COSC 1436 (PYTHON) Final Exam Review Sheet

Demonstrate knowledge of all these concepts:

1. Intro to Computers, Programs, and Python:

i. Computer hardware
   a. CPU
   b. Memory (RAM, ROM)
   c. Storage devices (USB, DVD)
   d. Input/Output devices (keyboard, monitor, printer, webcam, touch screen, scanner, etc…)
   e. Communication devices

ii. Computer software
   a. What is an Operating System (Windows, MacOS, etc…)
   b. Applications (Word, Excel, Canvas, Google, Visual Studio, Netbeans, etc…)

iii. Computer Programming Terms
   a. Bits, bytes, kilobytes, megabytes, gigabytes
   b. Syntax vs run-time errors vs logical errors (know examples of each)
   c. Low-level language (machine language, assembly language), high-level language (PYTHON, Java, Python, Swift, C#, etc…)
   d. What is machine language? What does platform dependent mean? What does platform independent need?
   e. Algorithm – steps needed to solve a problem

iv. Programming Environment
   a. What is an IDE
   b. IDEs (Visual Studio, Netbeans, Eclipse, Python IDLE, Xcode, etc…)
   c. IDEs contain editors, compilers/interpreters, debuggers, etc…
   d. Compiler vs. Interpreter (what is their purpose?)
   e. What is the source code? What is its file extension?

2. Elementary/Basic Programming:

i. Demonstrate knowledge of identifiers, variables, and constants.
   a. What is an identifier? What are valid identifier names (what characters are they composed of)? What does case-sensitive mean?
   b. What is a variable? What does a variable represent?
   c. Variable vs constant
   d. If this was the only line in your program, would it work? Why not??
      - `print (x)`

ii. Assign data to variables and know what a LITERAL means and how to assign an integer, float, and string literals to variables
   - What is a literal? A literal is *literally* the data (1, 4.3, ‘x’, “hello”).
   - You assign (store) literals to variables using the assignment operator (=)

- A variable (storage location) should be the only item to the left side of the equal sign.
- Look to the right side of the = and evaluate the expression. Any variables on the right side of the equal sign must have a value assigned to it before you can evaluate; if not, an error will occur. Evaluate, and then store the value in the variable to the left of the =.

iii. Understand arithmetic operators (`**, +, -, *, /, //, %.` Specifically, understand the INTEGER DIVISION (// operator) and MODULUS (% operator):

Expression	Output
`1/2`	`0.5`
`1//2`	`0`
`3/4`	`0.75`
`3//4`	`0`
`5%3`	`2`
`3%4`	`3`
`0%2`	`0`

iv. Evaluate arithmetic expressions. Know operator precedence (the order of operations). For example

- `5*2+3-8//3%4**2` ➔ `11`

v. Simultaneous Assignment: `n1, n2 = n2, n1  # swaps numbers`

vi. Augmented/Combined Assignment: `*=, +=, -=, /=, %=`

vii. Analyze code with simple I/O operations using the standard input and output functions:

a. `input()` vs `print()`

b. How do you use input and print?

c. Convert input() function strings to numbers using `int` and `float` functions

d. What's the difference between the following statements:

- `print(5 5)`
- `print("5 5")`

e. How do you print out the value of a variable? When do you use single or double quotes? What's the difference between the following statements:

- `print("number of drinks =", drinks)`
- `print(drinks + pizza)`
- `print("drinks + pizza")`

viii. Steps (in order) for Software Development Process/Software Development Life Cycle (SDLC):

a. Review the life cycle from revel readings

b. Know the order and the basics of each step

3. Selections/Conditional Statements:

i. Understand selection statement

i. Relational operators: `<  >  >=  <=  ==  !=`

a. Understand the difference between assignment operator and equality operator `=` vs `==`

ii. Logical operators: `not, and, or`

iii. Boolean types `(bool)` , values (`True` and `False`), Boolean variables (`flag = True, repeat = False`, etc.) and Boolean expressions. Examples below:

