

CMPT 120

Introduction to Computing
Science and Programming I

A Course Overview

Jeffrey Leung

Table of Contents

Page	Section
<i>02</i>	00 Introduction
<i>04</i>	01 Number Systems
<i>06</i>	02 Theory and Concepts
<i>10</i>	03 Application of Code
<i>12</i>	04 Python 2.X

Introduction

- Fundamental question of computing science: What can be (efficiently) automated?
 - Solving problems and managing data
- Hardware and software:
 - **Hardware:** Physical elements/components of a computer system
 - Composed of the CPU, main memory/RAM, secondary storage devices, input devices, output devices
 - **Central processing unit:** Small chip which performs operations
 - **Random-access memory:** Volatile memory used to keep temporary storage while the computer is running
 - **Secondary storage device:** Memory used to hold data for a long period of time
 - E.g. Floppy disk drive, CD, DVD, USB drive, hard disk drive, solid state drive
 - **Input device:** Component which collects and sends data to the computer
 - **Output device:** Component which produces data
 - **Software/programs:** Set of instructions to perform a task
 - Programs implement algorithms to be executed by the computer
 - **Programmer/software developer:** Person who designs, creates, and tests computer programs
 - **System software:** Controls and manages basic operations of a computer
 - **Operating system:** Controls hardware operations, manages connected devices, allows data to be saved/retrieved, and allows other programs to run on the computer
 - **Utility program:** Performs a specialized task which enhances the computer's operation or safeguards data
 - E.g. Virus scanners, data backup programs
 - **Software development tools:** Programs used by programmers to create, modify, and test software
 - E.g. Assemblers, compilers, interpreters
 - **Application software:** Makes a computer useful for everyday tasks

- CPU operations:
 - **Parse:** To find the meaning of
 - **Machine language:** Instructions executed directly by a CPU
 - **Instruction set:** Entire set of instructions a CPU can execute
 - **Assembly language:** *Low-level programming language* mimicking CPU instructions in human language
 - Must be 'translated' to a machine language by an assembler
 - **Fetch-decode-execute cycle:** A line of instruction is moved from memory into the CPU, the CPU decodes the instruction to understand what operation to perform, and the CPU executes the instruction
 - **High-level programming language:** Language which is very different from machine language, so as to simplify development and reduce necessary knowledge of the CPU
 - ASCII encoding (part of Unicode): 8 bits of binary code; 128 (2^7 different symbols)
- **Formal languages:** Languages designed by people for specific applications
- Layers of languages (Outer to inner): Application packages, system software, high-level languages, assembly languages, machine language
- Data storage:
 - **Bit** (Binary digit): Boolean storage location
 - Can only be set to '0' or '1'
 - Tiny rechargeable battery (capacitor)
 - **Byte:** Storage location consisting of 8 bits
 - Can only store a single letter of the alphabet or a small number
 - **Binary numbering system:** Numeric values written in base 2
 - From left to right, the values assigned to each position: 128, 64, 32, 16, 8, 4, 2, 1
 - Characters are stored as binary numbers
 - Negative numbers are encoded with *two's complement*;
Real numbers are encoded in *floating-point notation*
 - **Digital data:** Data stored in binary
 - **Digital device:** Device which works with binary data
- **Algorithm:** Finite sequence of unambiguous instructions for carrying out a procedure or solving a problem for any legitimate input in a finite amount of time
 - Can be described with flowchart diagrams or pseudocode
 - **Implementation:** To make something active/effective; to execute code
 - Code in programming language is translated by a compiler/interpreter into machine language, which is executed
 - **Interpreter:** Computer program which translates and executes each statement
 - **Compiler:** Computer program which translates a high-level program into a separate machine language program (object code/executable)

Number Systems

- Number systems in different bases:
 - Positional number system:** Weights of digits are based on their location
 - Notation: $\text{Number}_{\text{base}}$
 - Binary (base 2): 0, 1
 - Octal (base 8): 0, 1, 2, 3, 4, 5, 6, 7
 - Decimal (base 10): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
 - Hexadecimal (base 16): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
 - Conversions:
 - Binary to decimal:

- Each digit has weights:

Number:	1	1	0	0	1	0	1	1
Weight:	128	64	32	16	8	4	2	1
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

- Ruffini's method:

	1	1	0	1
2	0	2	6	12
	1	3	6	13

Add the leftmost column (First is 0)

Multiply by 2

Write in next column

Repeat

- Decimal to binary:

2	13	
2	6	1
2	3	0
2	1	1
	0	1

$$13_{10} = 1101_2$$

- Patterns in counting binary:

<i>Decimal:</i>	<i>Binary:</i>	<i>Hexadecimal:</i>
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9

10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

- Representing numbers:

- Integers:

- Signed binary*: Leftmost digit indicates the sign (0 – positive; 1 – negative)

