1. What’s the deal with React & TS
   1. Typical browser execution
   2. How (each) changes the game
      1. webpack
   3. Why (each)?
      1. Why anything at all?
         1. SPA tools
         2. Type Safety and a handful of features
      2. Compare to competitors
         1. React: functional style/one-way flow
         2. React: library, not framework
            1. Just “V” (with a little “C”)
         3. React: virtual DOM
         4. React: html in JS rather than js in html
         5. Typescript: less sugar, more safety
2. Overview of React
   1. JSX
      1. Pseudo-html
         1. Superset of html, with some minor tweaks
         2. Custom tags, which are really components that may be many tags
      2. Embedded in js, yet confined to a “render” method
   2. Functional? One way?
      1. No presentation state, just data state
      2. Pass data down to children, including delegates as necessary
      3. Easy to “reason about”;
   3. Virtual DOM (basics)
      1. Re-render entire thing virtually
      2. Diff to previous, change only the differences
         1. Nifty diagram
3. Light overview of Typescript
   1. Superset of javascript; plain javascript will work
   2. It’s all about the types
      1. Static types inferred for everything (defaulting to “any”)
      2. “Compile time” checks
      3. special treatment of null/undefined
   3. Special features
      1. Interfaces
      2. Generics
      3. Enums (including string literal types)
      4. Neat-o type “guards” and similar special-scope stuff
         1. Exhaustive union/enum/string cases
4. Snake!
   1. demo
   2. react component tree
   3. quick look at Snake.tsx