Expression	Output
`int(True)`	`1`
`int(False)`	`0`

`bool(1)`
`-OR-`
`bool(n) where n is >= 1`
	`True`

`bool(0)`	`False`

iv. Evaluate Boolean expression with relational and logical operators (know operator precedence). Examples below:

Expression	Output
`True and True`	`True`
`False or True`	`True`
`not False or True`	`True`
`True and False`	`False`
`True and False and False`	`False`
`2 * 2 -3 > 2 and 5 >= 6`	`False`
`3 + 4 * 4 > 5 * (4 + 3) - 1`	`False`
`(10 != 10) or (15 < 15)`	`False`
`(10 != 10) or (15 <= 15)`	`True`
`(10 == 10) and (15 <= 15)`	`True`

v. What's the difference between the following two items:
   a. (X == 5)
   b. (X = 5)

vi. How do you write a relational expression to check to see if a number (Z) is between X and Y.
(e.g., Z >= X and Z <= Y)
   a. Know when to use <=, >= and just <, >. *When a number is between two numbers, inclusive of those numbers, use the <=, >=.*

vii. Analyze simple if, if-else statements.

viii. Analyze complex if, if/else, if/elif/else, and nested if statements. Make sure to indent statements within if body.

ix. Analyze match statements. Example below:

```
grade = 'A'
match grade:
    case 'A':
        print("A grade")
    case 'B':
        print("B grade")
    case 'C':
        print("C grade")
    case 'D':
        print("D grade")
```

```
case 'F':
    print("F grade")
case _:
    print("Invalid grade")
```

x. Analyze/Evaluate code with conditional expressions. Examples below:

Statement	Output
`x = 2 if 2 > 3 else 3` `print(x)`	3
`print(0 if 10 % 3 == 0 else 1)`	1
`print("odd" if 10 % 2 else "even")`	even

xi. Know to import random module and analyze code with random functions (`randint, randrange`). Examples below:

Expression	Output
`random.randint(0, 5)`	generates integers between 0 and 5
`random.randrange(0, 5)`	generates integers between 0 and 4
`random.randint(0, 1)`	generates 0 or 1
`random.randrange(0, 1)`	generates 0 always

4. Math functions and strings:

i. Evaluate expressions that may involve built-in, math, `chr`, and `ord` functions. Examples below:

Expression	Output
`max(10,2)`	10
`min(10,2)`	2
`max(min(max(10,2), 2), 8)`	8
`pow(min(2,3),3)`	8

`round(4.5)`
`round(5.4)`
`round(5.5)`
	4
	5
	6

`int(4.5)`
`int(5.4)`
`int(5.5)`
	4
	5
	5

`math.ceil(2.1)`	3
`math.floor(2.1)`	2
`math.sqrt(9)`	3
`ord('a')`	97
`chr(98)`	'b'
`ord('A')`	65

ii. Know to use concatenation (+), repetition (*), `in`, and `not in` operators with strings and string slicing. Examples below:

Statement/Expression	Output
`print('''I made an 'A' in "COSC1436"''')`	I made an 'A' in "COSC1436"

`print("\"Knowledge\"", end = ' ')` `print('is the key to \'success\'')`	`"Knowledge" is the key to 'success'`
`print("price = $"+str(9.99)+'\n'+'~'*15)`	`price = $9.99` `~~~~~~~~~~~~~~~`
`"python" in "programming using python"`	`True`
`"python" not in "programming using PYTHON"`	`True`
`"Hey there" > "hi":`	`False`
`len("COSC 1436")`	`9`
`min("ABCabc")`	`'A'`

`str1 = "ABCabc"`
`answer = ""`
`answer += str1[4]`
`answer += str1[0]`
`answer += str1[-1]`
	
