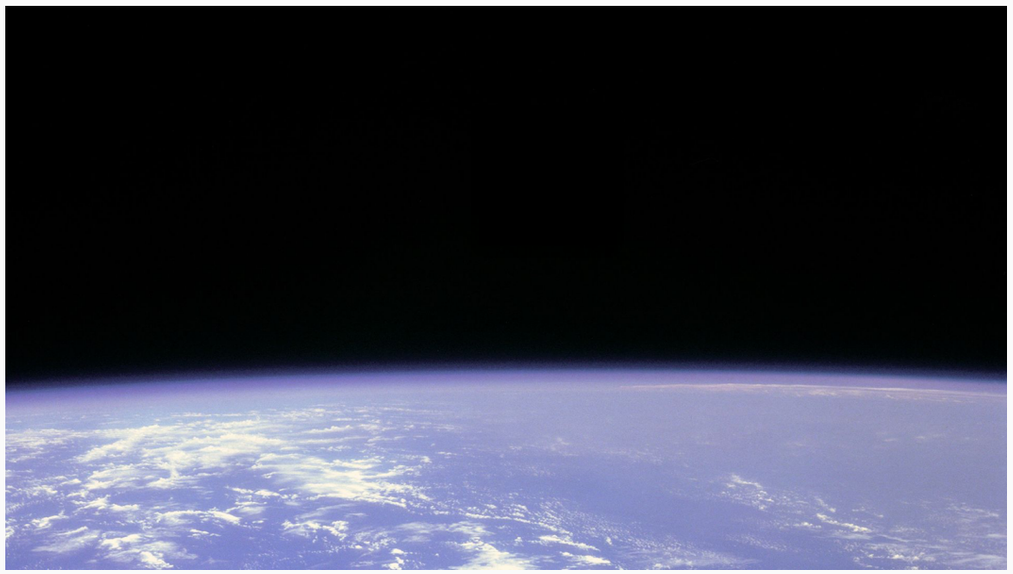
- Q: How can we make an object *as small as possible*, but still preserve what the object is?



