05-03—2023

CS051A

INTRO TO COMPUTER SCIENCE WITH TOPICS IN AI

Final Exam Study Guide

Alexandra Papoutsaki

she/her/hers

Lectures

Zilong Ye

he/him/his

Labs

How to study

- You can bring in **four** pages of notes (either four pieces of paper, single-side or two pieces, double-sided).
- Go over the slides/notes slowly and deliberately.
- Practice writing code on paper. Once you are done, consider that as your submission and transfer your code to PyCharm. Is it syntactically correct? Test it with various inputs. Does it do what you thought it would?
- Do the practice problems/exam WITHOUT looking at the solutions.
- Open all the provided python files; look at the docstrings of the functions and make yourself implement them before you compare your response with the provided code.
- Review the assignments and feedback.
- Office Hours: Monday May 8th 11am-noon, 1-4pm.

Lecture 1

- Built-in math calculations and order of precedence of operators
- int and float numeric types
- Variables, assignment statements
- Integer division
- Comments

Lecture 2

- Syntax for defining functions
- How to call functions
  - Parameters vs arguments
- How to store what a function returns
- Strings and how to concatenate them
- `int()`, `float()`, and `str()` functions

Lecture 3

- `print` vs `return`
- How to write multiline comments and docstrings
- How to import a module using `from module_name import *` so that you can use its functions
- Review `turtle` module functions
- `for i in range(number)` loops to execute a block of times `number` times, starting from 0 going to `number-1`

Lecture 4

- `random` module and its functions
- Importing only one function
  `from module_name import function`
- bools, T/F questions we can ask, and truth tables
- `if`, `if else`, `if elif elif… else` statements
  - Order matters! Satisfying one means you don't check the rest!
- `input` function
  - Gives you a string, you need to convert it to other types

Lecture 5

- `while` loops
- `for` loops being equivalent to `while` loops

Lecture 6

- How to define a list and how to reference it
- How to index a list (start at 0 to `len(list)-1`)
- Negative starting at -1 goes backwards
- You can have lists of mixed types or even lists of lists
- Slicing [a:b], from a to b-1. Variations of slicing
- [:] deep copy of a list
- For each loops
- list **methods** (lists are mutable, most methods do not return anything but alter the list)
    - *append*, *pop* (two versions), `insert`, `sort`
- Difference between append and + operator
- * operator for sequences
- Sequences and their shared properties
- Strings as sequences (strings are immutable!)
- Tuples as sequences (tuples are immutable!)
    - How to unpack a tuple (see three variations of print_movies in movies.py)

Lecture 7

- Be able to trace what happens when a function is called (and potentially calls another function) to its local vs global variables and parameters/arguments
- in keyword for sequences
- Number-game file contains good examples of how to structure while loops to ask user input
- Sequence operators (+, *)
- Aliasing and how lists can be affected

Lecture 8

- Scope (who can see what)
- String methods (strings are immutable, the methods return something but do not alter the original string)
  - `lower, replace, find`
- `for (index, value) in enumerate(sequence)`

Lecture 9

- How to open (and close!) a file to read it
- How to read line by line
- String methods split, trim, upper, and isupper

Lecture 10

- How to create a dictionary, add a key-value pair, update an existing key value pair, access a value given a key.
- Dictionary methods:
  - `pop, clear, keys, values, items`
- How to iterate through dictionaries (both simple for loop and (key, value) pair for all items)

Lecture 11

- Look at recursive functions we saw together and practice writing recursive functions using our usual recipe:
  - 1. Define what the function the name and parameters of the function
  - 2. Define the recursive case
    - Pretend you had a working version of your function, but it only works on smaller versions of your current problem.
    - The recursive problem should be getting "smaller", by some definition of smaller.
      - E.g., for smaller numbers (like in factorial), lists that are smaller/shorter, strings that are shorter
  - 3. Define the base case ‣ What is the smallest (or simplest) problem? This is often the base case
  - 4. Put it all together

Lecture 12

- The idea that a single neuron/perceptron has connecting inputs, $x_1, x_2, \ldots$ each contributing $x_i * w_i$
- The sum $\sum_i x_i * w_i$ will be compared against a threshold function.
  - If greater than or equal to threshold, the output is 1.
  - If less than than threshold, the output is 0.
- Go over practice example in slide 33-38.

Lecture 13

- Perceptron learning algorithm
  - Assume we want to train a neuron with n inputs, $x_1, x_2, \ldots, x_n$.
  - Start with adding an extra input neuron x_{n+1}whose input will always be 1.
  - Set threshold at 0.
  - Repeat until you get all examples right:
    - For each "training" example:
      - Calculate current prediction on example
      - If wrong:
      - Update weights and threshold towards getting this example correct.
        - $wi = wi + \Delta wi$
        - $\Delta wi = \lambda * (actual - predicted) * xi$, where λ is the learning rate.
- See practice example in slides 27–50.
- The perceptron learning algorithm will work if the training data is linearly separable (you can separate your data with a straight line). Counter-example: XOR

Lecture 14

- Given a training dataset with two labels (e.g., positive and negative reviews), know how to apply Naïve Bayes to classify a new data point (here, a review).
  - Start by calculating the conditional probabilities for each feature (here, a word) given a label.
  - Calculate the probability of the new data point given a label and pick the highest as the most likely label.
  - See example from slide 74 to 91.

