==
Modules
==

In the previous notes shared, the idea of breaking down a program into modules (procedures and functions) helps to better structure a program and to also allow us, programmers, the opportunity of better managing the processing of information for the solving of the problem.

Some of the notes that follow is a repetition of discussions already had in class, but it should be noted that we will cover all these points in future programs.

==
The Rules for Designing Modules

1. Each module is an entity in itself. There is a single-entry point and a single exit point.
(*Processing of the code does not jump to a middle point within a module*)

2. Each module has a single function, such as calculating, entering data, or printing answers.

3. Each module is short enough to be easily read and modified.

4. The length of a module is decided upon based on its purpose and needs of the solution and the number of instructions needed to perform the function.

5. A module is developed to control the order of processing.

==
Types of Modules

The following is the most commonly used types of modules.
(*These are different compared to the words, houseKeeping(), detailLoop(), & endOfJob(), that are referred to in our current textbook*.)

Names of the modules would differ from program to program, but it is the type that helps us learn and understand the purpose of the module

- **Control Module**
    - -- Controls the overall flow of the program.
    - -- All other modules are called within it.
- **Initialization Module**
    - -- Processes instructions only once during the execution of a program and only at the beginning. Examples include, opening files, initialize starting values of variables
- **Process Module**
    - -- These types of modules may be processed once, or if they are a part of a loop, then they would be processed many times during the execution of a program.

    - -- There are different types of <u>Process Modules</u>, namely:
    - -- **Calculation Module**
        - Perform calculations
        - Accumulating
        - Incrementing
        - Manipulating inputted data in some form so that it is presented as required for the output
        - Again, not only are you dealing with numerical values, it should also be noted that this type of module could deal with the manipulation of strings, characters and other types of variables.
    - -- **Print Module**
        - Print the results of processing, including headings and summaries
    - -- **Read and Data Validation Module**
        - Read and input data
        - Validate the information entered (print errors and request re-entering if needed) (We may consider validation as being a separate module being called within the read module, but that would be the choice of the programmer)
- **Wrap-up Module**
    - -- Process instructions that are executed once, at the end of the program.
    - -- Close files, print totals, etc.

The above is only a basic list to begin our understanding of the development of program. More information will be shared as we progress with lessons.

==
Local and Global Variables

The opportunity of having better control over the declaration of variables is valued by a programmer. But in saying this, we learn that by having control over when and where a variable is declared, also decides on the scope of the variable (*where it is accessible and by whom*)

Local variables can be used by the module itself. Other modules will have no knowledge of these declared variables. This can also lead to variables having the same name, but technically are different, because they have been declared and used by different procedures and functions.

Global variables on the other hand are variables that are accessible by all modules in a program including the main control module as well. Knowing that these types of variables are accessible by all the procedures and functions in a program, then the duplication of variable names are not allowed and can cause confusion, if multiple programmers are developing separate pieces and when the time comes to bring together these various pieces, the program will not perform, *run-time & logical errors are likely to occur*.

** At the moment in a majority of the programs we develop, our declarations happen once at the beginning of the main program and therefore considered to be a global variable.

** If and when we decide for a module to declare its own variables, then that is considered to be local to the module

** In the future, in the **real programming languages**, we will have the opportunity to declare not only at the beginning of the main program or at the beginning of a module, but basically anywhere in the program, just before we are required to use the variable. This concept will change the scope of variable even further and access will need to be better understood.

==
Parameters

When the time comes to develop both procedures and functions, we will learn that an opportunity arises to be able to pass values to a module. These values, parameters, are local variables within a module. Their declaration is done in the module header.

There are different types of parameters and this can affect how the modules communicate and pass information inside one another.

In the future lessons, we will learn more about them.

==
Return Values

In previous notes, we have become aware of functions that are already coded and accessible within a programming language, SQRT (square root), RND (random), AVG (average), are just some of them. In order to retrieve an answer from an accessed function, we need to pass a value to the function, SQRT(16), 16 being the value passed. The function will accept the value and perform its required coding and then "return" back an answer.

When a function is used as part of an equation:

```
        intAnswer = 10 + SQRT (16)
```

using the order of precedence, the function is executed first and the returned result, in this case, 4, is then used for the next part (the addition), which is, 10 + 4. Finally the result of the addition is then assigned to the variable, intAnswer.

We may also consider the following:

```
        intSquareRoot = SQRT (25)
```

and remembering how we read an assignment statement, the variable on the left, intSquareRoot is assigned the value of the result of the expression on the right-hand side. In this case, the function is executed and its result, once returned, is assigned to the variable.

Another lesson, soon to be learnt, is to design our own functions, and when we do, we are required to code a return statement in the syntax of a function

Please take note of the difference between the pseudocode and the Java programming language

The following example is a function to calculate the area of a rectangle, the function has declared 2 parameters (this would mean that when the function is called, 2 values need to be passed to the function)

** pseudocode

```
0. calcAreaOfRect (num intSide1, num intSide2)
1. Declarations
        num intAnswer

2.      intAnswer = intSide1 * intSide2

3. return intAnswer
```

** Java

```
public static int calcAreaOfRect (int intSide1, int intSide2) {
        int intAnswer;

        intAnswer = intSide1 * intSide2;

        return intAnswer;
}
```

// Calling the function in another part of the program – both the pseudocode and Java would look
// similar

```
        intArea = calcAreaOfRect(intLength, intBreadth)
```

The above version, while a module, is more specifically a function that is required to return a value and therefore the return statement contains the variable (and in-turn the value) that is passed in this case to the variable named intArea.

The code below is also a module, but more specifically, is a procedure, that is expected to perform a task, but is not required to return back a value

** pseudocode – a procedure also uses the return statement as an end statement to the block of code in similar style to the version above

```
0. printTotal()
1.      output "The total number of something is " + intTotal
2.      output "The average number of something is " + fltAverage
3. return
```
===