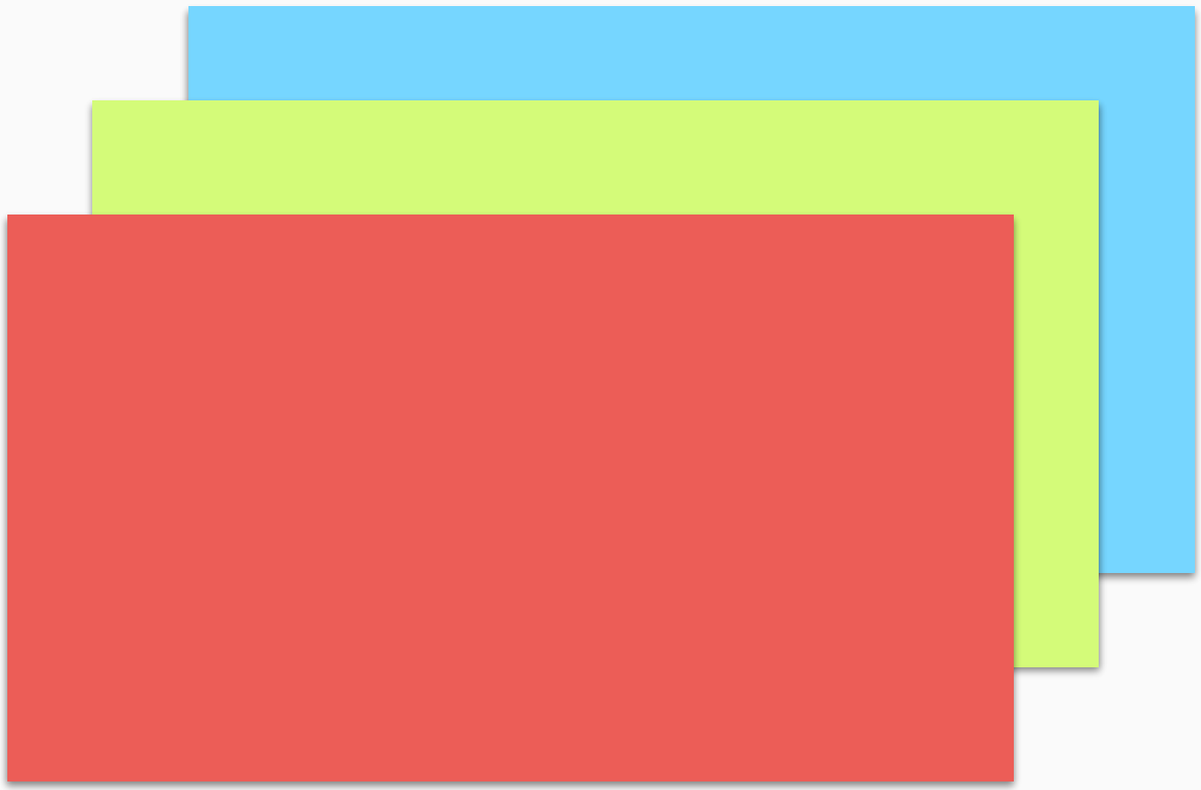
Compression: Intuition



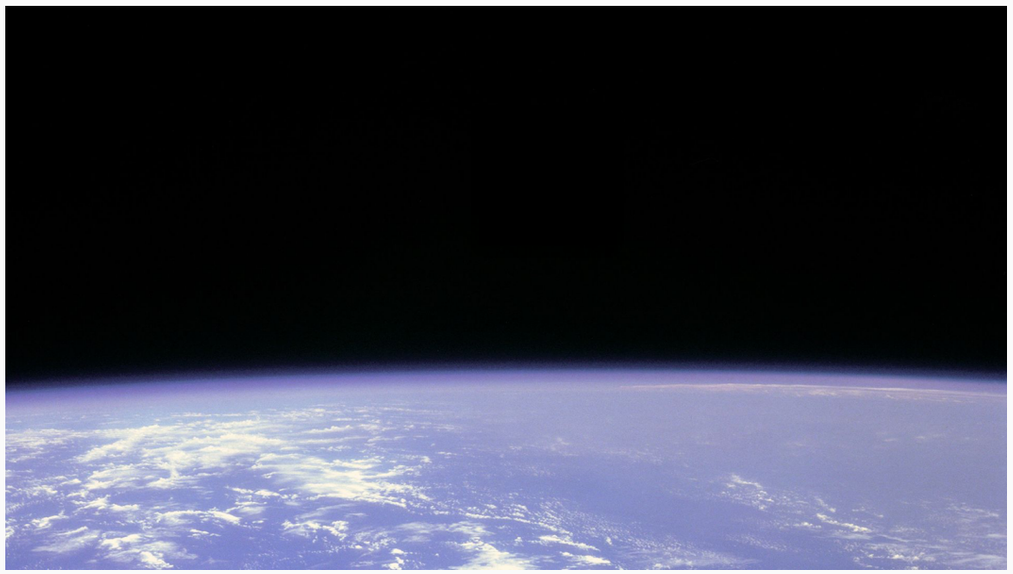
Compression: Intuition



=

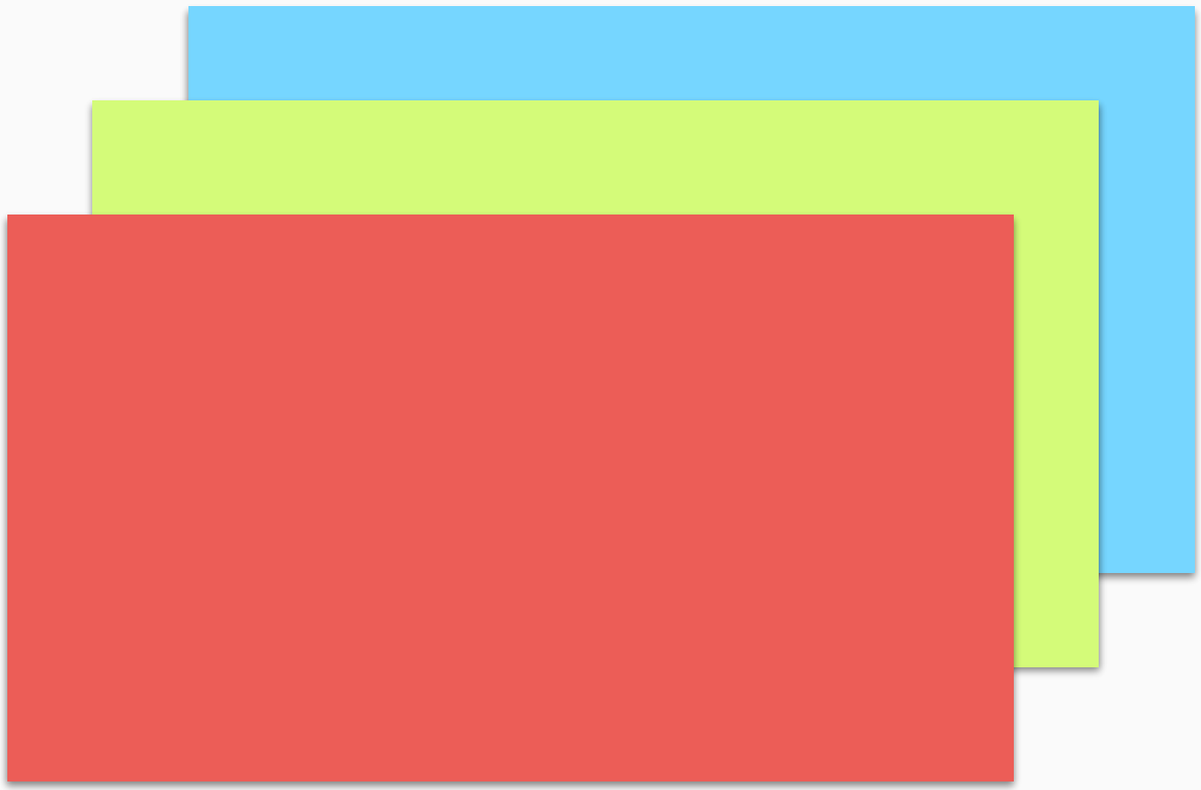


Compression: Intuition

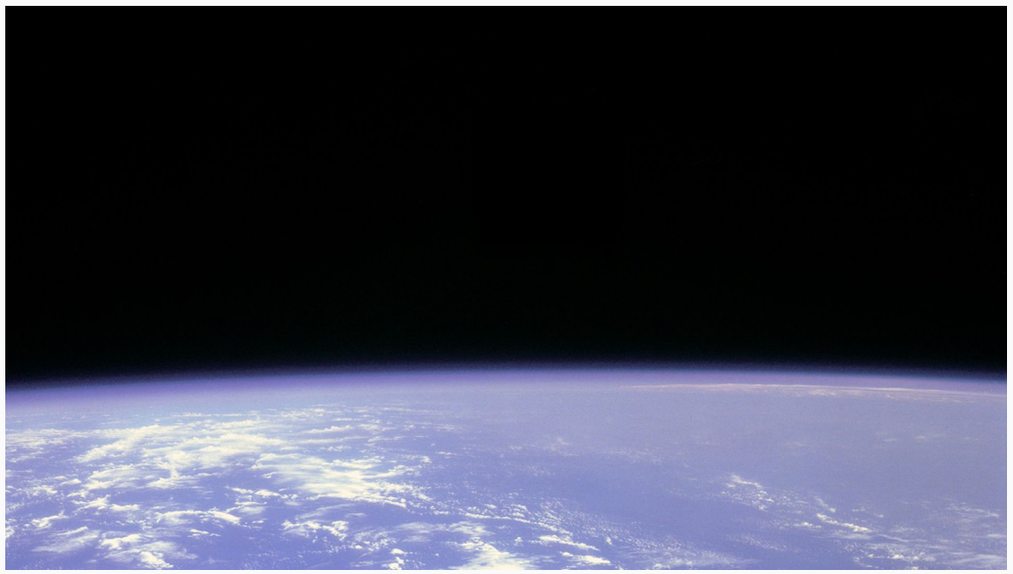


1Kb

=

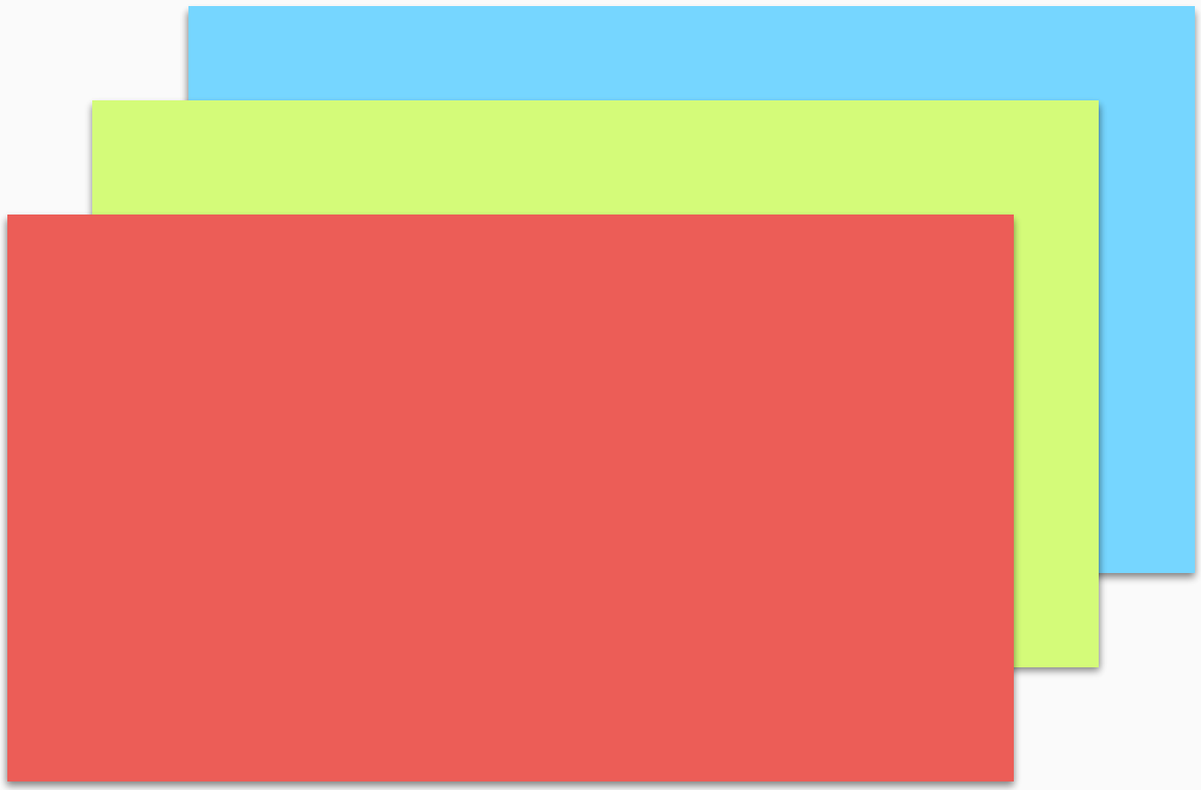


Compression: Intuition



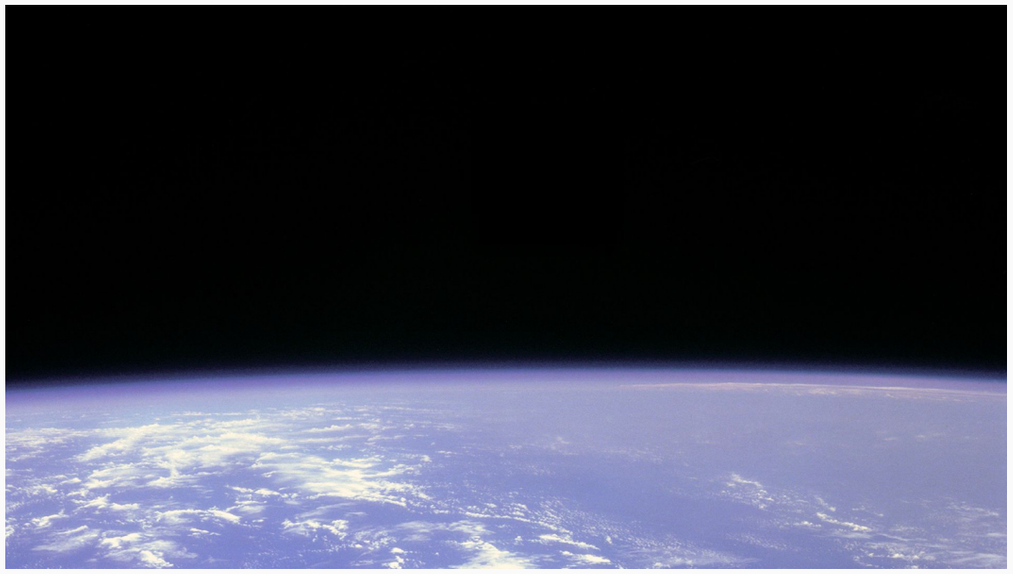
1Kb

=



1Kb

Compression: Intuition



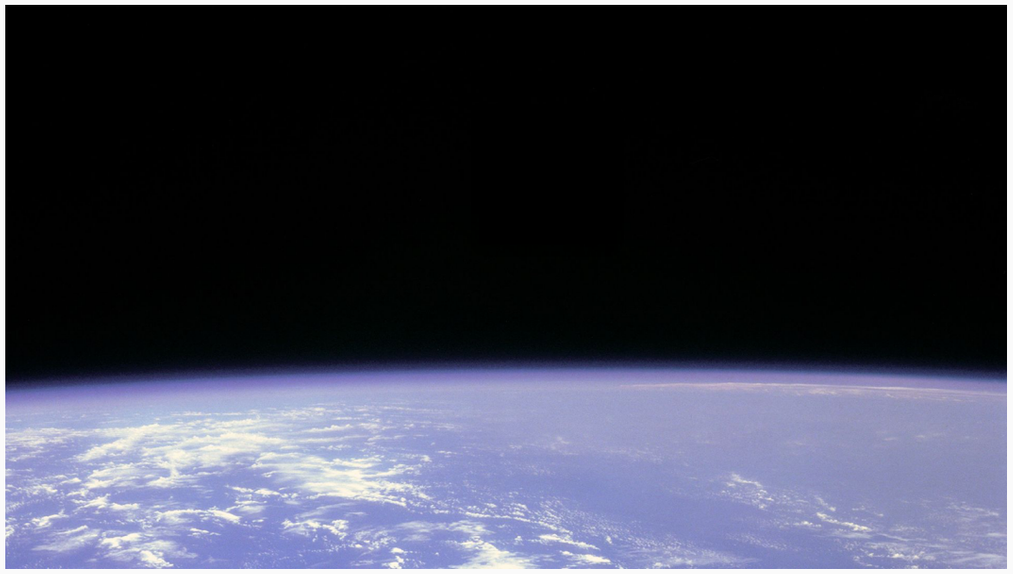
1Kb

=

Everything else, black

0.3Kb

Compression: Intuition



1Kb

=

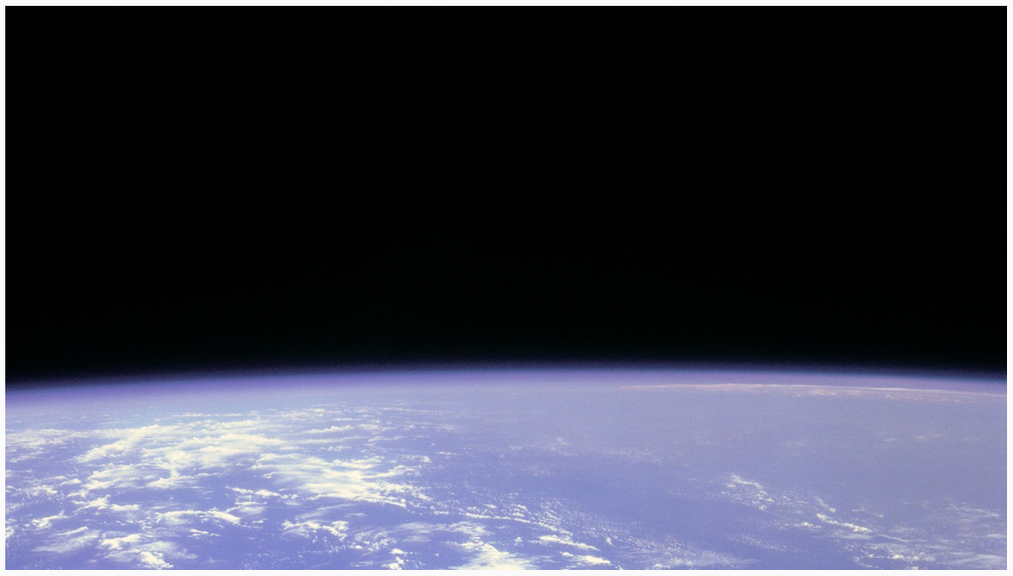
Everything else, black

0.3Kb

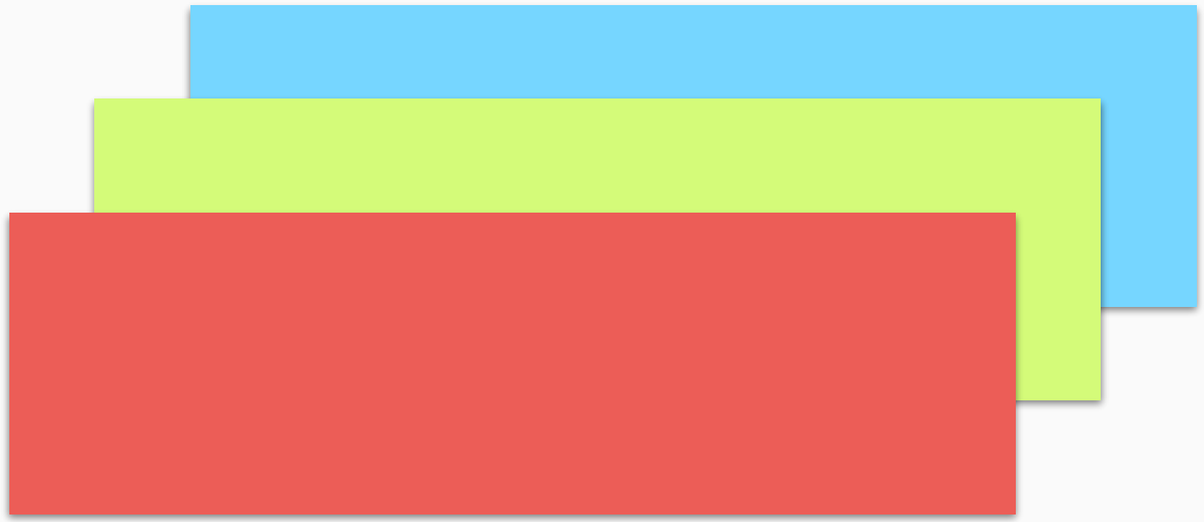
Shaved off .7 Kilobytes!

Compression

Everything else, black



=



1Kb

0.3Kb

Critical: we can do this with everything

Algorithmic Information

Critical: we can do this with everything

For a given object…

Q: What is the shortest algorithm that outputs the object we want to describe?

Algorithmic Information

