Self and JavaScript

CS 152 -- Programming Paradigms
San José State University

Michael McThrow
November 23, 2020

Agenda

- Course Announcements
- Overview of Prototype-Based Programming
- Self
- JavaScript
- Question and Answer Session

Announcements

- Project #2 due date has been extended to Wednesday, November 25 at 11:59pm PST.
    - Late submission deadline is Sunday, November 29 at 11:59pm PST.
- I will be catching up on all outstanding grading throughout the week; I'll have all grades except for Project #2 ready by Sunday evening.
    - I will also be posting solutions for Project #1, Lab #3, and Lab #4 this week for finals prep.
    - Project #2 grades will be issued no later than Wednesday, December 2; I will post a solution on Monday.
- Project #3 will be assigned Wednesday; more details to follow.
- Lab #5 will be assigned Monday, November 30. It will cover Rust and SQL.

Schedule for Remainder of Semester

- Monday, November 23 -- Self and JavaScript
- Wednesday, November 25 -- Thanksgiving break: **NO CLASS**
- Monday, November 30 -- Intro to SQL
- Wednesday, December 2 -- Intro to Rust
- Monday, December 7 -- Final Exam Review
- Wednesday, December 9 -- Final Exam

Final Exam Details

- Final exam will be on Wednesday, December 9.
  - Exam will be posted at midnight and must be turned in by 11:59:59pm PST.
- Final exam is worth 20% of your class grade.
- Format is exactly the same as the midterm (take-home, open note and open book [but restricted to the books and papers used in this course], available for 24 hours); there will be both free-response questions and programming segments.
- Final exam is cumulative, but there will be a greater weight on post-midterm topics.
  - All topics excluded from the midterm will remain excluded on the final exam.
- There won't be a practice final exam, but I will share a list of topics no later than December 2.

Prototype-Based Programming

- In traditional object-oriented programming, classes define objects.

```
public class Point {
  private int x, y;

  public Point(int x, int y) {
    self.x = x;
    self.y = y;
  }

  public void add(Point second) {
    x = second.getX();
    y = second.getY();
  }
}
```

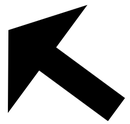
```
Point x = new Point(3,4);
Point y = new Point(5,6);
x.add(y);
```

- Object-oriented programming languages provide inheritance, creating a hierarchy of classes.

```
public class Point {
  private int x, y;

  public Point(int x, int y) {
    self.x = x;
    self.y = y;
  }

  public void add(Point second) {
    x = second.getX();
    y = second.getY();
  }
}
```



GUIPoint inherits Point

```
public class GUIPoint extends Point {
  private Color color;

  public GUIPoint(int x, int y, Color color) {
    super(x, y);
    self.color = color;
  }
}
```

```
Point x = new Point(3,4);
Point y = new Point(5,6);
x.add(y);
```

- A nice advantage of class-based programming languages is that each object has clear definitions regarding its state (e.g., variables) and its behavior (e.g., methods/messages).

- Class-based programming languages can also be useful in type-checking situations in statically-typed programming languages.

