# **CSSE1001: Sem. 2 2015 exam answers**

## [**UQAttic**](http://uqattic.net)

## **Get more out of your study time. Join UQAttic's revision chat.**

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

### 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.

### To get editing permissions, simply go to the [chatroom](http://uqattic.net) and provide us with your Google Account address.

### **Style.**

### Type answers in blue beneath each question.

### 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.

### **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.

If the answer is in blue, a tutor has confirmed this.

Q1. B&C

Q2. B

Q3. E

Q4. C

Q5. B

Q6. C

Q7. D

Q8. A

Q9. A

Q10. B

Q11. D

Q12. C

Q13. A

Q14. B

Q15. E

Q16. D

Q17. B

Q18. E

Q19. D

Q20. D

Q21. C

Q22. A

Q23. B

Q24. D

Q25. B

Q26. C

Q27. D

Q28. D

Q29. A

Q30. B

Q31. B

Q32. C

Q33. C

Q34. D

Q35. A

Q36. D

Q37. A

Q38. D

Q39. C

Q40. C

ANSWERS FROM BRAE WEBB (tutor) ON PIAZZA

**Question 1: B&C**

5.0 - 7/2  
5.0 - 3.5  
1.5

**Question 2: B**

list('a') + ['d', 'e']  
['a'] + ['d', 'e']  
['a', 'd', 'e']

**Question 3: E**

While both strings and lists have the ability to be multiplied, they can only be multiplied by an integer. In the question it tried to multiply a string by a list which causes an error. The examples below aim to demonstrate that string and lists can only be multiplied by an integer

>>> 'hello' \* 'hi'  
TypeError: can't multiply sequence by non-int of type 'str'  
>>> 'hello' \* 4  
'hellohellohellohello'  
>>> [1,2,3,4] \* [4,5,6,7]  
TypeError: can't multiply sequence by non-int of type 'list'  
>>> [1,2,3,4] \* 4  
[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]

**Question 4: C**

'54'+'28'  
'5428'

Just as we would expect for any string concatenation

name = 'John'  
'Hello, ' + name  
'Hello, ' + 'John'  
'Hello, John'

**Question 5: B**

Concatenation doesn't change the type of the items in a list, they retain their type but are concatenated.

**Question 6: C**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Forward Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | **9** | 10 | 11 | 12 | 13 | 14 |
| Reverse Index | -15 | -14 | -13 | -12 | -11 | -10 | -9 | -8 | -7 | **-6** | -5 | -4 | -3 | -2 | -1 |
| String | Q | u | i | c | k |  | b | r | o | **w** | n |  | f | o | x |

**Question 7: D**

**IMPORTANT: Slicing slices a list or string up to but not including the end index**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Forward Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | **12** | **13** | **14** |
| Reverse Index | -15 | -14 | -13 | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | **-3** | **-2** | **-1** |
| String | Q | u | i | c | k |  | b | r | o | w | n |  | **f** | **o** | **x** |

**Question 8: A**

Note: Leaving an index out of the index means go to the end, in the case of the second index (the end) if it is blank then go to the end of the sequence

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Forward Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | **12** | **13** | **14** |
| Reverse Index | -15 | -14 | -13 | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | **-3** | **-2** | **-1** |
| String | Q | u | i | c | k |  | b | r | o | w | n |  | **f** | **o** | **x** |

**Question 9: A**

The statement x[-5 : -8 : -1] is equivalent to start at the index -5 and decrease by one up to but not including the index -8.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Forward Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | **8** | **9** | **10** | 11 | 12 | 13 | 14 |
| Reverse Index | -15 | -14 | -13 | -12 | -11 | -10 | -9 | -8 | **-7** | **-6** | **-5** | -4 | -3 | -2 | -1 |
| String | Q | u | i | c | k |  | b | r | **o** | **w** | **n** |  | f | o | x |

**Question 10: B**

For this question our focus is only on the part of the statement which concerns y and not y[:] as y[:] makes a copy of y.