Q: What is the shortest algorithm that outputs the object we want to describe?

Let’s consider sequences of english characters…

Algorithmic Information

Q: What is the shortest algorithm that outputs the object we want to describe?

Let's consider sequences of english characters…

Example: Scooooooooooooooooooooooby doo!

Algorithmic Information

Q: What is the shortest algorithm that outputs the object we want to describe?

Let's consider sequences of english characters…

Example: Scooooooooooooooooooooooby doo!

Algorithmic Information

Q: What is the shortest algorithm that outputs the object we want to describe?

Let's consider sequences of english characters…

Example: Scooooooooooooooooooooby doo!

Shaggy's algorithm:
Output "Sc", then 20 "o"'s, then "by doo!"

Algorithmic Information

Q: What is the shortest algorithm that outputs the object we want to describe?

Let's consider sequences of english characters…

Example: aaaaaaaaaaaaaaaaaaaaaaaaa

Shaggy's algorithm:
Output 23 "a"

Problem: Compress Words

- INPUT: A phrase in english
- OUTPUT: A compressed version of that phrase.

Algorithm: Run Length Encoding

- Compress repeated sequences into #repeats, then the letter:
  - E.g. “fooooooooooxesssss!”
    - Becomes: 1f11o1x1e5s
  - E.g. “abbbbbaa zabbbaa”
    - Becomes: 1a5b2a 1z1a3b2a

Clicker Question!

Q: What is, “1W5o1d1e1n 1B3i1r1d1s”, when *uncompressed*?

[A] Woooooden Biiirds

[B] Wooden Birds

[C] Wooden Biiirds

[D] Wooooooden BBBirds

Clicker Answer!

Q: What is, “1W5o1d1e1n 1B3i1r1d1s”, when *uncompressed*?

[A] Wooooooden Biiirds

[B] Wooden Birds

[C] Wooden Biiirds

[D] Wooooooden BBBirds

What Do We Think of This?

- Compress repeated sequences into #repeats, then the letter:
  - E.g. “fooooooooooxesssss!”
    - Becomes: 1f11o1x1e5s
  - E.g. “abbbbbaa zabbbaa”
    - Becomes: 1a5b2a 1z1a3b2a

What Do We Think of This?