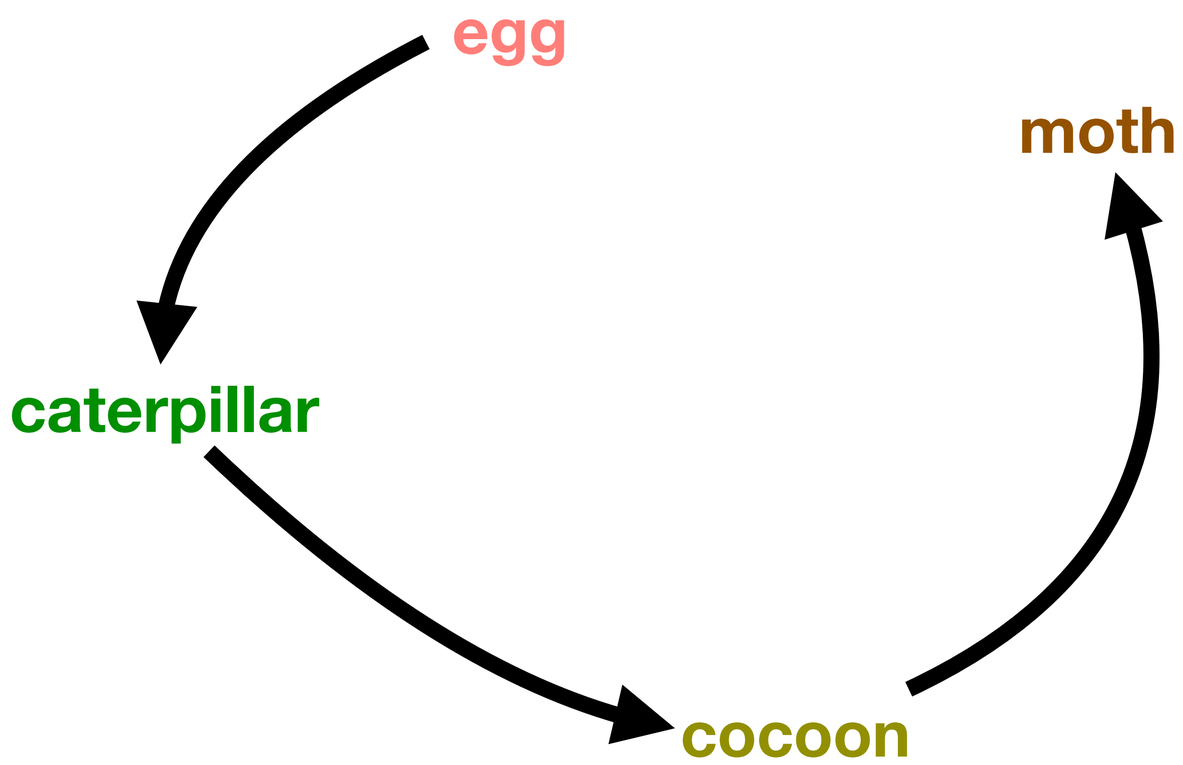
- In an environment where we are describing objects with fixed behavioral characteristics, class-based object-oriented programming seems to be a natural fit.

- Consider modeling situations such as driving a car, or making a transaction at a bank.

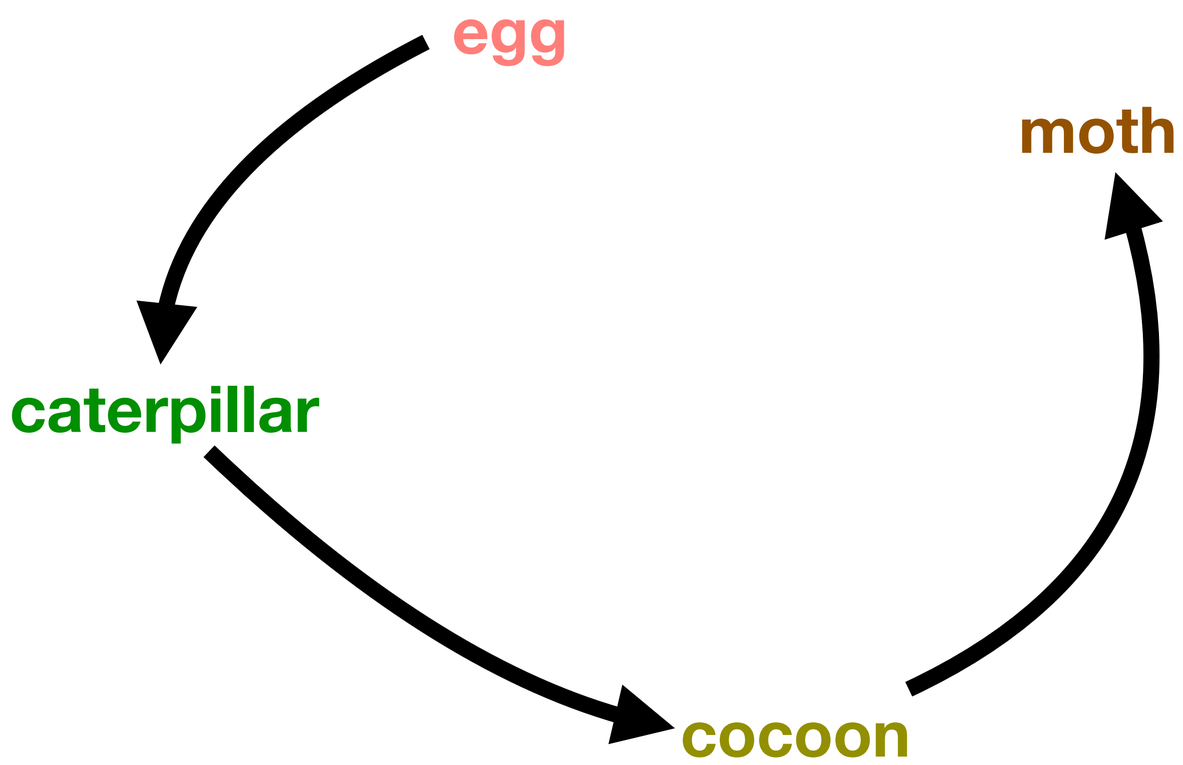
    - Generally, during the lifetime of the object (such a car), its behavior (in terms of the steps needed to carry out a procedure) is most likely not going to change.