	`b`
	`bA`
	`bAc`

`str1 = "ABCabc"`
`str1[1:3]`
`str1[:2] + " " + str1[2:-1]`
	
	`'BC'`
	`'AB Cab'`

iii. Analyze code that uses string methods:

a. Manipulating strings: `isalpha(), isdigit(), isalnum(), islower(), isupper(), isspace()`

b. Searching strings: `find(), count(), endswith(), startswith()`

c. Converting strings: `lower(), upper(), capitalize()`

d. Stripping characters: `strip(), lstrip(), rstrip(), etc.`

Expression	Output
`"1436".isdigit()`	`True`
`"COSC1436".isalpha()`	`False`
`"COSCXXXX".isalpha()`	`True`
`"COSC1436".isalnum()`	`True`
`"COSC XXXX".isupper()`	`True`
`"COSC XXXX".islower()`	`False`
`"COSC XXXX".isspace()`	`False`
`"  ".isspace()`	`True`
`"welcome".endswith("come")`	`True`
`"welcome".startsswith("good")`	`False`
`"welcome".find("we")`	`0`

`"welcome".find("me")`	`5`
`"welcome".find("become")`	`-1`
`"welcome".rfind("e")`	`6`
`"welcome".count("e")`	`2`
`"welcome".capitalize()`	`Welcome`
`"cosc".upper()`	`COSC`
`"WELCOME".lower()`	`welcome`
`"    welcome    ".strip()`	`welcome`
`"    welcome    ".rstrip()`	`'    welcome'`
`"    welcome".lstrip()`	`welcome`

iv. Formatting output (numbers and strings) using `format` function, using end argument, etc.:

Statement	Output
`print(format(12345.678910, "10.2f"))`	`  12345.68`
`print(format(12.345, "10.2f"))`	`     12.35`
`print(format(12.3, "<10.3f"))`	`12.300`
`print(format(12.345, ".2f"))`	`12.35`
`print(format("python", ">10s"))`	`        python`

`gpa = 3.5`
`grade = 'B'`
`print(f"GPA {gpa} is {grade} grade")`
	`GPA 3.5 is B grade`

`roman = 'X'`
`print(f"Roman numeral '{roman}' is", end = ' ')`
`print(10)`
	`Roman numeral 'X' is 10`

5. Loops:

i. Know basic loop terminology: Iteration, nested loops, loop body, infinite loop, incrementing the counter

ii. Analyze code that uses `while` and `for` loops. Make sure to indent statements within body of the loop

iii. Differentiate user controlled and sentinel controlled `while` loops

iv. Analyze code with counter-controlled `for` loop with range functions

v. Know to use `continue` and `break` statements

vi. Analyze code that uses more complex loops e.g., nested loops, loops with conditionals, combining different types of loops, loops with break/continue, etc.

vii. Analyze problems using loop structures (e.g., running total, average, min, max, power, factorial, etc.)

a. Know how to analyze code to display even or odd numbers, horizontally or vertically.

b. Know how to analyze code to sum up numbers between X and Y.

c. Know how to analyze code to print out numbers between X and Y, and then print out their squares, or cubes, etc.

viii. Analyze code to execute statements repeatedly using simple `while` and `for` loops. Look at the following loops. Understand why some have 5 iterations, and some 6.

a. Understand the difference between the ones that use counters that increment (i += 1) and counters that decrement (i -= 1). Don't memorize, but rather work them out with a pen and paper, incrementing/decrementing the counters. Examples below:

Below iteration/repetition of loop body: 6 times	**Below iteration/repetition of loop body: 5 times**
`for i in range(6):` `  print("Hello!")`	`for i in range(5):` `  print("Hello!")`
`for i in range(1, 7):` `  print("Hello!")`	`for i in range(1, 6):` `  print("Hello!")`
`for i in range(6, 0, -1):` `  print("Hello!")`	`for i in range(5, 0, -1):` `  print("Hello!")`
`for i in range(5, -1, -1):` `  print("Hello!")`	`for i in range(4, -1, -1):` `  print("Hello!")`

`i = 0`
`while i < 6:`
`  print("Hello!")`
`  i += 1`
	`i = 0`
	`while i < 5:`
	`  print("Hello!")`
	`  i += 1`

`i = 1`
`while i <= 6:`
`  print("Hello!")`
`  i += 1`
	`i = 1`
	`while i <= 5:`
	`  print("Hello!")`
	`  i += 1`

`i = 6`
`while i > 0:`
`  print("Hello!")`
`  i -= 1`
	`i = 5`
	`while i > 0:`
	`  print("Hello!")`
	`  i -= 1`

`i = 5`
`while i >= 0:`
`  print("Hello!")`
`  i -= 1`
	`i = 4`
	`while i >= 0:`
	`  print("Hello!")`
	`  i -= 1`

`etc…`	`Etc…`

ix. SHORT ANSWER: Solve a problem using loop structures:

a. Know how to write code to display even or odd numbers, horizontally or vertically.

b. Know how to write code to sum up numbers from X to Y.

c. Know how to write code to print out numbers from X to Y, and then print out their squares, or cubes, etc.

d. Know how to use a loop to read in a number from a user that is between X and Y.

e. Know how to print 1000 items, vertically or horizontally (such as: 1000 numbers with spaces in between each, 1000 numbers backwards, 1000 numbers with a comma, 1000 '+' signs).

f. Use a loop to count the number of uppercase or lowercase letters in a string.

g. Use a loop that reads in characters from a user, until a certain character is entered (such as 'q' or 'Q').

6. Functions:

i. Function terminology: void function, value returning functions, returning multiple values, calling, and defining functions, function header, function body, formal/actual parameters (arguments), default function arguments, variable scope, etc.

ii. Know the difference between global and local variables (having two with the same name, their scope, which ones you should use, where you define a global variable, etc.). Examples below:

Code	Output	Code	Output	Code	Output

`def globalFunc():`
`  global x`
`  x = 1`
`  print(x)`
`def main():`
`  globalFunc()`
`  print(x)`
`main()`
	1
	1
		`x = 1`
		`def globalFunc():`
		`  x = 5`
		`  print(x)`
		`def main():`
		`  globalFunc()`
		`  print(x)`
		`main()`
			5
			1
				`x = 1`
				`def`
				`globalFunc():`
				`  global x`
				`  x += 1`
				`  print(x)`
				`def main():`
				`  globalFunc()`
				`  print(x)`
				`main()`
					2
					2

iii. Distinguish/demonstrate knowledge of function structures e.g., function header, function parameters, function calls, single and multiple value-returning functions & void functions, function with default arguments.

  a. Hint: Void functions should not have return statements

  b. Hint: non-void (value-returning) functions should have return statements

iv. Know the difference between a function call and function header syntax. Also, understand functions with default arguments. Look at different examples in revel readings to get the hang of it:

# Function Calls	# Function Definitions
`myFunc1()`	`def myFunc1():` `  print("Hello!")`
`myFunc2("COSC1436", 95.5)` -OR- `myFunc2("COSC1436")` # default argument	`def myFunc2(cName, avg=70):` `  print("You made", avg, "in", cName)`
`result = myFunc3()` `print(result)`	`def myFunc3():` `  return 5 % 2`
`print(myFunc3())`	
`anotherFunc((myFunc3())`	`def anotherFunction(num):` `  print(num)`
`num1, num2 = myFunc4(5, 10)` -OR- `num1, num2 = myFunc4(11)` -OR- `num1, num2 = myFunc4()` # default arguments and return multiple arguments	`def myFunc4(n1=0, n2=1):` `  return n2, n1`
etc…	etc…

v. Given a function header, create an appropriate function call, and vice versa.

vi. Know how to define, call, and analyze

  a. single/multiple value-returning functions with or without arguments
  b. void functions with or without arguments
  c. void or value-returning functions with default-arguments

vii. Analyze code to determine the scope of local and global variables.

viii. SHORT ANSWERS: Write functions to solve common/typical problems (Note: know how to print decimal numbers with 1 or 2 decimal places):

h. **Know how to determine the highest or lowest number of its parameters, and either returns the answer or prints the answer.**

i. **Know how to calculate the sum/average of its parameters, and either returns the answer or prints the answer.**

j. **Know how to calculate pay based on its parameters (hours times salary), and either returns the answer or prints the answer.**

k. **Know how to calculate an equation based on its parameters, and either returns the answer or prints the answer.**

l. **Know how to determine if a value sent to a function meets a certain requirement (lower or higher than a number, even or odd, etc...).**

m. **Have a function return a true or false if a string sent to the function contains all uppercase letters (or all lowercase letters, or perhaps, only alphabetic characters).**