- Two drawbacks of note:
  - Q: Are repeated sequences of letters that regular in English?
  - Q: What about numbers?

Algorithmic Information

Q: What is the shortest algorithm that outputs the object we want to describe?

Algorithmic Information

Q: What is the shortest algorithm that outputs the object we want to describe?

Q: How *complex* is the object?

Kolmogorov Complexity

Q: How *complex* is the object?

This question ends up being *super fascinating*.

Definition: The length of of this algorithm is called "Kolmogorov Complexity" of the object

Kolmogorov Complexity

Q: How *complex* is the object?

This question ends up being *super fascinating*.

Kolmogorov Complexity

Q: How *complex* is the object?

This question ends up being *super fascinating*.

Q: What objects generally have shorter algorithms?

Q: What objects generally have complex algorithms?

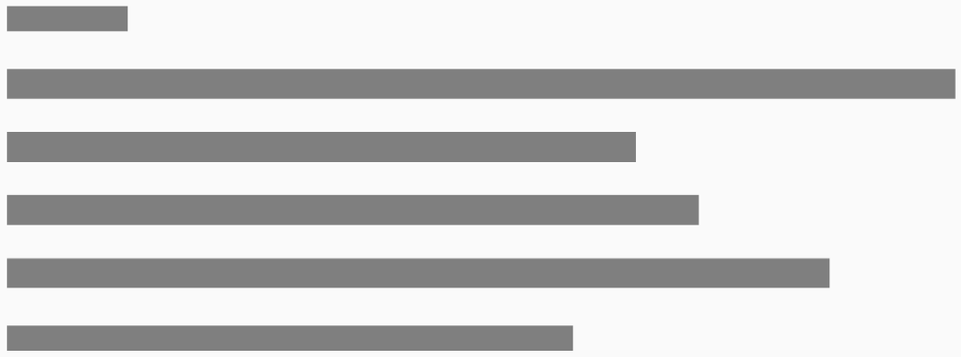
Q: Given an object, how can we *find* the shortest algorithm for describing it?

Kolmogorov Complexity

Q: How *complex* is the object?

This question ends up being *super fascinating*.

Q: What objects generally have more complex algorithms?

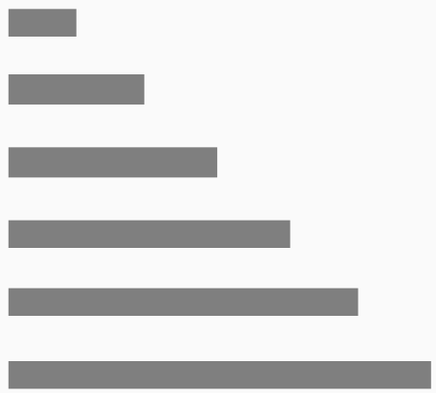


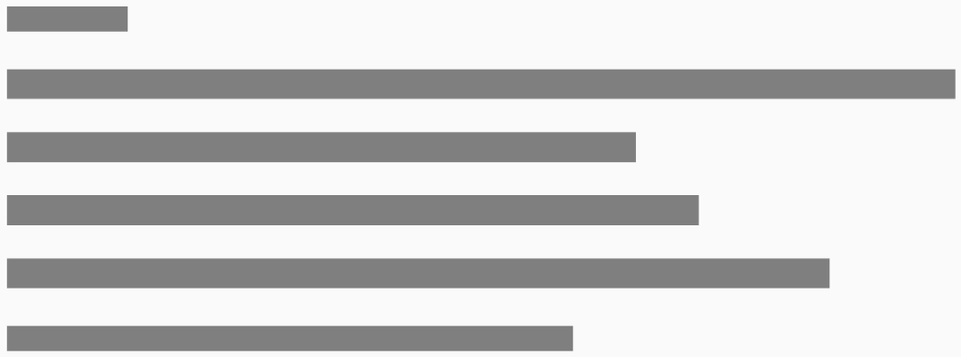
Kolmogorov Complexity

Q: What is the shortest algorithm that outputs the object we want to describe?

This question ends up being *super fascinating*.

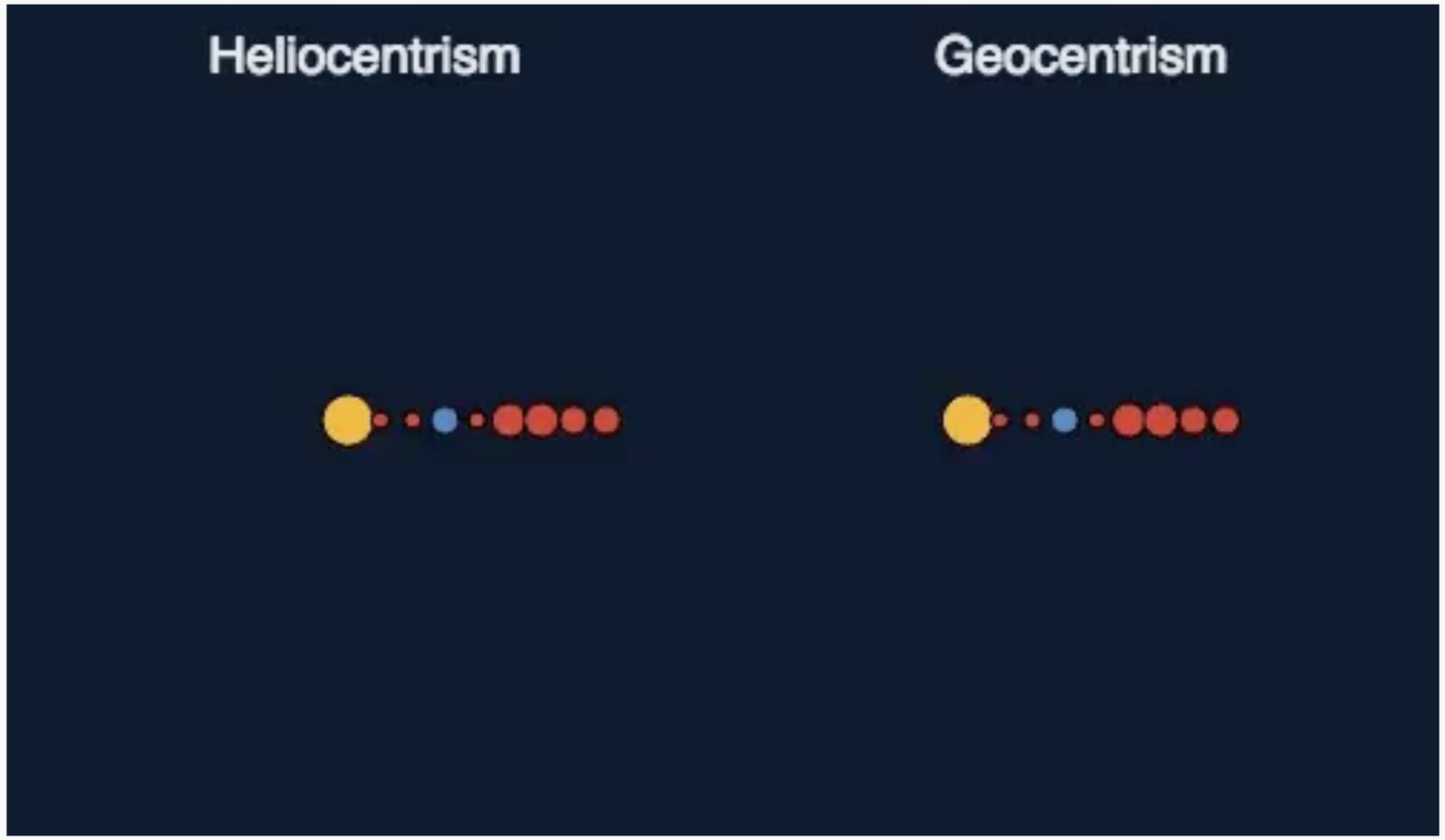
Q: What objects generally have more complex algorithms?





