**Q: When is classical inheritance an appropriate choice?**

**A:** This is a trick question. The answer is never. I’ve been issuing this challenge for years, and the only answers I’ve ever heard fall into one of several [common misconceptions](https://medium.com/javascript-scene/common-misconceptions-about-inheritance-in-javascript-d5d9bab29b0a). More frequently, the challenge is met with silence.

**Good to hear:**

* Rarely, almost never, or never.
* “Favor object composition over class inheritance.”

**Red flags:**

* Any other response.
* “React Components” — no, **the pitfalls of class inheritance don’t change just because a new framework comes along** and embraces the *`class`* keyword. Contrary to popular awareness, you don’t need to use *`class`* to use React. This answer reveals a misunderstanding of both *`class`* and React.

prototypal inheritance instead of classical inheritance. This can be puzzling to programmers trained in conventional object-oriented languages like C++ and Java. JavaScript's prototypal inheritance has more expressive power than classical inheritance, as we will see presently.

|  |  |
| --- | --- |
| **Java** | **JavaScript** |
| Strongly-typed | Loosely-typed |
| Static | Dynamic |
| Classical | Prototypal |
| Classes | Functions |
| Constructors | Functions |
| Methods | Functions |

But first, why do we care about inheritance at all? There are primarily two reasons. The first is type convenience. We want the language system to automatically *cast* references of similar classes. Little type-safety is obtained from a type system which requires the routine explicit casting of object references. This is of critical importance in strongly-typed languages, but it is irrelevant in loosely-typed languages like JavaScript, where object references never need casting.

The second reason is code reuse. It is very common to have a quantity of objects all implementing exactly the same methods. Classes make it possible to create them all from a single set of definitions. It is also common to have objects that are similar to some other objects, but differing only in the addition or modification of a small number of methods. Classical inheritance is useful for this but prototypal inheritance is even more useful.