CMPT 120 Detailed Learning Outcomes

This page lists the ***tentative*** learning outcomes for each week of this course, and will be updated as the course progresses (**subject to revision**). Students are responsible for all content covered in the Readings, Lectures and Assignments, and should consider this page primarily as a study guide.

Week	**Code, Algorithms, Concepts and Processes**	**Functions and Keywords**	**Textbook**

Week 1
 	
Sep 4+
	Introduction to CS and first programs
	
	1. Know that CS is problem solving
	2. Understand problem solving by subdividing tasks into subtasks
	3. Able to explain the main characteristics of an algorithm
	4. Know what is plagiarism for programming, based on course policy
	5. Know what pseudocode is
	6. Able to write Python comments
	7. Know what information is in a program header block (i.e. initial comments with author, date and purpose)
	8. Able to output a "Hello World" using print()
	9. Able to contrast programming languages with natural languages
	10. Able to use an IDE like VS Code, IDLE, or repl.it
	11. Able to submit code in a .zip or .py file
		`print("Hello, World")`
		
		`print("I have",1)`
		
		`#`
		
		`"a string"`
		
		`""" a multi-line string """`
			• 1.1. The way of the program
			• 1.2 Algorithms
			• 1.3 The Python Programming Language
			• 1.4 Executing Python in Runestone Textbook
			• 1.5 More about programs
			• 1.11 Formal and Natural Languages
			• 1.12 A Typical First Program
			• 1.13 Comments

Week 2
 	
Sep 9+
	Strings, concatenation, conditional statements, lists, random module, relational operators, booleans
	
	1. Able to design/plan an algorithm, e.g. using comments or pseudocode
	2. Able to apply some common problem solving strategies, such as breaking down the problem into smaller pieces
	3. Able to obtain input into Python from the terminal to a variable
	4. Able to receive input from terminal without saving it to a variable
	5. Know how to assign a value to a variable
	6. Able to output a string variable in a print statement
	7. Able to concatenate two strings
	8. Know the constraints and conventions on variable naming
	9. Know that there are different types of data, although String is the focus for now
	10. Able to create a list of strings and assign it to a variable
	11. Able to use the random.choice() function on a list (including import)
		`= (variable assignment)`
		
		`+ (string concatenation)`
		
		`" " (string type)`
		
		`[ ] (lists)`
		
		`random.choice()`
		
		`import`
		
		`if / elif / else`
		
		`== < > !=`
		
		`and, or, not`
			• 2.1 Variables, Expressions and Statements
			• 2.2 Values and Data Types
			• 2.4 Variables
			• 2.5 Variable names and keywords
			• 2.6 Statements and expressions
			• 2.7 Operators and operands
			• 2.8 Input
			• 2.10 Reassignment
			• 5.1 Modules
			• 7.1 Boolean values and expressions
			• 7.2 Logical Operators
			• 7.4 Conditional execution: Binary Selection
			• 7.5 Omitting the else clause: Unary selection
			• 9.3 Concatenation
			• 10.2 List values

	12. Knows what the . after a module name does 13. Understands that modules contain functions (light treatment) 14. Know to put import statements at the top of the program, after header 15. Knows how to use if/elif statements with == 16. Knows how to use the else clause 17. Understands the meaning of logical operators and, or 18. Knows about comparison operators such as < > == and how to use in a conditional statement 19. Understands what a Boolean expression is and what it can be evaluated to (True/False) 20. Understands the basics in combining Boolean expressions, e.g. x or y, using strings only 21. Is able to print a Boolean expression 22. Knows some of the characteristics of good software: usable, pleasing to read, minimizes duplication, robust to errors 23. Knows to include a short description of the program in the header 24. Able to test a program for the desired outcome, interactively 25. Able to test a program (lightly) for unexpected cases, interactively 26. Knows how to test smaller pieces of code by commenting out blocks 27. Knows the interpreter's role in catching errors	`True, False`	