- Ways of representing negative numbers:

- Sign/magnitude
 - 1's complement
 - 2's complement (Used by the CPU)

- Conversion algorithm:

- Positive to negative or negative to positive
 - Start with the number as the opposite sign (with the given number of bits), and flip all bits to the left of the rightmost 1, OR
Start with the number as the opposite sign (with the given number of bits), flip all bits, and add 1

- Usable numbers with 4 bits:

$$-8 \leq x \leq 7 \quad \text{OR} \quad -2^3 \leq x \leq 2^3 - 1 \quad \text{OR} \quad -2^{n-1} \leq x \leq 2^{n-1} - 1$$




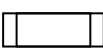
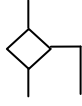

Theory and Concepts

- Programming languages:
 - Have *operators* (which perform operations on data) and *syntax* (a set of language rules which must be followed when writing a program)
 - Concatenate: To join together
 - **Flow of execution**: Order in which statements are run
 - **Syntax error**: Mistake in the rules of a language
 - **Semantic error**: Program does not do the intended task
 - Structured programming:
 - Sequence: Straightforward flow of code
 - Alternative/conditional structure: If/else conditional flow of code
 - **Recursion**: Fundamental programming technique in which the definition of something includes itself
 - Similar to matryoshka dolls
 - Examples of recursive definitions:
 - A list is either a value or a value followed by a list.
 - For any positive integer N , $N!$ is defined as the product of all integers between 1 and N inclusive.
Base case: $1! = 1$
Recursive case: $N! = N \cdot (N-1)!$
 - Functions involving recursion:
 - Must be structured to handle both a base case and a recursive case
 - **Direct recursion**: A function which calls itself
 - **Indirect recursion**: A function which calls another function which calls the first function
 - **Linear recursion**: A function which only calls itself once
 - **Binary (tree) recursion**: A function which calls itself multiple times

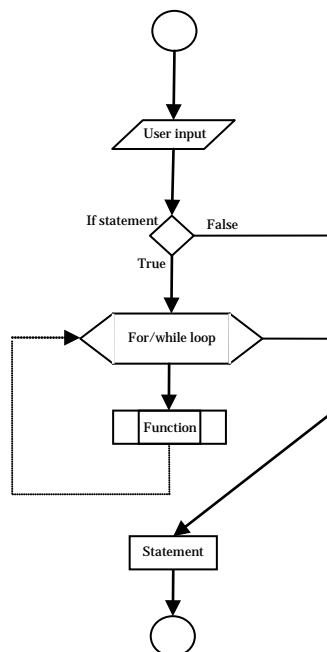
- Visual representations:

- State diagram: Way to represent variables on paper where the name of the variable has an arrow pointing to its value
 - Shows the 'state' (value) of the variable
- Stack diagram: Way to show the value of each variable and which function (if any) it belongs to, with an arrow pointing from the variable to its value and a frame surrounding the parameters and variables of each function

- Flowchart:

-  Start/stop
-  Statement
-  Input/output
-  Calling a function
-  If/elif/else statement
-  For/while loop

- Example diagram:



- **Order:** Time complexity of an algorithm
 - Denoted as $O(_)$
 - $O(n)$, $O(n^2)$, $O(2^n)$, $O(\log n)$, etc. where n is the size of the problem
 - Disregarded properties:
 - Constants e.g. $O(2n) = O(n)$
 - Lower exponents e.g. $O(n^2+n) = O(n^2)$
 - Big O notation (describes limiting behaviour of a function nearing infinity)
 - An algorithm is said to be in the order of some reference function
 - Relevant for algorithms involving large amounts of data
 - **Problem input size:** Number of *critical operations* (e.g. mathematical operations, prints, conditional statements, assignments) executed
 - As n (approximate size of the algorithm) approaches infinity, the algorithm requires different amounts of time to be executed depending on the value of the $O(_)$ function
 - Examples:
 - $x = 0$
 $y = 10$
 $x += 1$
 for i in range(n):
 $x += y$
 $y += 100$

 $2n$ additions are executed. Therefore, $O(2n) = O(n)$.
 - $count = 0$
 for i in range(n):
 for j in range(n):
 $count += 1$

 $n \cdot n$ additions are executed. Therefore, $O(n^2)$.
 - $count = 0$
 for i in range(n):
 for j in range(n):
 for k in range(n):
 $count += 1$

 $n \cdot n + n$ additions are executed. Therefore, $O(n^2+n) = O(n^2)$.