However, not all objects behave the same throughout their lifetimes.

Consider the metamorphosis of an insect.



Consider the metamorphosis of an insect.



In a live system where objects have long lifetimes, objects could potentially change their behavior and exhibit new behaviors (or lose old behaviors).

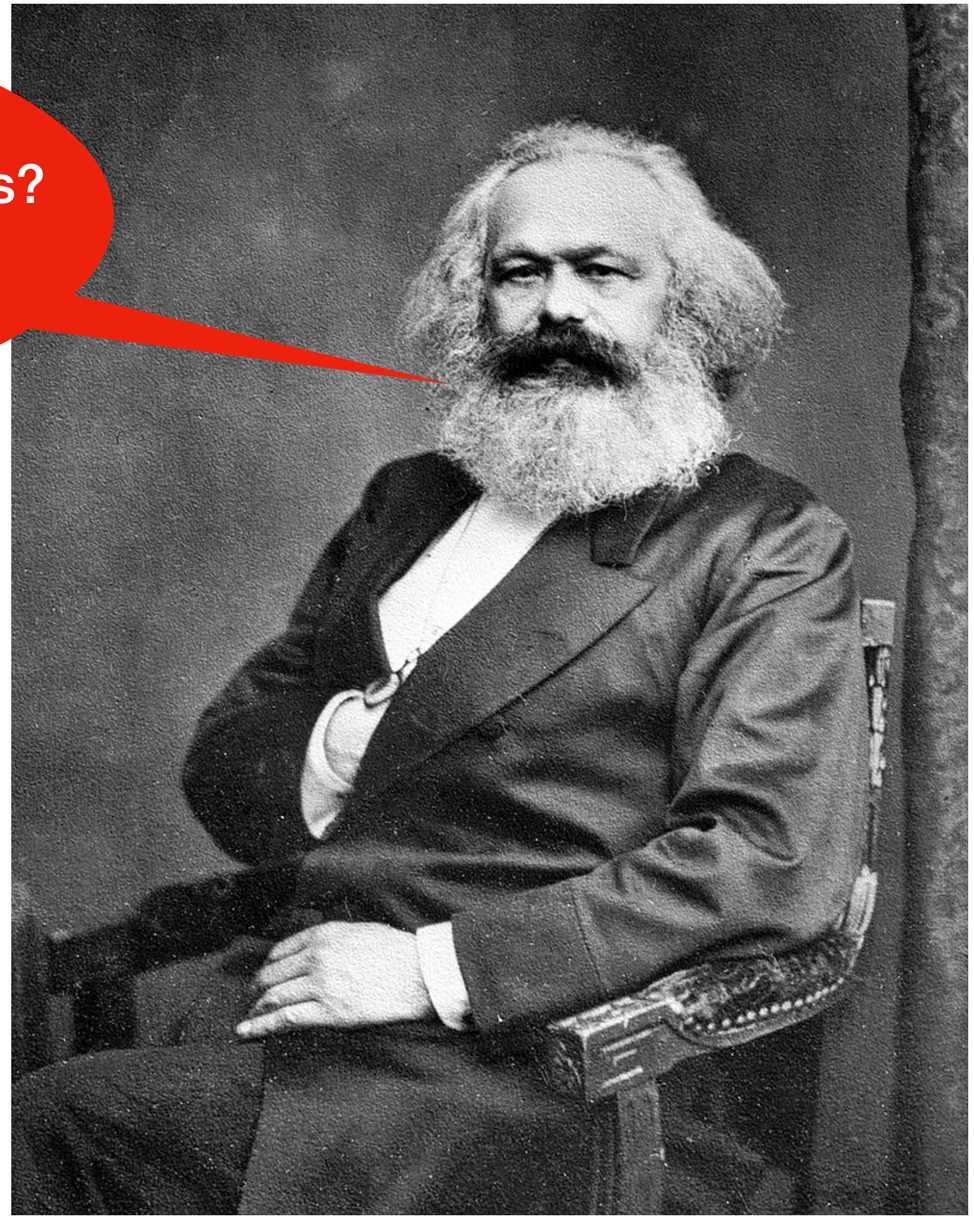
Limitations of Class-Based OO Programming

