OBJECT ORIENTED VS FUNCTIONAL

OBJECT ORIENTED

- Data *and* logic in discrete objects
- Classes and methods are predefined
- Imperative commands change object state
- For reuse, extend objects (polymorphism)
- For decoupling, use interfaces and code to them
- Instance separate from definition (class)

OBJECT ORIENTED

- Noun-centric
- All or nothing creation
- Relational-object impedance
- Not easily searched in memory
- Not easily composable

FUNCTIONAL

- Functions operate on data
- Does not change state - no side effects
- No pointers or memory references (pass by value)
- Consistency - same inputs yield same results, like math formula
- Composable - output of one function can be the input of another function

FUNCTIONAL

- For decoupling, functions "act" and data is "acted upon"
- For reuse, use library modules
- Execution can happen in parallel or in any order if there's no data dependency between functions
- Code can be hot-swapped

RECAP

OBJECT ORIENTED

- Data and model are objects
- Definition separate from instance
- No partial creation
- Not easily search in memory

FUNCTIONAL

- Data separate from code
- No pointers or memory references
- No side effects
- Output of one function can be input of another