Week 3
 	
Sep 16+
	String methods, for loops, nested if's, integers, lists with variables
	
	1. Can apply the strip, lower, and upper String methods appropriately
	2. Can identify the String data type
	3. Able to use the REPL interactive console (or IDLE shell) to test methods and inspect variables
	4. Able to use the **in** keyword for both (1) string in a list and (2) character(s) in a string.
	5. **[New]** Able to use an interactive debugger to step through Python code and see variables change values.
	6. Can create a list using variables (e.g. from user inputs)
	7. Can use a for loop over elements of a list
	8. Understands the range(...) function and what it represents
	9. Knows the concept of the index variable in `for i in range(...)`
	10. Understands the Integer type
	11. Able to convert an Integer to a String
		`mystring.lower()`
		`mystring.upper()`
		`mystring.strip()`
		
		`in (strings/lists)`
		
		`for <var> in <sequence>:`
		
		`str(...)`
		`int(...)`
		
		`nested conditionals, i.e.`
		
		`if <condition>:`
		
		`    if <condition>:`
			• 1.7 Syntax Errors
			• 1.8 Runtime Errors
			• 1.9 Semantic Errors
			• 9.5 String Methods (except 9.5.1)
			• 9.13 The in and not in operators
			• 10.5 List Membership
			• 4.4 The for loop
			• 4.5 Flow of Execution of the for Loop
			• 4.7 Range function (except Turtle examples)
			• 2.2 Values and Data Types
			• 2.3 Type conversion functions

	12. Knows that concatenation is only applicable between 2 strings, not Int and String
	13. Is able to design and implement nested conditionals
	14. Understands the concept of robustness with respect to code
	15. Can identify whether an error is a syntax error or a semantic error
	16. Understands the concept of method chaining (applying to an object and sending output from one method to the method to its right, using the . operator)
		`else:`
		
		`range(4)`
		`range(1,5)`
		`range(1,10,2)`	

Week 4
 	
Sep 23+
	Arithmetic, conversion, accumulation pattern, for loop with range
	
	1. Can use the range function with arguments that are variables (not only numbers)
	2. Knows that a loop is a way to reduce duplication of code
	3. Able to use integers and floats and manipulate them in variables
	4. Knows how to initialize a variable of type Integer
	5. Can apply the accumulator pattern (including initialization) and += shortcut
	6. Able to get the length of a list
	7. Able to convert strings to integer type (esp. user input)
	8. Knows that division of integers converts type to float
	9. Able to perform arithmetic operations on numbers
	10. Can use the accumulator pattern with other arithmetic operators
	11. Can print floats to a given number of decimal places
	12. Able to generate formatted user output using either string's .format() function, or the f-string literals (f"I'm {age} years old!")
		`type(17)`
		`type(0.0)`
		`type("xoxo")`
		`type([1, 2, 3])`
		`type(True)`
		
		`2 ** 2`
		`3 * 4`
		`5 - 3`
		`4 + 4`
		`5 / 3`
		`5 // 3`
		`12 % 5`
		
		`x = 1`
		`x = x+1`
		`x += 1`
		`int(4.3)`
		`float("123.45")`
		`str(12.3)`
		
		`len(myList)`
		
		`print("Number: {:.3f}".format(myfloat))`
		`print(f"Number: {myfloat:.3f}")`
			• 2.2 Values and Data Types (review of Integers and Floats)
			• 2.3 Type conversion functions (review)
			• 2.7 Operators and Operands (review)
			• 2.9 Order of Operations
			• 2.10 Reassignment (review)
			• 2.11 Updating variables
			• 6.5.1 The Accumulation Pattern (activity func-4-6 only)
			• 9.5.1 String Format Method
			• 10.3 List Length

Week 5
 	
Sept 30+
	Working with text files, indexing and slicing strings and lists
	
	1. Able to open and read lines from a text file
	2. Able to split a string into a list
	3. Able to access a specific element(s) of a list using indexing/slicing
		`file = open("myfile.txt")`
		`file.readline()`
		`for line in file:`
		
		`mystring.split(...)`
			• 9.4 Index Operator: Working with the Characters of a String
			• 9.7 The Slice Operator
			• 9.8 String Comparison