- In a class-based object-oriented system, how would we deal with new methods added over time?
    - For example, moths can reproduce, but eggs, caterpillars, and cocoons cannot.

One way of dealing with these types of issues is adopting class-less object-oriented programming.

In a sense, class-less object-oriented programming seems to be the logical conclusion of everything being an object, including classes. If you think about it really hard, are classes truly necessary?

Why are objects defined by their classes?
Why have classes to begin with?



Deep Thoughts

by Karl Marx

NOTE: This conversation is historically inaccurate.

It turns out that we don't need classes to do object-oriented programming.

Self

The Self Programming Language

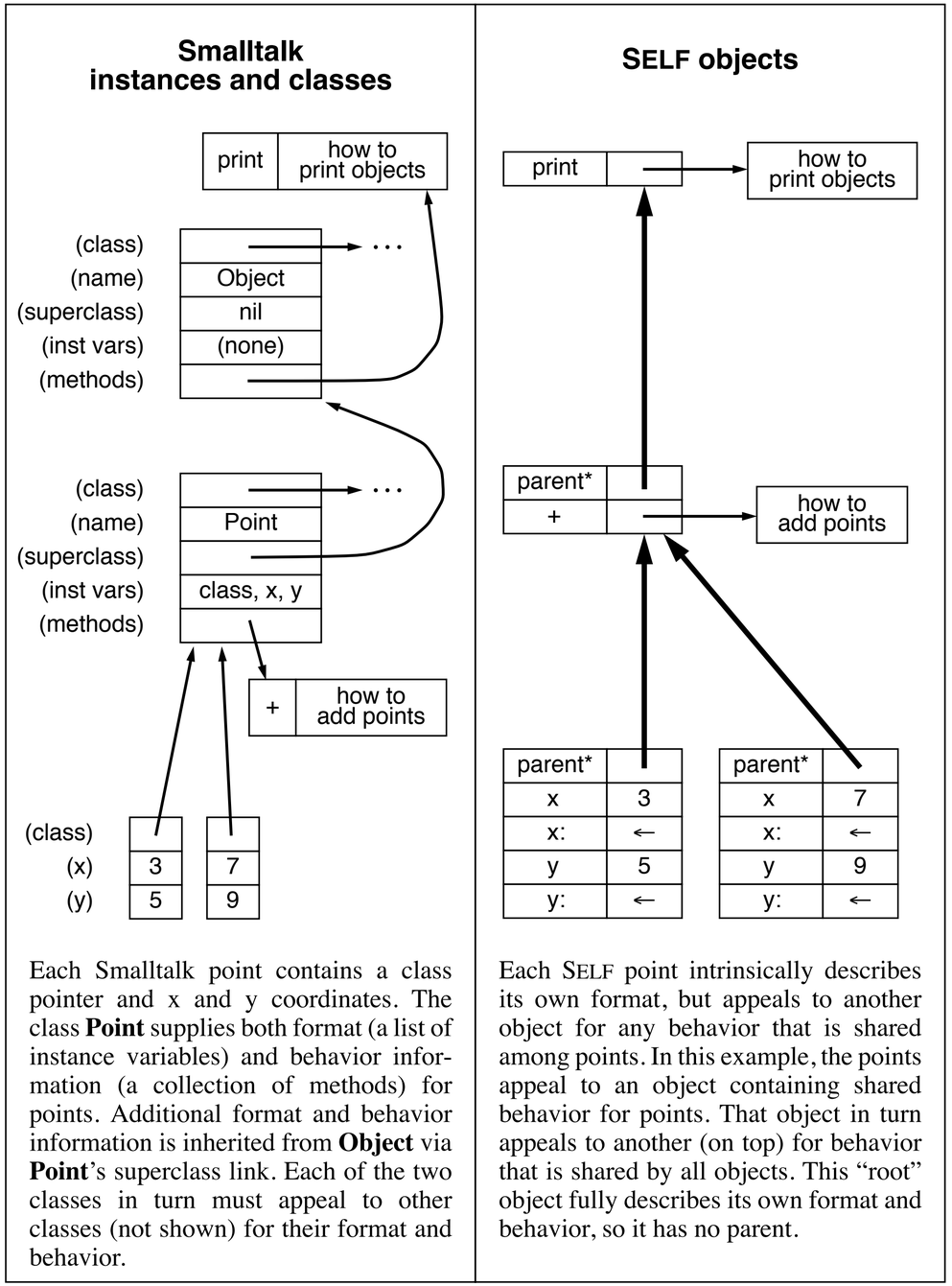
- Created by David Ungar.
  - His PhD thesis, under advisor David Patterson of RISC fame, was on writing a high-performance Smalltalk environment.
  - Self was originally done at Stanford University back when Ungar was a professor, before he left Stanford to go to Sun Microsystems, where he continued working on Self.
- Self is heavily influenced by Smalltalk
  - Syntax is very similar, and everything is an object.
  - Like most traditional Smalltalk implementations, Self is also a self-contained GUI environment.
- Self has no support for classes.