7. Lists:

i. What is a list?

ii. Know how to create and traverse lists using for loop, use of `append` and `list` functions, and analyze list comprehension. Examples below:

Create a list	Explanation
`listName = [0, 1, 2]`	
`listName = ["black", "white"]`	
`listName = ["one", 1, 'I']`	In Python, lists are heterogeneous (i.e. can have values of different types)
# the **list** function `listName = list("ABCD")`	Creates a list of the characters in the string i.e. [‘A’, ‘B’, ‘C’, ‘D’]

`listName = []`
`for ch in "ABCDF":`
`  listName.append(ch)`
	Same as above

Traverse a list	

`listName = [0]*3`
`for i in range(3):`
`  listName[i] = i`
	listName is [0, 1, 2]

# append function 	
`listName = []`
`for i in range(3):`
`  listName.append(i)`
	listName is [0, 1, 2]

# list comprehension `listName = [i for i in range(3)]`	listName is [0, 1, 2]
`list1 = [ 10, 12, 15, 23, 40]` `list2 = [x for x in list1 if x %2 == 0]`	list2 is [10, 12, 40]

`list1 = [ 10, 12, 15, 23, 40]` `list2 = [x//5 for x in list1 if x %2 == 0]`	list2 is [2, 2, 8]

iii. List functions and methods: `len, max, min, sum, count, index, reverse, sort, pop, remove, etc.`

Expression	Output
Given: **`lst = [30, 1, 2, 0, 30, 30]`**	
`min(lst)`	`0`
`max(lst)`	`30`
`sum(lst)`	`94`
`len(lst)`	`7`
`lst.index(30)`	`0`
`lst.count(30)`	`3`
`lst.reverse()`	lst will be ➔ **`[30, 30, 0, 2, 1, 30]`**
`lst.sort()`	lst will be ➔ **`[0, 1, 2, 30, 30, 30]`**

`lst.pop()`
 	
`lst.pop(3)`
	`30` (last element is removed)
	so lst will be **`[30, 1, 2, 0, 30]`**
	`0` (index 3 element is removed)
	so lst will now be **`[30, 1, 2, 30]`**

`lst.remove(30)`	lst will be ➔ **`[1, 2, 0, 30, 30]`**

iv. List slicing `[start : end : step]`

Expression	Output	Display list elements
Given: **`lst = [30, 1, 2, 0, 30, 30]`**		
`lst[2:4]`	`[2, 0]`	start index 2 to end index 4 – 1: so elements in lst[2], lst[3]
`lst[:2]`	`[30, 1]`	start index 0 to end index 1: elems in lst[0], lst[1]
`lst[3:]`	`[0, 30, 30]`	start index 3 to end of lst: elems in [3], [4], [5]
`lst[1:5:2]`	`[1, 0]`	start index 1, end index 4, step 2: elems in [1], [3]
`lst[:5:2]`	`[30, 2, 30]`	start index 0, end index 4, step 2: elems in [0], [2], [4]
`lst[-4:-2]`	`[2, 0]`	start index -4, end index -3: elems in [-4], [-3]
`lst[:-2]`	`[30, 1, 2, 0]`	start index 0, end index -3: elems in [0], [1], [2], [3]
`lst[1:-3]`	`[1, 2]`	start index 1, end index -4: elems in [1], [2]
`lst[-2:-5:-2]`	`[30, 2]`	start index -2, end index -4, step -2: elems in [-2], [-4]
`lst[-2::-2]`	`[30, 2, 30]`	start index -2, end index 0, step -2: elems in [-2], [-4], [0]
`lst[:-1]`	`[30, 1, 2, 0, 30]`	start index 0, end index -2: elems in [0] to [4]
`lst[::-1]`	`30`	start index -1, end index 0, step -1: all elements in reverse

v. List operators: `+, *, in` and `not in`

Statement	Output
Given: **`lst1 = [2, 3]`** and **`lst2 = [1, 9]`**	
`lst3 = list1 + list2`	`[2, 3, 1, 9]`
`lst4 = 3 * lst1`	`[2, 3, 2, 3, 2, 3]`
`2 in lst4`	`True`
`9 not in lst4`	`True`
`9 not in lst3`	`False`

vi. Copying/duplicating list, differentiate the two methods below:

a. Using assignment operator will not copy the contents of *`list1`* to *`list2`* but copies the **reference** from *`list1`*

- `list1 = [0, 1, 2]`
  `list2 = list1`
  `list1[0] = 55`
  `print(list2)` ➔ **`[55, 1, 2]`** *(since `list2` references list1 any element changed in `list1` will show in `list2`)*

b. To get a **duplicate copy** of list1 to list2, you can use the following methods:

- `list1 = [0, 1, 2]`
  `list2 = [i for i in list1]` **-OR-**
  `list2 = [] + list1` **-OR-**
  `list2 = list1[:]`
  `list1[0] = 55`
  `print(list2)` ➔ **`[0, 1, 2]`** *(since a copy of `list2` is made, any change to `list1` element will show only in `list1` and not `list2`)*

vii. Analyze functions with list as arguments

viii. Analyze functions that returns list

ix. Analyze common problems using lists e.g., linear search, selection-sort, counting occurrences etc.

x. Understand basics of two-dimensional lists:

a. Know how to create 2D lists:

 # 3x2 list initialized to 0 to 5:

```
matrix3by2 = [[0, 1], [2, 3], [4, 5]] -OR-
matrix3by2 = [[0, 1],
              [2, 3],
              [4, 5]]
```

 # 3x4 list initialized to 0:

```
matrix3by4 = [[0]*4, [0]*4, [0]*4] -OR-
matrix3by4 = [[0]*4,
              [0]*4,
              [0]*4]] -OR-
matrix3by4 = []
for r in range(3):
    matrix3by4.append([0]*4)
```

 # 2x3 list initialized to random single digit integer:

```
matrix2by3 = [[0]*3, [0]*3]
for r in range(2):
    for c in range(3):
        matrix2by3[r][c] = random.randint(0, 9)
```

 # 2x2 list initialized to '$' symbol:

```
matrix2by2 = [['']*2, ['']*2]
for r in range(2):
    for c in range(2):
        matrix[r][c] = '$'
```

b. Know how to access/print 2D list elements:

Statement	Output
Given: **`matrix = [[0, 1], [2, 3], [4, 5]]`**	
`print(matrix[0][-1])`	1
`print(matrix[-1][-1])`	5
`print(matrix[1][1])`	3
`print(matrix[2][0])`	4
`print(matrix[-1])`	[4, 5]
`print(matrix[0])`	[0, 1]

xi. SHORT ANSWERS: Write common list operations (displaying list, values summing all elements, finding min and max elements, random shuffling, shifting elements, reading data into a list) - use loops to process lists.

a. Know how to print out the items in a one-dimensional list

b. Know how to sum the items in a one-dimensional list

c. Know how to average the items in a one-dimensional list

d. Know how to search for an item in a one-dimensional list and/or to count how many times an item appears in an array.

e. Know how to copy items from one-dimensional list into another one-dimensional list

f. Know how to print out the elements of a one-dimensional list in reverse order

g. Know how to count the number of vowels in a given string, or count the number of capital letters in a given string, or perhaps count the number of lower letters in a given string.

h. Know how to send a list to a function. Know how to copy the odd numbers from the original list to a new list, returning the new list. Know how to copy the even numbers from the original list to a new list, returning the new list. Know how to copy numbers less than 10 from the original list to a new list, returning the new list.

i. Know how to split a string into substrings using a given delimiter e.g. splitting a sentence into a list of words, counting number of words in a sentence, etc.