Lecture 15

- An object is a software bundle of state (data/variables) and behavior (methods)
- A class is a blueprint for what data and methods objects should have
- You need to create an object, that is an instance of a class, using a constructor (through the magic method `__init__`)
  - `object_name = ClassName(potential_parameters)`
- `__str__` returns (doesn't print!) the string representation of the state of an object.
- `self` is a reference to the current object.
- We also use `self.variable` and `method()` to access an object's instance variables and methods *within* the class. `object.variable` and `object.method()` *outside* the class.
- `self` should be the first parameter in all methods

Lecture 16

- Optional parameters and how to use them.
- Look at the code and be familiar with how queues (FIFO) and stacks (LIFO) work and their methods
- Identity vs equality
  - `id` function and `is` keyword vs `__eq__` method and == operator.
- Look at the fruit.py as an example of how to practice writing classes and creating and using objects.

Lecture 17

- add the start state to to_visit
- Repeat
  - take a state off the to_visit list
  - if it's the goal state
  - we're done!
  - if it's not the goal state
  - Add all of the next possible states to the to_visit list
- Depth first search (DFS): to_visit is a stack
- Breadth first search (BFS): to_visit is a queue
- Know how to apply DFS and BFS on a graph

Lecture 18

- Look into implementation of BFS
- Look into implementation of DFS
  - Iterative variant using a stack
  - Recursive variant returning A solution
  - Recursive variant returning ALL solutions
- Syntax for creating, accessing, and manipulating matrices (see files in last slide).
  - Be careful with aliasing issues
  - Practice with visualization tool https://pythontutor.com/visualize.html#mode=edit
-

Lecture 21

- Trick to avoid repeats in DFS
- DFS and BFS are uninformed search algorithms
- We can use heuristics to bias our search towards the solution.
- Best first search is an example of informed search algorithms where the to_visit list is sorted based on some evaluation function and the most desirable state (rather than the first- or last-added like in BFS and DFS, respectively) is picked.
    - Best first search for sudoku example demonstrate their difference.
- Coming up with good heuristics is hard and can be computationally expensive.

Lecture 21

- Know the gist of minimax algorithm.
- Know how to apply it similar to slide 33-48.

Lecture 22

- Function `urlopen` from module `urllib.request` allows us to read webpages line by line.
  - We need to call `decode` to convert them from type `byte` to `str`.
- `string.find(substr, begin = 0, end = len(string))`returns the starting index in `string` that `substr` can be found in `string[begin:end]` or `-1` if not found.

Lecture 23 - Exceptions

- Exceptions are raised when certain types of error (such as passing empty lists, files not found, empty files, division with zero) occur.
- Be familiar with the `raise` and `try except` keywords and the flow of information if we don't catch them (prints error and terminates program) vs catch them (jumps to `except` block and then continues execution of the rest of the code)

Lecture 23 - Sets

- Sets are unordered collections of unique items
- Be familiar with how to construct sets (`{}` and `set` constructor), e.g., :
  - `my_set = {1, 2, 3, 4}`
  - `my_set = set([1, 2, 3, 4])`
    - `set` constructor can take any iterable object
- Basic set methods: `add, clear remove, pop, difference, intersection, union`
  - Are they accessors or mutators?
- Choose sets over lists when you don't care about ordering, duplicates.
- Sets are faster when it comes to questions of membership (keyword in)
  - About O(1) vs O(n) for lists
- Although are both O(n) for iteration, lists are *slightly* faster than sets.

Lecture 23

- Exceptions are raised when certain types of error (such as passing empty lists, files not found, empty files, division with zero) occur.
- Be familiar with the `raise` and `try except` keywords and the flow of information if we don't catch them (prints error and terminates program) vs catch them (jumps to `except` block and then continues execution of the rest of the code)

Lecture 24

- Functions are objects (class `function`)
- A higher order function is a function that takes a function as a parameter, or returns a function.
- Lambda functions are anonymous functions that can take multiple parameters and have one expression.
  - `lambda <inputs> : <expression>`
  - The inputs are passed and the expression is evaluated and returned.
- `map` function takes a function `f` and an iterable object `iter` as parameters and returns a `map` object of the results after applying `f` to each item of `iter`.
- `filter` function takes a function `f` that returns `bool` and an iterable object `iter` as parameters and returns a `filter` object of all items of `iter` for which `f`returned True.

Lecture 25

- Asymptotic analysis uses mathematical tools to find how algorithms scale as the problem input size grows.
- Big O provides an upper bound on the growth rate of the runtime (or memory) of an algorithm
- Common time complexities:
- $O(1)$ = constant
    - Doubling the input size, won't affect the run-time.
- $O(\log_2 n)$= logarithmic
    - Doubling the input size, will increase the runtime by a constant.
- $O(n)$= linear
    - Doubling the input size, will result to double the run-time.
    - single for loop
- $O(n^2)$ = quadratic
    - Doubling the input size, will result to quadrupling the run-time
    - nested for loop
- Time complexities for selection, insertion, and merge sort.

GOOD LUCK!