The Self Programming Language

- Objects consists of *slots*.
  - This "slot" terminology is also in the Common Lisp Object System and Dylan (a language developed by Apple in the 1990s that is influenced by Lisp but has a conventional Algol-style syntax).
  - Slots store either state (the equivalent of variables) or behavior (methods).
- There are no variables; instead, if an object wants to maintain state, it sends a message to itself (e.g., `self x: 5` creates a slot named `x` and stores the value `5`).
  - This is how Self got its name.
- Instead of constructing objects from class constructors, in Self we *clone* objects by copying other objects.

Inheritance in Self

- In Smalltalk and other class-based object-oriented programming languages, each object contains a pointer to its class.
- In Self, each object contains a pointer to its parent object.
- To perform inheritance in Self, "[i]f an object receives a message and it has no matching slot, the search continues via a *parent* pointer" [Ungar and Smith 1991, p. 3]

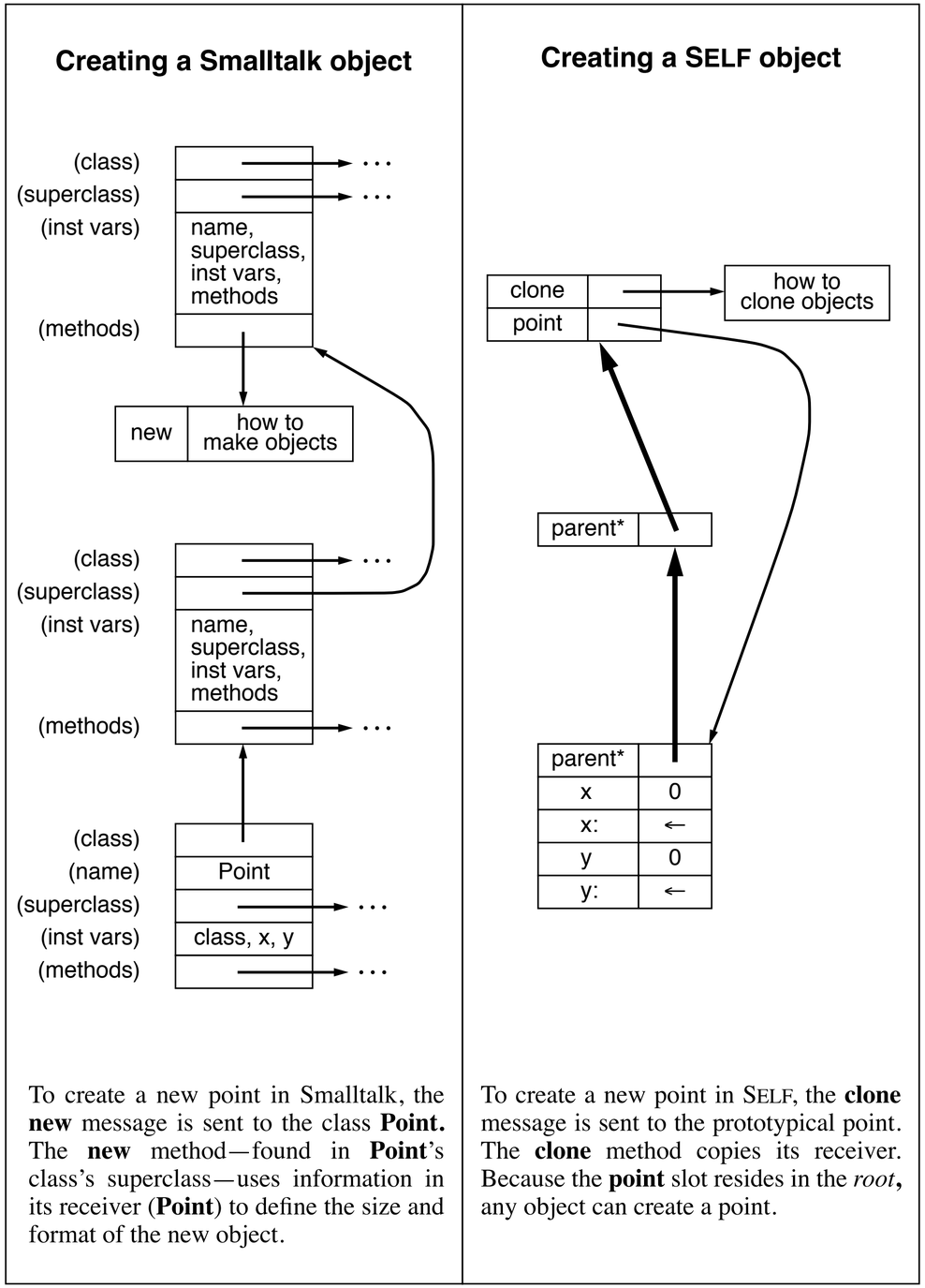


Each Smalltalk point contains a class pointer and x and y coordinates. The class **Point** supplies both format (a list of instance variables) and behavior information (a collection of methods) for points. Additional format and behavior information is inherited from **Object** via **Point**'s superclass link. Each of the two classes in turn must appeal to other classes (not shown) for their format and behavior.

Each SELF point intrinsically describes its own format, but appeals to another object for any behavior that is shared among points. In this example, the points appeal to an object containing shared behavior for points. That object in turn appeals to another (on top) for behavior that is shared by all objects. This "root" object fully describes its own format and behavior, so it has no parent.

Figure 1. A comparison of Smalltalk instances and classes with SELF objects.
At the bottom of each figure are two point objects that have been created by a user program.

Credits: Ungar and Smith 1991, p. 4



To create a new point in Smalltalk, the **new** message is sent to the class **Point.** The **new** method—found in **Point**'s class's superclass—uses information in its receiver (**Point**) to define the size and format of the new object.

To create a new point in SELF, the **clone** message is sent to the prototypical point. The **clone** method copies its receiver. Because the **point** slot resides in the *root*, any object can create a point.

Figure 2. Object creation in Smalltalk and in SELF.

Credits: Ungar and Smith 1991, p. 6

JavaScript: A Lecture and a Demo

Credit: https://www.reddit.com/r/ProgrammerHumor/comments/621qrt/javascript_the_good_parts/

JavaScript is one of the most deployed languages on Earth.

Yet JavaScript is the blunt of many jokes.

What's 2 + "3" in JavaScript?

Begin Rant and Demo

Question and Answer Session