# Assignment 2 – Procedural Decomposition and Python 3 Modules

---

## Part One

In this part of the assignment, you are given a text description of a situation that requires a programming solution. You are NOT required to generate any code. Rather, you are required to do a top-down analysis of the problem, decomposing it into a set of modules and descriptions of functions within those modules that, taken together and eventually implemented (in Python 3, for example), would solve the problem. Rather than doing your work in a programming editor, like VS Code or IDLE, you should do it in a word processing program, like MS Word, Google Docs, or LibreOffice Writer, and save it as a .docx file called **a2_p1.docx**.

Your farming friends, Kelly and Rudy, have set up a road-side vegetable (veggie) stand to sell their own and other local farmers' produce.

The other farmers drop their produce off at the veggie stand and leave it for sale on consignment, meaning no money exchanges hands until after the produce is sold. To keep track of which produce came from whom, Kelly and Rudy decide to computerize their operation somewhat and they have turned to their programmer friend, you, for help.

Your immediate task is to do a decomposition that will meet with Kelly's and Rudy's approval before you start writing executable code. Here are some things you, Kelly, and Rudy have agreed upon going into the process:

1. To keep things simple, the program and data files will be stored on Rudy's laptop. Backup will be done by drag-and-drop file copying onto flash/USB drives and will not be part of the design. The data files will be text, but the structure of those files will be left up to you. The data files and your program will constitute a rudimentary database management system.
2. The information to be stored for each supplier will be a unique ID number (one that is mostly used internally by the software) and the supplier's name and telephone number. For many suppliers, this information will only ever have to be entered once, but the system should allow edits and occasional deletions.
3. The information to be stored for each transaction between a supplier and the veggie stand owners will be the supplier's ID number (thereby allowing a transaction to be associated with a supplier), the date produce was received from that supplier, a description of the produce (for example, "field tomatoes"), the weight of the produce (as measured on scales borrowed from Kelly's kitchen), an agreed-upon price per kilogram, the date on which the last of the sellable produce was sold, and the weight of any waste owing to spoilage (Each of the suppliers has agreed not to charge Kelly and Rudy for spoiled produce.)
4. For reporting and invoicing, the system must allow Kelly and Rudy to enter a date, choose a supplier from a list of suppliers, and then get a listing of all transactions with that supplier, and amounts owing to the supplier, from the entered date forward.

At the conclusion of the design process, produce a set of modules with meaningful names and descriptions and, within those modules, sets of one or more function prototypes (each consisting of the name of the function, a list of its parameters, descriptions of the purpose of the function and the purpose of each of the function's parameters, and the value or values – if any – returned by the function) for presentation to Kelly and Rudy. (You may assume that they know enough about programming to understand what a function is.)

This is the sort of information we are looking for here, but for a different project, obviously:

Requirement: A text-based software system to build and maintain a memberships database for a small community organization.

**Module: memberships**

Memberships database interface.

Functions:

**show_members(status=all)**
Print a list of information from the membership database. where status is one of:

- all (default) – All members, past and current
- regular – Current members
- past – Past members
- life – Life members
- honourary – Honourary members

**add_member(status=current, name, address, email, phone, expiry)**
Append a new membership to the membership database.
Parameters:

- status (default current) is one of those shown for show_members(), above.
- name, address, email, and phone are strings.
- expiry is the date on which the membership expires (required only for regular, past, and honourary members).

Return a new, unique member id if the operation succeeds, None otherwise.

**edit_member(member_id, status, name, address, email, phone, expiry)**
Update the record of member member_id.
Parameters:

- status is one of those shown for show_members(), above.
- name, address, email, and phone are strings.
- expiry is the date on which the membership expires (required only for regular, past, and honourary members).

Returned value: True if the operation succeeds, False otherwise.

**… You'd be expected add extra functions here as required by the specifications.**

**Module: menu**

Facilitate the presentation of text-based menus and processing of user choices via the keyboard.

Function:

> **do_menu(title, choices)**
> Print a text menu, return an int representing the user's choice or None if the user enters "x" or "X" to exit.

**Module: main**

Imports menu, membership.
Primary user interface and program control.

Function:

> **main()**
> Looping, menu-driven interface allowing controlled access to the memberships database by way of functions imported from memberships.

---

## Part Two

**Note: This part of the assignment is unrelated to Part One, mostly, although both are exercises in modularity.**

The purpose of this part of the assignment is to give you some practice in creating Python 3 functions organized by category into several modules and united by another module that controls a text-based (that is, non-graphical) user interface.