Definition of randomness: *more complicated algorithmic descriptions!*

Kolmogorov Complexity



Definition of randomness: *more complicated algorithmic descriptions!*

Unit 8: Recursion

Dave Abel

April 4th, 2016

Takeaway

Repeated self reference, or “recursion”, is *everywhere*, in the world and in computation! It’s simple, beautiful, and incredibly powerful.

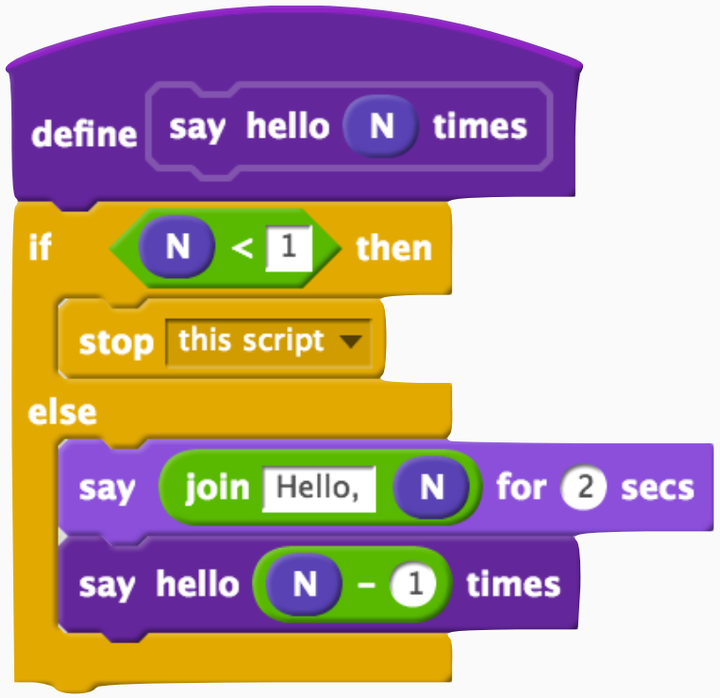
Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference

Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference

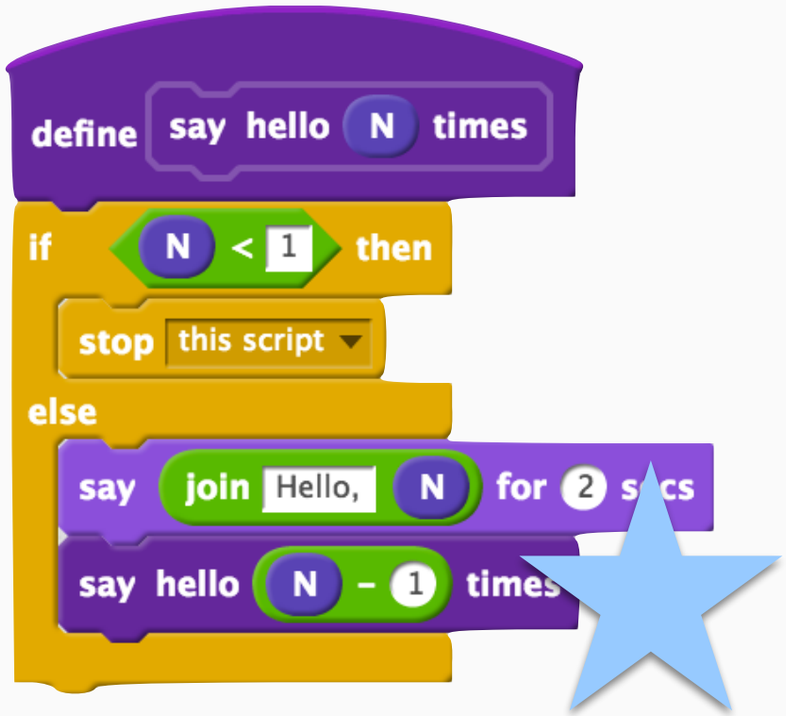
- Example one: **A Scratch block** is recursive if it *calls itself:*



Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference

- Example one: **A Scratch block** is recursive if it *calls itself:*



Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference

- Example two: a tree!

Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference

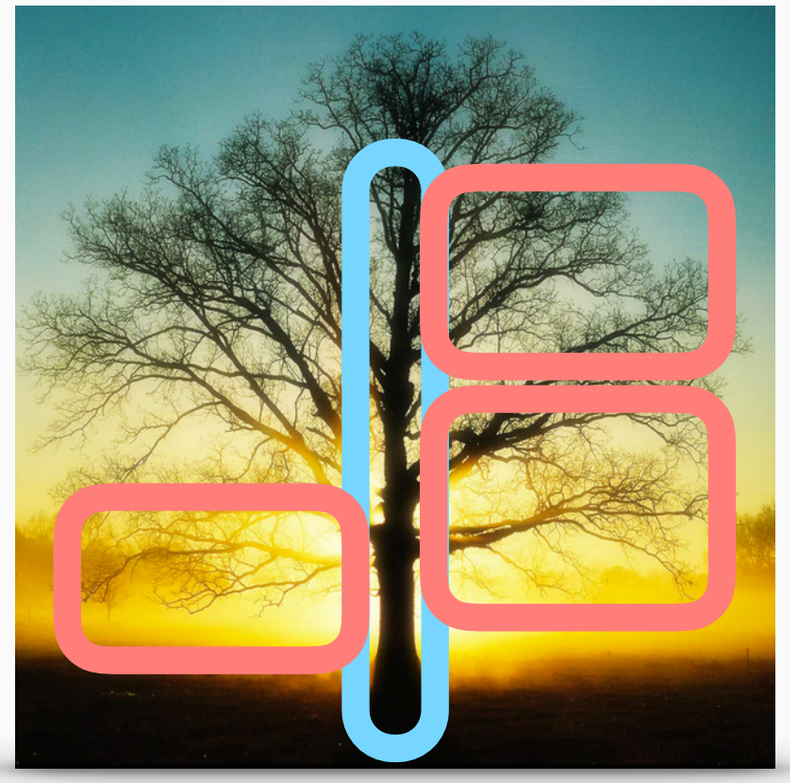
- Example two: a tree!

A tree is: a stick, with some number of trees coming off of it.

Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference

- Example two: a tree!

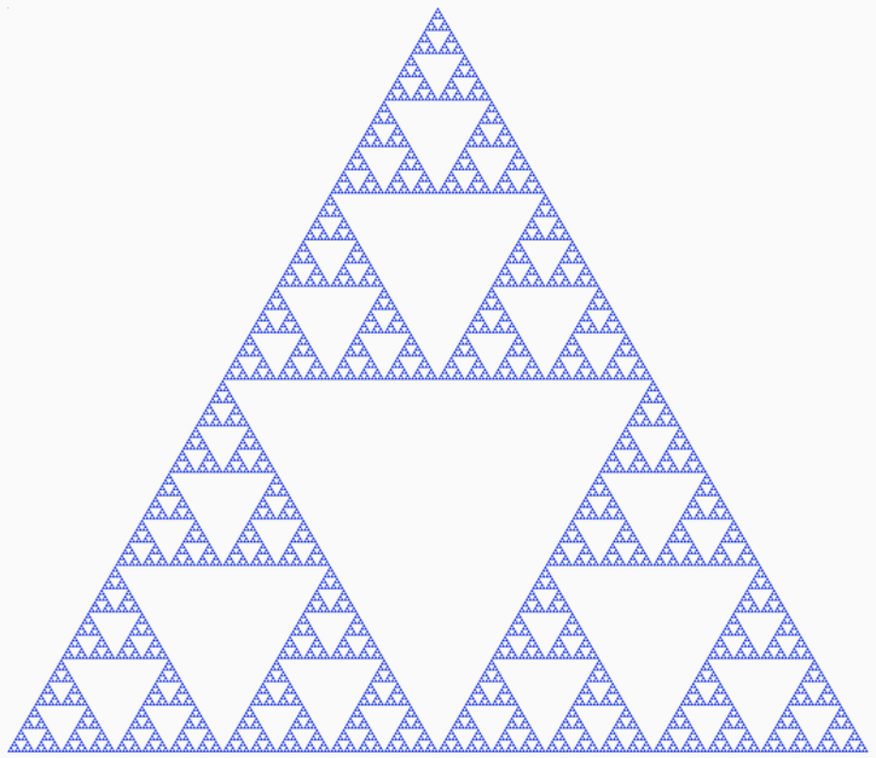


A tree is: a stick, with some number of trees coming off of it.

Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference

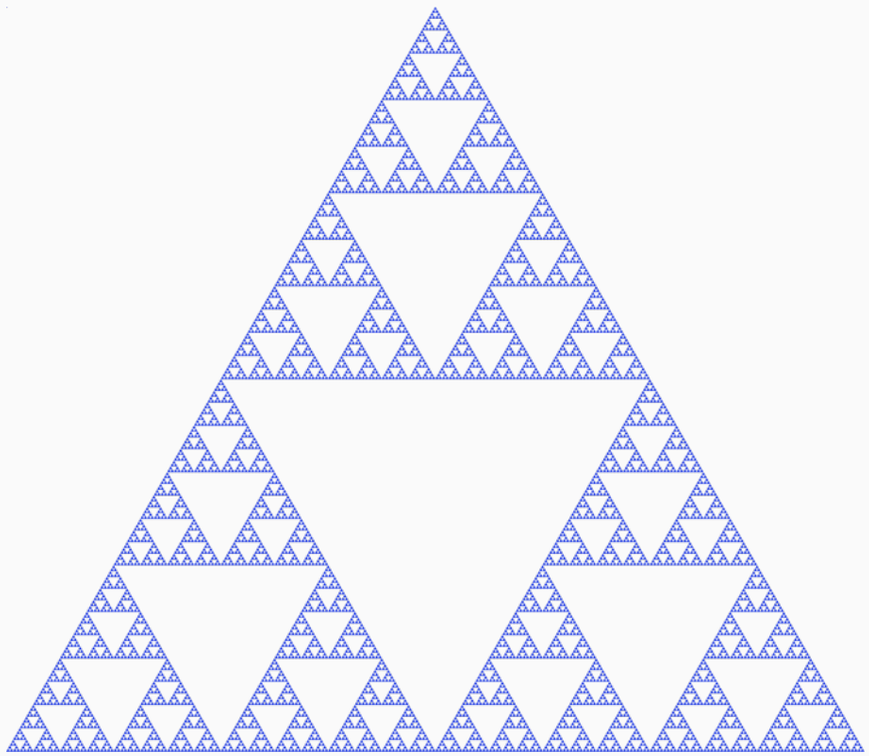
- Example three: **Recursive Shapes!**



Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference

- Example three: **Recursive Shapes!**



A recursive triangle is: a triangle, with a recursive triangle inside of it

Recursion

Discuss with your neighbors and come up with *something* recursive!

Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference

- In general, *recursive entities can be described as:*
  - A simple step
  - A recursive step

Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference

- In general, *recursive entities can be described as:*
  - A simple step
  - A recursive step

Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference

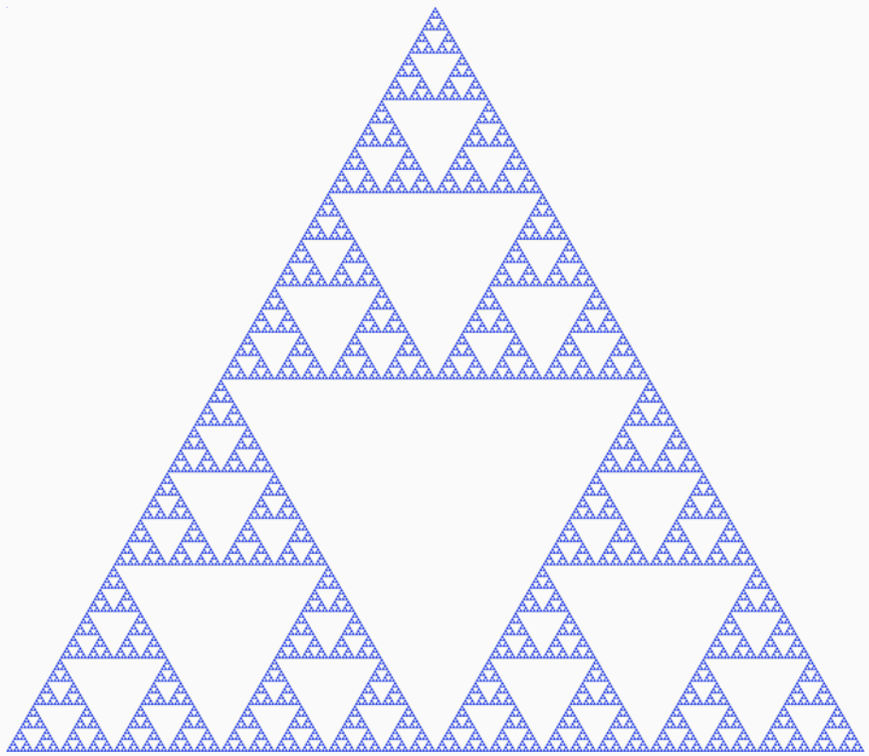
- Example two: a tree!

A tree is: a stick, with some number of trees coming off of it.

Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference

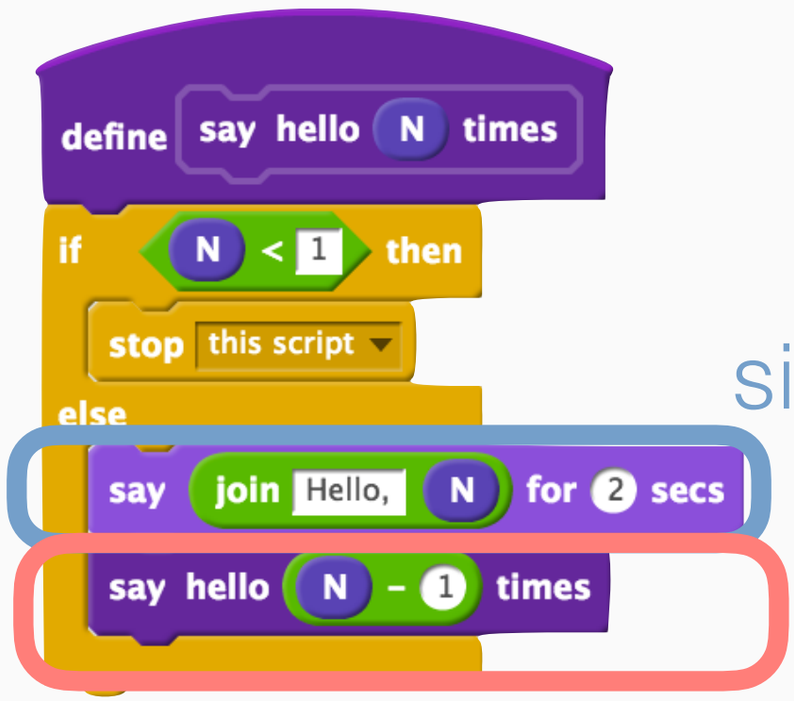
- Example three: **Recursive Squares!**



A recursive triangle is: a triangle, with a recursive triangle inside of it

Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference
- Example one: **A Scratch block** is recursive if it *calls itself*



Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference
- Many algorithms are *recursive!*
- Let's look at a few.

Recursive Algorithms

- Problem: Is a word a palindrome?
  - *INPUT:* a word
  - *OUTPUT:* True if the word is a palindrome, False otherwise.
- Recursive solution:
  - A word is a palindrome if: the outermost two letters are the same AND the remaining word is a palindrome.

Recursive Algorithms

- Problem: Is a word a palindrome?
  - *INPUT:* a word
  - *OUTPUT:* True if the word is a palindrome, False otherwise.
- Recursive solution:
  - A word is a palindrome if: the outermost two letters are the same AND the remaining word is a palindrome.

Recursive Palindrome

- A word is a palindrome if: the outermost two letters are the same AND the remaining word is a palindrome.
  - This basically tells us a solution for solving the problem

Recursive Algorithms

- Problem: Length of a word
  - *INPUT:* A word
  - *OUTPUT:* The length of the word

Brainstorm a recursive solution with your neighbors!

Recursive Algorithms

- Problem: Length of a word
  - *INPUT:* A word
  - *OUTPUT:* The length of the word
- Here's my solution:
  - The length of a word is just 1, plus the length the word you get if you remove one character.

Recursive Algorithms

- Problem: Factorial
  - *INPUT:* A number
  - *OUTPUT:* The *factorial* of that number
  - Example: factorial(4) is 4*3*2*1, factorial(6) is 6*5*4*3*2*1

Brainstorm a recursive solution with your neighbors!

Recursive Algorithms

- Problem: Factorial
  - *INPUT:* A number
  - *OUTPUT:* The *factorial* of that number
- Here's my solution:

Simple step

  - The factorial of a number is just that number times the factorial of one minus that number.

Recursive step

Recursive Algorithms

- One thing that's been swept under the rug!
- Q: When do we *stop* repeating?
- With length of a word, we know to stop at an empty word.
- With factorial, we know to stop because factorial(1) = 1, and nothing after that makes much sense.
- In general, we need to specify *when to terminate*.

Reflection

- Errors happen! So we use codes to make them less destructive to our communicative channels
  - Solution one: repetion
  - Solution two: checksums
- Compression lets us store and send smaller things!
  - Algorithmic Information asks: what is the shortest algorithm that outputs the object we want to describe?
- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference