Finals Study guide

Finals for CE 311K is a three-hour closed-book exam held on **17th December 2019** in **Graduate School of Business Building 2.124** `https://utdirect.utexas.edu/apps/campus/buildings/nlogon/facilities/UTM/GSB/` between **9AM and 12PM**. You may bring three sheets of 8.5 x 11 inch of your own handwritten notes to the examination. You may use your calculator. You may not access the internet during the exam. The exam questions will be determined such that they satisfy a subset of the objectives listed here.

Finals will cover:

- Lecture Introduction, Control flow, errors, functions, data structures (only lists), Taylor series and Newton Iterations, vectors, matrices, solution of linear system of equations. *Tuples, Dictionary, Exceptions and Useful libraries are not part of the exam*
- Relevant assignments and labs

To perform successfully on the finals, you should be able to:

1. Determine if a given Matrix-Vector operation is valid or not (for e.g., $A \cdot b$, where A is a 3x3 matrix and b is a 3x1 vector) and estimate the shape of the output.
2. Use array slicing in vectors and matrices to get selected sub-set of elements, rows or columns.
3. Perform Gauss Elimination by hand for a given set of 3 linear equations.
4. Perform a maximum of three iterations of Gauss-Seidel for a given system of 3 linear equations by hand.
5. Evaluate if a given matrix A can be solved using Gauss-Seidel iterative approach.
6. Construct the system of linear equations of a truss in a matrix format. Linear equations will be provided and there is no need to solve the system of equations.
7. Exceptions are not part of the exam.
8. Useful libs are not part of the exam.

In addition, Finals will also cover the topics from Exams I and II:

1. Determine data types (`bool`, `int`, `float`, `string`) of different operations (for e.g, $*, +, -, /, \%, //$)
2. Write simple programs using variables and evaluate the value bound to a variable(s) at different step(s) in the code.

3. Understand and evaluate the order of precedence in a given statement(s).
4. Identify syntax, semantic and static-semantic errors in the code. It is not required to identify the exact type of error in the code, but the location of an error(s) in the code.
5. Identify and fix incorrect code either using the output error message, verification result or logic.
6. Understand and develop logic / algorithms / code that involve iterations (single and nested `for`, `for - else` and `range`) and control flow (`if`, `elif`, `else`, `break` and `continue`).
7. Understand the output of a given code or expression (for e.g., `range()`)
8. Explain in your own word what are relative and absolute errors.
9. Evaluate relative and absolute error(s) for a given program.
10. Evaluate a suitable termination criteria (`break`) based on the relative or absolute error.
11. Determine data types and outputs during casting operations. Please note only Python standard data types (`str`, `int`, `float`, `bool`) will be covered. Numpy data types are not included (for e.g., `np.float16` and others)
12. Define function with arguments (including default) and multiple return types for a given problem and call (use) the functions in a Python code.
13. Rewrite a given program using functions to make re-use of code as much as possible.
14. Identify and fix errors in passing function arguments and return types.
15. Evaluate the output of a given function(s).
16. Evaluate the value of different variables within and outside the function (scoping)
17. Recursions are **not** part of the exam.
18. Develop Python code that use, index, manipulate and search (`in` and `not in`) lists.
19. Iterating through a list using indexing and `in` operations.
20. Write a simple list comprehensions with a filter for a given list and a condition.
21. Deduce the value of a variable after trying to modify a list item and a tuple using an index or a key.

22. Use of dictionary is **not** part of the exam.
23. Develop Taylor series approximation for non-polynomial functions for single variable functions. Write a Python code to solve for the Taylor approximation with relative errors.
24. Develop Newton-Raphson code to find the root of a function. Compute the tolerance error at each iteration.

You won't be required to write lengthy code (more than 30 lines). I will not penalise for obvious typos and syntax errors in your code (for e.g., missing : at the end of function definitions), unless that is what is tested.