Chapter 1 Recap
==========

Please note that the following are mainly points with some brief explanations to add to the notes you are compiling. The recap is an opportunity to revisit some of the work we have spoken about.

==========
Overview of Computers and Programming
==========
Chapter highlights:
-- Computer systems
-- Simple program logic
-- Steps involved program development cycle
-- Pseudocode statements and flowchart symbols
-- Using a sentinel value to end a program
-- Programming and user environments
-- The evolution of programming models

Computer systems
==========
- components required to process and store data using a computer

-- **Hardware**
	- An understanding of the components that make up a computer
	- Its purpose is important to understand
	- Motherboard, CPU, RAM, Graphics Card, HDD, Ports
-- **Software**
	- Application software (processing programs)
	- Systems software (manage a computer)
	- Programs - instructions written by programmers to perform a task
-- **3 major operations for computers *and the same could be said for programming***
	- Input (data that is read in based on the solution being developed)
	- Processing (calculations performed by the CPU)
	- Output (knowledge provided to the user)
-- **Programming languages**
	- Used to write programs
	- The term coding a program is more appropriate
	- **Syntax**
	-- rules that define the combinations of structured statements and expressions
	-- https://en.wikipedia.org/wiki/Syntax_(programming_languages)
	- **compiler/interpreter**
	-- a program that translates the source program written in some high-level language into machine code
	-- an interpreter converts each high-level statement into machine code during the program run
	-- compiler code runs faster, while interpreted code runs slower
	- **source code**
	-- programming statements that are created by a programmer with an editor
	- **object code**
	-- a compiled file, which is produced when the source code is compiled
-- **Computer memory**
	- **RAM**
	-- temporary, internal storage
	-- information loaded, is accessible during the execution of a program
	-- its volatile - information is lost, wiped out when the power is off
	- **Storage devices**
	-- non-volatile memory
	-- can store files that could be accessible by programs at a later time when required

---

Simple program logic

==========

-- **instructions**
- Remember the recipes for baking & other dishes
- There is an order of instructions that needs to be followed (*THE METHOD IN THE RECIPE*)
- We follow instructions to complete a task

-- **algorithms**
- like the statement made above, it is also a set of instructions to complete a specific task

-- **errors**
- 3 different kinds of errors: syntax, run-time and logical
- **syntax**
-- code was incorrectly written - did not follow the syntax rules
-- errors in the language
-- these types of errors are usually picked up when the compiler is invoked, telling you that mistakes have been made
- **runtime**
-- while the program is being executed, some code or value deals with something that is not acceptable & the error occurs and the causes the program to crash
- **logical**
-- an error in the way a program works
-- the program would still work, but the output produced is incorrect
-- so here the problem is you the developer

-- **procedures (break things down)**
- we relearn that the words, input, processing, and output help with the coding a program and that it is better when the program is broken down into smaller blocks of code – these blocks at the moment are known as procedures.

-- **variable**
- reserving a space in memory, referenced by a name, which is used to store values, processed in a program

-- **code statements**
- these are developed within the sections known as *input, processing, and output*
- equations & expressions form part of our learning journey and being able to develop these will help formulate solutions
-- *examples*
--- equation *x = y + 1*
--- expression *d >= 123 + f*

Steps involved program development cycle

==========

1. **Understand the problem**
- understand the end user - talk to them
- read supporting documentation to better understand the problem
2. **Plan the logic**
- develop the algorithm (*the sequence of steps that need to be executed*)
-- tools we use to develop solutions/programs
--- pseudocode & flowcharts
--- IPO chart (*input, processing, and output*)
--- desk checking / trace tables (*be able to walk through the code to evaluate process*)
3. **Code the program**
- combine the logic planned and the coding to develop source code
4. **Use software to translate the program into machine language**
- using the compiler / interpreter
- identify syntax errors and make the required corrections
- compile to make sure code is correct
5. **Test the program**
- execute the program with sample date to see if the results produced are correct
- identify logical errors
- debug the program if needed
- test many sets of sample data
6. **Put the program into production**
- allow the program to form part of the process used by company to now actively use your executable program to perform tasks
7. **Maintain the program**
- Be able to make changes as required
- repeat the development cycle

Pseudocode statements and flowchart symbols

==========

- **pseudocode standards**
    - English like representation of logical steps to solve a problem
    - Begin and end with the words start and stop
    - these 2 words are always aligned

    - module names require brackets ()
    - modules will end with the word return - aligned

    - each statement performs one action
    - statements are indented
    - continuation lines are also indented

    - no punctuation is used to end statements

Example of a pseudocode | program

```
0. start
1.      input greeting
2       output greeting
3. stop
```

- **flowchart**
    - visual representation of the logical steps developed in the pseudocode

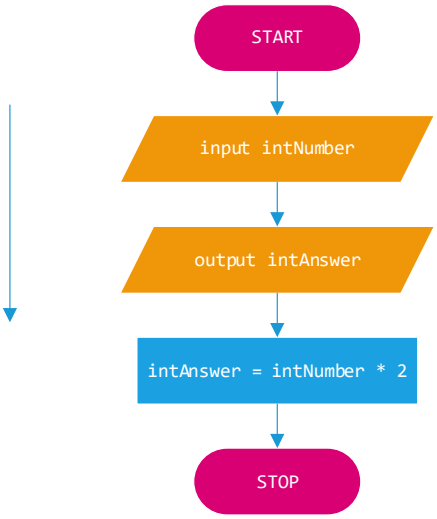
Flowchart symbols

Start/Stop - rounded rectangle.

input/output - parallelogram

Processing - rectangle

Flowlines - arrow lines - connecting shapes



Using a sentinel value to end a program

==========

** Since this is an overview of the work we are developing - there is an opportunity to explain how complex a program could possibly get

An idea presented here in the textbook is to execute code more than one time.

Imagine wanting to run the program 100 times - at the moment that would mean that we would need to rewrite the code 100 times. This is just not feasible. So the solution would be to develop the program using a loop logic structure to repeat the required lines of code 100 times. This means that we technically wrote the required code once and now have the ability to execute it multiple times.

Within the title of this section - the sentinel value - this can be thought of a flag - a value used internally by the program to decide when to stop of the execution of the loop.

While I do expect you to read through the contents of this section in the textbook, please note that this will be explained in more detail later in the semester.

Programming and user environments

==========

The author felt it necessary to help with knowing where this going.

After developing the pseudocode and the flowcharts - we would most likely use an IDE (Integrated Development Environment) to formally develop the source code for the program and use the embedded compiler and other tools to produce a working solution

There are 2 places where our programs would be executed.
- *command line - in a console window /*
- *graphical user interface - through a windows-based application*

The evolution of programming models

==========

procedural programming

- Writing down a list of instructions to tell the computer what needs to be done, step-by-step to complete a task
- This is the programming model we will be following for most of the semester

object-oriented programming

- This model organizes the design around the data, creating classes that contains internal variables, functions, procedures and effectively logic to deal with data that is loaded into a class.
- This type of programming is dealt with in the future and also important to note that some form of procedural programming will still exist within this model

---