PROGRAM DESIGN

COMP130 – INTRODUCTION TO COMPUTING

DICKINSON COLLEGE

RECALL… WHY FUNCTIONS?

- **Readability:** Programs written using functions are easer for humans to read and understand and thus are also easier for us to maintain.
  - Ex: Reading `print_EAT` vs if it contains all of the individual `print` statements.
- **Maintainability:** Changing the behavior of a function in one place changes that behavior everywhere the function is called, making maintenance less error prone.
  - Ex. Printing all of our letters in X instead of *. Change functions, changes every program that uses them.
- **Problem Solving:** Functions provide a natural mechanism for breaking a big problem down into smaller easier to solve problems, and then putting the solution back together again (see Readability above).
  - Ex. `print_EAT` was a lot easer with `print_E`, `print_A` and `print_T` already done and tested.
- **Reusability:** Functions that are general can be written and tested once and then reused over and over in many programs (e.g. built-in functions and modules).
  - Ex. With all letters in hand, we could print any words we wanted in any program we want.

DESIGN PROCESSES

- Two design processes help to realize the benefits of functions:
  - ***Encapsulation***: the process of identifying and wrapping up a nameable unit of work in a function.
    - Enhances readability and facilitates problem solving.
  - ***Generalization***: the process of making a function reusable in a wider range of applications, typically by adding parameters that modify its behavior.
    - Facilitates reusability and enhances maintainability

REFACTORING

- ***Refactoring*** is the *modification of a working program* to improve its readability, maintainability, reusability or efficiency *without changing its functionality*:
  - Some processes used for refactoring include:
    - Using more meaningful names and better formatting
    - Encapsulating functionality
    - Generalizing functionality
    - Making function interfaces cleaner
    - Factoring out repeated code.
      - Identifying other generalizations that can be reused.
    - Using better algorithms.

INTERFACE DESIGN

- The ***interface*** of a function tells us *what the function does* and *how to use it* (but not how it does what it does). The interface includes:
  - *name, description (doc string), parameters, return value*
- A ***clean interface*** is:
  - ***Clear***: functions and parameters have meaningful names for that communicate their purpose.
  - ***Consistent***: parameter names, types, order and meaning are consistent across functions.
  - ***Simple***: does not require unnecessary or esoteric information.

DEVELOPMENT PLAN/PROCESS

- A ***Development Plan*** is a sequence of steps that can be followed that can help to produce high quality software.
  1. Sketch a small program to start getting an idea of how it will work. Focus on one small part of the larger problem.
  2. Identify a coherent part of that program and **encapsulate** it in a function with a descriptive name and revise.
  3. **Generalize** that function by adding parameters and revise.
  4. Repeat 1-3 until you have a working solution.
  5. Look for opportunities to improve via **refactoring**