**What you are given**

In the provided Assignment 2.zip file, we have given you some working code to get you started and for you to emulate and learn from. In particular:

- **main.py**: This module is the starting point for execution. With the assistance of various other imported modules, it provides the text-based user interface for the program and controls all keyboard input and screen output. In completing this assignment, you will make additions to this file.
- **menu.py**: Do not make changes to this file. It is used to provide a text-based menu system for main.py. If run by itself, it will execute some test code. You should study the code in this file, including the testing code in the **`if __name__ == "__main__":`** section at the bottom. Testing done in this manner will be covered in Unit 2.

- **get_number.py**: Do not make changes to this file. It is used to provide a fail-safe way of getting float or integer values from a user via the keyboard. If run by itself, it will execute some test code. You should study the code in this file, too. Some of what's there may not be covered in the course in time for you to submit this assignment, but don't worry: Unless we cover it in lecture, you won't be expected to know that stuff to complete the assignment or to do Quiz 2.
- **temperatures.py**: This module already contains functions that perform temperature conversions between the Fahrenheit scale used in the US, and the Celsius scale used in Canada. It is meant to give you a model to emulate in other modules. You will be asked to make additions to this module.

**What you must do**

1. Modify the temperatures.py module by adding the following functions.
    - `cels_to_kelv(deg_c)`: Return `deg_c` degrees Celsius in kelvin.
    - `kelv_to_cels(kelv)`: Return `kelv` kelvin in degrees Celsius.
    - `fahr_to_kelv(deg_f)`: Return `deg_f` degrees Fahrenheit in Kelvin.
    - `kelv_to_fahr(kelv)`: Return `kelv` kelvin in degrees Fahrenheit.

    Note:
    - You will have to research how to do the kelvin conversions if you don't already know. (The same applies to conversions in the other modules you're required to build.)
    - As is the case with the functions you are given, your functions must have single parameters of type float (floating-point numbers).
    - All functions should return single values of type float.
    - **None of the functions you are to create for this module may perform input or output, so no use of input() or print() is permitted. All data must enter a function by way of its parameter(s) and leave it by way of a return statement.**
    - Extend the self-testing section of the module (under the **`if __name__ == "__main__":`**) to test your new functions with easily confirmed values.
    - Update the docstring at the top of the module by adding your name as a contributor and the date you completed your contributions.
2. Create the following module and self-testing code for them. **The module files must be saved to the same folder as converters.py, menu.py, get_number.py, and temperatures.py**.
    - **distances.py**: This module must contain the following functions and adequate testing with easily confirmed values for them:
        - `miles_to_kiloms(mi)`: Return `mi` miles in kilometres.
        - `kiloms_to_miles(km)`: Return `km` kilometres in miles.
        - `feet_to_metres(ft)`: Return `ft` feet in metres.
        - `metres_to_feet(m)`: Return `m` metres in feet.

- `inches_to_centims(ins)`: Return `ins` inches in centimetres. (Note that we avoid using the usual abbreviation for inches, "in", as `in` is a keyword in Python 3.)
- `centims_to_inches(cm)`: Return `cm` centimetres in inches.

Note:

- **None of the functions you are being asked to create may perform input or output, so no use of `input()` or `print()` is permitted in your work. All data must enter a function by way of its parameter(s) and leave it by way of a return statement.**
- All the functions you are being asked to write have single parameters of type float (floating point numbers) and return values of type float.
- All modules should begin with a correctly formatted and descriptive docstring and include your name as its author and the date of completion.
- All functions should have correctly formatted and descriptive docstrings. As the functions will be very brief, you may not require any inline (#) comments.
- The testing code for each module, except for converters.py, must be placed at the bottom of the module as the block of code for an **`if __name__ == "__main__":`** statement, as was done for each of the provided import modules.
- Run each module as a standalone program and check that it produces correct output for your test cases. If it doesn't, fix it before continuing.

3. Make additions as needed to the main.py module to incorporate the additional functions in temperatures.py and the functions in your new module. Test main.py as you complete each small section to make sure it is still working correctly.

---

**What to Submit**

Submit, to onQ, the following files **combined into a SINGLE zip file** named **a2_#######.zip,** where the #s are replaced by your 9-digit student number:

1. a2_p1.docx (your solution to Part One)
2. main.py (as modified by you)
3. menu.py (as provided)
4. get_number.py (as provided)
5. temperatures.py (as modified by you)
6. distances.py (as created by you)