	4. Able to access a specific character(s) in a string using indexing/slicing
	5. Able to perform comparisons between numbers, taking into account order of operators (operator precedence)
	6. Able to perform comparisons (e.g. !=, <,>) with strings
	7. Can interpret code with nested conditionals with comparison operators (e.g. !=, <,>=) and numbers
	8. Able to find the common elements between 2 lists
	9. Able to understand and use a nested for loop
	10. Able to apply operator precedence to evaluate expressions
	11. Able to concatenate lists
	12. Able to apply accumulation pattern for strings and lists (previously was numbers)
	13. Able to calculate the maximum among several values
		`mystring.strip(...)`
		
		`mystring[0]`
		`mystring[-1]`
		`mystring[:]`
		`mystring[3:5]`
		`mystring[:3]`
		`mystring[3:]`
		
		`"a"*3`
		`["a"]*3`
		`alist[2][0]`
		`list1+list2`
		`list1 = list1 + [elem]`
		
		`alist[0]`
		`alist[:3]`
		`alist[1:3]`
		`alist[4:4]`
		`alist[4:]`
		`alist[3:-1]`
		
		`"a"<"b"`
			• 10.4 Accessing Elements in a List
			• 10.6 Concatenation and Repetition for Lists
			• 10.7 List Slices
			• 10.8 Lists are Mutable
			• 10.18 Accumulation with Lists
			• 11.1 Working with Data Files
			• 11.2 Finding a File on your Disk
			• 11.3 Reading a File
			• 11.4 Iterating Over Lines in a File

Week 6
 	
Oct 7+
	Continuing with learning outcomes from Week 6
	
	1. Able to calculate the maximum/minimum among several values
	2. Able to coordinate between 2 or more lists using a common index
		`listA = [10, 30, 20, 30]`
		
		`maxIndex = 0`
		`maxVal = listA[0]`
		`for i in range(len(listA)):`
		`  if listA[i] > maxVal:`
		`    maxVal = listA[i]`
		`    maxIndex = i`
		`print(maxIndex)`
		`print(maxVal)`
		
		`listA = [10,20,30]`
		`listB = ["a", "b", "c"]`
		`index = ...`
		`listA[index]`
		`listB[index]`
			• 9.6 Length of strings
			• 9.9 Strings are Immutable
			• 9.11 Traversal by index
			• Accumulation of strings

Week 7	----- *Midterm content ends here* -----	*`Midterm content ends here -`*	**-- Midterm content ends here --** • External reading from Gaddis' textbook (PDF).

Oct 14+	**Bits and Bytes**
	
	1. Knows what a bit represents in a computer
	2. Can convert a given number of bytes, kilobytes, megabytes, gigabytes, into the number of corresponding bits (both 1000 and 1024 will be accepted as conversion constant)
	3. Able to convert a decimal number into its binary representation, and vice versa
	4. Knows the relationship between binary numbers and hexadecimal numbers
	5. Knows that binary numbers can be converted to hexadecimal by grouping in 4 bits (and vice versa)
	6. Knows the purpose of ASCII
	7. Knows the purpose of Unicode with respect to ASCII
	8. Knows that RGB colors are represented with 3 bytes (or 24 bits, or 6 hexadecimal digits)
		`1101(2 = 13(10`
		
		`10A(16 <-->`
		`0001 0000 1010(2`
			Chapter 1, especially section 1.3.

Week 8
 	
Oct 21+
	Midterm Week
	
	• Wednesday: Going over questions from practice midterm
	• Friday: Midterm
	
	Turtles, defining functions
	
	1. Able to use the Turtle package to create drawings, namely the functions listed here -->
	2. Able to read and understand basic Turtle code to visualize its output
	3. Able to create a function
	4. Able to create a function with parameters
	5. Able to call a function previously created in the program
	6. Able to call a function from a loop, possibly using the for loop index in the arguments
	7. Able to color the turtle using turtle color names
	8. Able to color the turtle using color coded with RGB values as a 3-tuple with values for (red,green,blue)
	9. Able to identify the scope of a variable, especially in relation to function scope
		`import turtle`
		`pet = turtle.Turtle()`
		`pet.forward(10)`
		`pet.stamp()`
		`pet.right(180)`
		`pet.left(90)`
		`pet.penup()`
		`pet.pendown()`
		`pet.goto(10,-10)`
		`pet.color("blue")`
		
