Chapter 3 Summary: Modules

When creating large programs, it is recommended to design those programs into smaller chunks called modules. This summary will explain the reasons why modules exist and how they are used.

When building programs, it can get overwhelming what the overall program should do. Creating small steps to achieve the goal is what modules are for. A module is a group of statements that exist within a program for the purpose of performing a specific task. There can be many modules in a program that can be executed in the desired order to perform an overall task. It is like cooking by following a recipe for baking a cake. One module can be for beating the eggs, another can be for making the icing; when all the modules have performed their task, you end up with a accurate looking cake.

Using modules allows for simpler, readable code. It allows the programmer to see what changes they need to make to a specific task without affecting the rest of the code. Modules are also great for reusing throughout the program. An example would be if you wanted to make another batch of icing, you wouldn't need to create another program. You could just reuse the module that makes the icing.

To create a seamless program that works together, the modules would need to gain input in order to perform in order. An example would be having a module called beat_eggs, call the next step in the process called add_vanilla. Vanilla will not be executed until the beat_eggs module has completed its task, in which it will then call add_vanilla.

Variables created inside a module are called local variables. This means that their scope is local only to the module itself and not to the rest of the program. If I had a variable inside beat_eggs module called grab_whisk, grab_whisk will not exist in the add_vanilla module unless it's again defined in that module.

More than one variable cannot have the same name. If we declared a variable named grab_whisk and another named grab_whisk, this will cause an error to the program. It makes sense because how will the program know which one to use? However if you created the grab_whisk variable in beat_eggs and then declared it again in the add_vanilla module, that would be ok to do as both of these variables are local to their modules and will not conflict with each other.

Passing an argument to a module is basically passing input into that module. If I have a program that has multiple modules, I would want each module to have a single responsibility. The output from a module can then be passed into another module, where it can repeat the process.