Writing Modular Code

> ***Tip: DRY (Don't Repeat Yourself)***
> *Don't repeat yourself! Modularization allows you to reuse parts of your code. Generalize and consolidate repeated code in functions or loops.*

> ***Tip: Abstract out logic to improve readability***
> *Abstracting out code into a function not only makes it less repetitive, but also improves readability with descriptive function names. Although your code can become more readable when you abstract out logic into functions, it is possible to over-engineer this and have way too many modules, so use your judgement.*

> ***Tip: Minimize the number of entities (functions, classes, modules, etc.)***
> *There are tradeoffs to having function calls instead of inline logic. If you have broken up your code into an unnecessary amount of functions and modules, you'll have to jump around everywhere if you want to view the implementation details for something that may be too small to be worth it. Creating more modules doesn't necessarily result in effective modularization.*

> ***Tip: Functions should do one thing***
> *Each function you write should be focused on doing one thing. If a function is doing multiple things, it becomes more difficult to generalize and reuse. Generally, if there's an "and" in your function name, consider refactoring.*

> ***Tip: Arbitrary variable names can be more effective in certain functions***
>
> *Arbitrary variable names in general functions can actually make the code more readable.*

> ***Tip: Try to use fewer than three arguments per function***
> *Try to use no more than three arguments when possible. This is not a hard rule and there are times it is more appropriate to use many parameters. But in many cases, it's more effective to use fewer arguments. Remember we are modularizing to simplify our code and make it more efficient to work with. If your function has a lot of parameters, you may want to rethink how you are splitting this up.*

NEXT