Thus we will only consider g(y)

g(['d','e','f'])  
def g(p):  
 w = p.pop(1) # removes the element at the 1st position ('e' since counting starts at 0) and sets w equal to it  
 p.extend(w) # adds the value set as w above to the end of the list (['d', 'f'].extend('e'))  
 return p # returns the altered p which will be ['d', 'f', 'e']

**Question 11: D**

Given that the way of understanding the function is handled above it won't be included here.

g(y).extend(g(y[:]))  
['d', 'f', 'e'].extend(g(y[:])) # y is now equal to ['d', 'f', 'e'] as it was given to the function directly (I'll use the direct ['d', 'f', 'e'] but it is associated with y  
['d', 'f', 'e'].extend(g(['d', 'f', 'e'])) # since the list was copied it will be given to the function and returned by y won't be modified  
['d', 'f', 'e'].extend(['d', 'e', 'f'])  
['d', 'f', 'e', 'd', 'e', 'f'] # remember that since I was using the first list in place of y we know that y has been modified here

**Question 12: C**

The important fact to remember for this question is that by default the split method splits strings at whitespaces (space, newline, tab, etc)

':'.join(z).split() 'a:b:c'.split() ['a:b:c'] # as there were no whitespaces in the string it just places the string in a list

**Question 13: A**

y.insert(3, y.pop(2))  
y.insert(3, 3) # 3 at index 2 was removed from y so y = [1,2,4]  
# 3 was inserted into y at index 3 which is the end of the list so y = [1,2,4,3]

**Question 14: B**

The important fact to remember for this question is that by default the split method splits strings at whitespaces (space, newline, tab, etc)

It is also good to note that \n stands for a newline and \t stands for a tab.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| g |  | r | **\n** | **\t** | c |  | m | p |

**Question 15: E**

In this question an error is raised because the dictionary does not contain the key 2 and it is using index which creates an error in the index is not found.

**Question 16: D**

In this question since the get method is being used an error is not raised, instead when the get method doesn't find a value it returns None.

d.get(0) # returns 'Zero'  
d.get(3) # returns 'Three'  
d.get(2) # returns None

Since the str of None is 'None' the concatenated results is 'ZeroThreeNone'

**Question 17: B**

The important fact in this question is that giving the get method a second positional parameter tells the get method to return that value if it can't find what you're looking for.

So when it couldn't find the index at 2 in the above question and returned None it will now return 'Unknown'

Thus the answer is 'ZeroThreeUnknown'

**Question 18: E**

An error will be raised as the function m is defined with only one positional argument and the question attempts to pass two.

**Question 19: D**

def m(x):  
 a,b = x # (3, 2) is assigned to a and b so that a=3 and b=2  
 if a > b: # 3 is greater than 2  
 return (b, a\*a-b\*b) # returns (2, 3\*3-2\*2) = (2, 5)  
 elif a < b:  
 return (a, b\*b-a\*a)  
 else:  
 return (a, b)

**Question 20: D**

def m(x):  
 a,b = x # (2, 1) is assigned to a and b so that a=2 and b=1  
 if a > b: # 2 is greater than 1  
 return (b, a\*a-b\*b) # returns (1, 2\*2-1\*1) = (1, 3)  
 elif a < b:  
 return (a, b\*b-a\*a)  
 else:  
 return (a, b)  
  
def m(x):  
 a,b = x # (1, 3) is assigned to a and b so that a=1 and b=3  
 if a > b: # 1 is not greater than 3  
 return (b, a\*a-b\*b)  
 elif a < b: # 1 is less than 3  
 return (a, b\*b-a\*a) # returns (1, 3\*3-1\*1) = (1, 8)  
 else:  
 return (a, b)