		`turtle.colormode(255)`
		`mycolor = (255,0,120)`
		`pet.color(mycolor)`
		
		`def myfunction(a,b):`
		
		`myfunction(45,20)`
		
		`for i in range(...):`
		`      myfunction(i*2,i)`
			• 4.1 Hello Little Turtles!
			• 4.2 Our First Turtle Program
			• 4.3 Instances of Turtles
			• 4.6 Iteration and Turtles
			• 4.7 Range (review with Turtles)
			• 4.8 A few more Turtle Methods
			• 4.9 Summary of Turtle Methods
			• 6.1 Functions
			• 6.4 Local Variables (ignore return statement for now)
			• 6.11 Turtle Bar Chart

Week 9
 	
Oct 28+
	Defining fruitful/productive functions
	
	1. Able to create and use functions that return values
	2. Knows how to call a function so that the value returned from a function is received
	3. Knows that print() is different than return in a function
	4. Knows the effect that a return has if executed inside a loop (inside a function)
	5. Identifies cases when the value None is produced when calling a function
	
	While Loop
	
	1. Able to identify when a while loop would be appropriate compared to a for loop
	2. Able to create a valid while loop with a sentinel (control variable)
	3. Able to create a valid while loop with multiple control variables
	4. Able to use a while loop to validate user input
		`def multiplier_100(a):`
		`  return (a*100)`
		
		`receive = multiplier_100(5):`
		
		While Loops
		
		`<init variable/s>`
		`while <boolean with variable/s>:`
		`  <update variable/s>`
			• 6.2. Functions that Return Values
			• 6.5 The Accumulator Pattern (review, now with knowledge of return)
			• 6.6 Functions can call other functions
			• 6.7 Flow of Execution Summary
			
			While Loops
			• 8.3 The While Statement
			• 8.8 Other Uses of While

Week 10
 	
Nov 4+
	Image Processing - mutability of lists
	
	1. Able to create and use a module containing one's own defined functions
	2. Knows how pixel colors are represented by RGB values
	3. Knows how to pass global variables into local scope
	4. Knows to directly return the Boolean value evaluated from a Boolean expression in a function without using if statement in the function, and call the function and use the returned Boolean value.
	5. Able to access and modify a 2D image in the form of a list of lists, containing RGB values in the form of a list
	6. Able to manipulate 2 dimensional lists, process a row, process a column, process a specific element
	7. Knows how to import and access the contents of packages and modules
	8. Knows how to import a module with a short name
	9. Able to read, show and save images using the 3Dlist representing an image as provided in the cmpt120images module
	10. Able to extract and/or change the color as RGB and/or individual color components of a pixel using a 3DList representing an image as
		`def my_func(a,b):`
		`  return a < b`
		
		`if  my_less_than(2,30):`
		`     ..............`
		
		`my_3d_list[0][10][4]`
		
		`import cmpt120images`
		`import my_custom_module`
			• 8.11. Two-Dimensional Iteration: Image Processing (Note: The Runestone textbook image processing is analogous but slightly different from what we will use and is required in the course, which is provided in the cmpt120images module. Follow the readings for the theory)
			• 5.1. Modules and Getting Help (Revisited)
			• 5.2. More about using modules
			• 5.3 The Math Module
			• 5.5 Creating Modules
			• 10.8. Lists Are mutable
			• 10.24 Nested Lists

	provided in the cmpt120images module		

Week 11
 	
Nov 11+
	Alias vs. copy, lists and functions, mutability of lists
	
	1. Knows what a list alias is, versus a copy
	2. Knows the implication of sending a list as an argument to a function (i.e. argument and parameter become aliases)
	3. Knows how to modify a list in place using append()
	4. Knows the effect of modifying a list inside a function when the list is sent as parameter, even if it is not returned
	
