# CSSE1001: Sem. 1 2008 Practice exam answers

## Get more out of your study time. Join UQAttic's Semester 1, 2012 revision chat.

#### **[Other exam papers](https://docs.google.com/folder/d/0B_gnBD5zp3rqT2hrLWhvd2Rkc1U/edit)**

### Read this page first!

### Please contribute to these documents.

### If you're looking for an effective way to familiarise yourself with the course material, you can't go past collaborating with fellow students. We have laboured to put these up, and so at the very least point out where you think we are wrong!

### You'll get more out of the course, you'll do better in the exam, and other students will benefit from your input as well.

### Style.

### If you're unsure of your answer, highlight your answer text then hit Ctrl+Alt+M to create a comment beside the text. Once you're satisfied with the answer, click the "Resolve" button on the comment.

### If you want some extra explanation from someone else on their answer, highlight the other person's answer and repeat the procedure above.

### If you are confident that your answer is correct, then just type it in red below each question.

### Communicate.

### Head over to [uqattic.net](http://uqattic.net/) and click "Chat Now!". You'll find a chatroom full of students just like you. Talk about a revision document (like this one) or swap prep tips.

### If you have your own IRC client, point it to irc.uqattic.net, port 6667, channel #attic.

### You can disable automatic email alerts of comment discussions via the "Discussions" button.

Semester 1 of 2008 was the first time this course was taught in Python, and the exam was very different to previous semesters, so a practice exam was published to help students that were taking the course during that semester. A copy of it is available here: <https://docs.google.com/file/d/0B_gnBD5zp3rqQ1RiMURYZS1EWlk/edit>

The following answers were initially made by a past tutor of this course, the original version is published at<http://codefisher.org/csse1001/> with the disclaimer: “I don't guarantee the answers are 100% correct, however as someone who achieved a 7 for the subject, used Python regularly ever since, and tutored it 3 times I would suggest the chance of an error is small.”

Feel free to modify, add to, or comment on these answers.

1. b)

2. a) Python integer division always goes down, that is down as in the direction of negative

infinity.

3. c) Subtracting strings is nonsensical, so it gives an Error

4. d) You can't add (concatenate) int's with str's

5. c) Though maybe not perfectly intuitive, this can be done. It add the string to its self a given

number of times.

6. a) Counting from 0, S is the 7th character.

7. b) Starting at the 3rd character up to but not including the 6th

8. a) Starting at the ­8th character (the last one is the ­1th) going up to but not including the ­5th

9. d) Staring the the 7th character, going down to but not including the 4th in steps of ­1

10. c) The spaces are not including, and it is split on all occurrences

11. b) 16 and 17 are added onto the the end, then the second last ( now 16 ) is then removed

12. b) the setting a value a second time overwrites it.

13. e) using square brackets to get a non­existent value results in an error

14. c) The get function on dictionaries takes two arguments, the first is the key to find, the

second is the value to return if not found, it defaults to None. It never causes an error.

15. d) z will not be set if they are equal, so and error will be given when you try and return it.

16. a) after the inner functions are called it becomes f(0, 2) and then 1

17. c) The order which the arguments are give is import, not the names

18. c) the function checks if the numbers are even and only then increments r.

19. e) Since m is greater then the length of the list given as xs, it will run over the end of the list

which will cause and error.

20. a) n starts off greater then m so the loop will never run.

21. b) simply joins the two lists range(n) and ys into a single list of tuples (there is a built in

function called zip that does something like this).

22. e) The first list is longer then the second, which will result in an error when trying to get the

2nd element from ys

23. b) The first list is shorter, so not all the values from the second will be used.

24. c) We have seen 0 lines, and have no results yet. The semi­colon can be used to put multiple

statements on a line. Don't do this in your own code.

25. d) We have to convert the line to a float, and since the indexing must start from 0 (according

the comment) we can't want to increment the line count till after the line has been added.

26. d) From the loop invariant x < xs[end] must always be true, so we have to check this before

changing the end variable.

27. c) c and d are more accurate then a or b because it says the first argument is a reference to

the object. The first argument is always this, regardless of name (unless you deliberately do

some things your unlikely to have seen yet). data attributes are what your normally prefix

with self. Class variables are prefixed with the name of the class.

28. d) A method can do any or all of these things. But it does not have to.

29. d) Think about what happens if X is an int, and Y is a float, all except d) are true. Or

another example. If I wanted to have an Apple and Orange class there is not reason adding

them can't return FruitSalad. There is also a case for e) though if you know some of the

more obscure parts of the documentation.

30. b) constructor sets self.x to 2, and then f() adds 3 to it. g() returns this value (5) and the

\_\_str\_\_ function gets called by str() so the result will be the integer 5 converted to a string.

31. c) self.x in this case starts off as 0 then f() now sets it to the number squared, so 9

32. a) g() which is called \_\_str\_\_ now returns self.x times by ­2 and since self.x is 5 ­10

33. d) don't miss the 2\*x in the constructor, and now the result of g() (from the B2) which was

giving ­2\*self.x is not effectively 2\*self.x

34. c) If you have done the second assignment this should be quite obvious.

35. b) if you think about it this is n + n­1 + n­1 …. 2 + 1 which is what is wanted

36. b) a is wrong because it will consume the list (that is what .pop() does) c will not work

because it will not include the last element.

37. a) the first element shuffle(d) with all the permutations just computed.

38. c) this has to copy the list, the time taken will depend linearly on the length of the list.

39. d) The function for this would be roughly (x^2)/2 which is quadratic.

40. e) g() check if the letter is upper case, f() is just a function that returns “spam”. The list

comprehension simply says add f() to the list for every upper case letter in the string “Hello Word” which gives two spam's in a list.