- **Sorting:**
 - **Bubble sort:** Algorithm which compares each two consecutive elements and orders them as many times as the length of the sequence, then as many times as the length of the sequence minus 1, etc.
 - $O(n^2)$
 - **Selection sort:** Algorithm which, on each of the (length of the sequence minus 1) passes, changes the positions of the most comparable element (disregarding the 1st to $(n-1)^{\text{th}}$ elements) with the n^{th} element, given that n is the number of passes which has occurred
 - $O(n^2)$

Application of Code

- **Keywords:** Words used by a program to recognize the structure of the program
 - Cannot be used as variable names
- **Module:** Package containing a collection of related functions (e.g. 'math')
- **Dot notation:** Methods of a module or class are accessible by the name of the module then the method, separated by a period
- **Variable:** Name which refers to a specific stored value
 - Similar to an empty box which has a name and can hold a single value
 - Naming conventions:
 - Starts with a lower-case letter and each subsequent word is capitalized (CamelCase)
 - No spaces
 - Relevant to purpose of variable
 - **Assignment statement:** Statement which creates a variable and assigns a value to it
 - **Flag variable:**
 - **Global variable:** Variable available to the entire program
 - Unless explicitly defined as global, variables will be unavailable to (other) functions
 - **Local variable:** Variable available only to a function or method
 - **Aliases:** Multiple variables which point to the same object
 - E.g. A list passed to a function
- **Values:**
 - Sequences:
 - Can be 'empty'
 - **String:** Sequence of characters
 - **Immutable:** Elements cannot be changed directly
 - Repeated concatenation: `<string> * <number>`
 - Concatenating with a negative number creates an empty string
 - **List:** Sequence of elements
 - **Mutable:** Elements can be changed directly
 - **Integer:** Number without any fractional part
 - **Floating point value:** Number with a fractional part
 - To output a floating point value, at least one of the numbers in the operation must be a floating point value

- **Boolean value:** Either True or False
 - Boolean expressions:
 - Can have relational expressions (comparisons)
 - Comparison operators: <, >, <=, >=, ==, !=
 - Operators used on strings will compare ASCII numbers
 - Used to control the flow of execution
 - Can be connected with logical operators (not, and, or, xor)
- **Operator:** Symbol used to represent a computation
 - Expressions are evaluated in the standard order of operations
 - **Expression:** Combination of values, variables, and operators
- **Statement:** Individual instruction in a program in a high-level programming language
 - **Code/source code:** Statements in a high-level programming language
- **Function:** Named sequence of statements which performs a computation
 - **Function definition:** Creates a new function with a name and a sequence of statements
 - **Argument:** An expression given to and used by the function
 - **Parameter:** Argument assigned to a variable inside a function brackets
 - **Local variable:** Variable created and existing inside a function
 - **Return statement:** Terminates execution of the function, and may return a value
 - **Void/subroutine function:** Does not return a value
 - **'Fruitful' function:** Returns a value
 - Can be built-in or created by the programmer
- Loop statements:
 - **For loop:** Repeats a code a number of times, guided by an index number
 - **While loop:** Repeats the code as many times as necessary until a specific condition becomes false
- **Methods:**
 - Applied to objects
 - Uses dot notation

Python 2.X

- General purpose high-level programming language
- Interpreter
- IDLE: Integrated DeveLopment Environment for Python
- Shell window allows:
 - Working interactively with Python
 - E.g. Doing calculations, executing one statement at a time, showing the output/results from a program
 - Chevron prompt (>>>) represents Python being ready for input
- Editor window (script mode) allows:
 - Editing and saving a Python program
- Using Python:
 - All Python files must be saved with a `.py` extension
 - F5 will execute the program
 - Indentation determines what each statement encompasses
 - Unnecessary indentation creates an error
 - Keywords: `int`, `string`, `in`, `input`, `if`, `elif`, `else`, `for`, `while`, `list`
 - Use code comments (`#` or `'''`) to describe code
 - Revision statements are useful
 - To duplicate an object without creating an alias:
 - E.g. Lists: `list2 = list1 + []`
 - Variables:
 - Case-sensitive
 - Do not require declaration of type
 - Global variables must be declared as `global` in each function utilizing them
 - Functions must be defined at the beginning of the code (or before they are called)
 - In boolean statements such as `if (____ and ____)`, if the first expression exits the conditional, the second expression will not be evaluated even if an error should be created
 - Range function generates and returns a list from 0 (or *start*) to the index before *stop*
 - Automatically increments the index by 1 if the step is unspecified
 - Lists can be taken as user input using the `<variable> = input()` function

- Turtle module:
 - Requires modifications to Python:
Navigate to Python's installation folder and create a shortcut to python.exe, or right-click the shortcut to the program. In Properties, at the end of the path in Target, add:
C:\Python27\Lib\idlelib\idle.pyw -n
 - Provides methods allowing the programmer to draw graphics
- CMPT 120:
 - Documentation of functions must include which arguments are received and which values are returned
 - Examples:

• () -> ()	No arguments; nothing returned
• (str, int) -> ()	String and integer arguments; nothing returned
• () -> (str)	No arguments; string returned
• (str) -> (int)	String argument; int returned