	Recursion
	
	1. Knows the basic elements of a recursive function
	2. Able to analyze a recursively drawn tree in Turtle
	3. Able to write a simple recursive function that does not return any value, e.g. to draw concentric circles
	4. Understands the difference between executing a line of code before a recursive call vs. after a recursive call
	5. Is able to apply the 3 laws of recursion to write or analyze a basic recursive fruitful function
	6. Able to write a factorial function recursively
	7. Able to write code that can produce the sum of a list using recursion
	8. Able to write code that can reverse a string using recursion
	9. Able to write a recursive or iterative function to check if a string is a palindrome
		`mylist = [1,2,3]`
		`mylistalias = mylist`
		`mylist.append(4)`
		`# mylist and mylistalias now have 4 elements`
		
		`---`
		`def changes_list(alist):`
		`  alist[0] = 1`
		
		`origlist = ["a","b","c"]`
		`changes_list(origlist)`
		`# origlist is now changed`
		
		`---`
			• 10.10. Objects and References
			• 10.11. Aliasing
			• 10.12. Cloning Lists (fyi only)
			• 10.13. Repetition and References (fyi only)
			• 10.16. Append versus Concatenate
			• 10.19. Using Lists as Parameters
			• 10.22. Functions that Produce Lists
			Recursion
			• 16.1 What is Recursion?
			• 16.3 The Three Laws of Recursion
			• 16.5 Visualizing Recursion
			• 16.6 Sierpinski Triangle

Week 12 *Nov 18+*	**Recursion (Cont'd)** 1. Able to write code that can produce the sum of a list using recursion 2. Able to write code that can reverse a string using recursion 3. Able to write a recursive or iterative function to check if a string is a palindrome **Searching**		• 16.2 Calculating the Sum of a List of Numbers • 16.4 Converting an Integer to a String in Any Base • 6.2 Searching (pythonds book) • 6.3 Sequential Searching (pythonds book, not yet 6.3.1) • 6.4 Binary Search (pythonds book, not yet 6.4.1)

	1. Able to write a linear search function with various return types (Boolean using for/while, index of unique found element, indices of all found elements) 2. Able to recognize a binary search and produce code for it 3. Able to produce the code for a <u>recursive</u> binary search		

Week 13
 	
Nov 25+
	1. Knows how to swap different elements in a list
	2. Able to identify and write the code for a selection sort
	3. Able to describe intermediate steps of a selection sort
	4. Able to use the datetime or time module to compute the running time of code
	5. Able to apply learned search and sort algorithms on a dataset from a file
	6. Able to use range() with multiple parameters to iterate over a sublist, or to iterate backwards over a list
	7. Able to list 10 characteristics of a good algorithm/code
	8. Understands in what case(s) time complexity is important to consider
	9. Able to give 7 examples of reference functions commonly used with Big O notation and compare them
	10. Able to describe the general approach and functioning of Merge Sort
	
	Complexity Analysis
	
	1. Able to list 3 examples of critical operations
	2. Able to identify the number of critical operations in terms of an input size n, given a piece of iterative code
	3. Able to calculate the time complexity in big O notation for a piece of iterative code
	4. Able to give best/worst case scenarios (order and description of scenario) for Linear Search and Selection Sort
	5. Able to identify a piece of code that is O(logn)
	6. Able to analyze the time complexity of binary search and compare its use over linear search
	7. Able to give the complexity of Merge Sort
	8. Able to write the code to merge two ordered lists
		`a="1"`
		`b="2"`
		`# swapping`
		`temp=a`
		`a=b`
		`b=temp`
		
		`import datetime`
		`t = datetime.datetime.now()`
		
		`import time`
		`start = time.time()`
		`end = time.time()`
			From Python DS book:
			• 6.6 Sorting
			• 6.8 Selection sort (not yet about O(n2))
			• 6.11 Merge sort (Algorithm only, not required to learn to code it)
			• 3.2 What is algorithm analysis?
			• 3.3 Big-O notation
			• 6.3.1 Analysis of Sequential Search
			• 6.4.1 Analysis of Binary Search
			• 6.8 Selection Sort (complexity)
			• 6.11 Merge Sort (complexity)

Week 14	Review		

Dec 2			