**Question 21: C**

t = []; s = 1  
n = 1; t = ['b']; n = 2; # 1\*(4-2) > 0  
n = 2; t = ['b', 'c']; n = 3; # 1\*(4-3) > 0  
n = 3; t = ['b', 'c', 'd']; n = 4 # 1\*(4-4) is not greater than 0

**Question 22: A**

t = []; s = -1  
-1\*(4-1) is not greater than 0

**Question 23: B**

t = []; s = -1  
n = -1; t = ['e']; n = -2 # -1\*(-4--2) > 0  
n = -2; t = ['e', 'd']; n = -3 # -1\*(-4--3) > 0  
n = -3; t = ['e', 'd', 'c']; n = -4 # -1\*(-4--4) is not greater than 0

**Question 24: D**

t = []; s = 1  
n = 1; t = ['b']; n = n + 3 = 4; # 1\*(5-4) > 0  
n = 4; t = ['b', 'e']; n = n + 3 = 7 # 1\*(5-7) is not greater than 0

**Question 25: B**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | r |
| Run | (1,'d') | (2,'e') | (3,'f') | (-1,'g') |  |
| 1 | i at 0, appends d and sets i=1 |  |  |  | ['d'] |
| 2 |  | appends e and sets i=3 |  |  | ['d','e'] |
| 3 |  |  |  | appends g and sets i=2 | ['d','e','g'] |
| 4 |  |  | appends f and sets i=5 which is longer than the list length so exit loop |  | ['d','e','g','f'] |

**Question 26: C**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | r |
| Run | (1,'d') | (-2,'e') | (3,'f') | (-2,'g') |  |
| 1 | i at 0, appends d and sets i=1 |  |  |  | ['d'] |
| 2 |  | appends e and sets i=-1 which is less than zero so loop exits |  |  | ['d','e'] |

**Question 27: D**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | r |
| Run | (1,'d') | (-1,'e') | (4,'f') | (-2,'g') |  |
| 1 | i at 0, appends d and sets i=1 |  |  |  | ['d'] |
| 2 |  | appends e and sets i=0 |  |  | ['d','e'] |
| 3 | appends 'd' and sets i=1 |  |  |  | ['d','e','d'] |
| 4 |  | appends e and sets i=0 |  |  | ['d','e','d','e'] |

The loop will continue to bounce between index 0 and 1. Which will cause an non-terminating error

**Question 28: D**

As \_\_init\_\_ is not overwritten or extended in B the statement b = B(3) will set self.x to 3.

The method g is overwritten in B so calling B.g calls B's method.

self.x + 2\*y  
3 + 2\*3  
9

**Question 29: A**

The trick to this question is that the x referenced in the function refers to the local x not the classes self.x so x=3 not 2

2\*x-1  
2\*3-1  
5

**Question 30: B**

The B class doesn't make any modifications to the f method or the \_\_init\_\_ method so the result will be the same as above.

2\*x-1  
2\*3-1  
5

**Question 31: B**

The C1 class extends the \_\_init\_\_ method, it calls the regular \_\_init\_\_ method on the first parameter which sets self.x to the first parameter but it also sets the second parameter to self.y.

The value passed to f is not relevant in the result as only the class variables are used not the parameters.

self.x + self.y  
3 + 2  
5

**Question 32: C**

C2 has the same setup as C1 but it changes the f method slightly to also add the parameter x to the equation.

x + self.x + self.y  
3 + 3 + 2  
8

**Question 33: C**

We have a class variable for BankAccount which stores the current balance (self.\_balance). We need to update this variable in the deposit method. Answer C is the only method which actually uses this variable and it correctly adds the given amount to the balance using +=.

self.\_balance += amount

**Question 34: D**

As mentioned above we have a class variable which stores the balance (self.\_balance). The docstring for the get\_balance method specifies that we need to return the value not just print it. Hence the correct solution is to return that variable.

return self.\_balance

**Question 35: A**

The deposit method is in the BankAccount class and since john is an instance of BankAccount we have access to the methods through that variable. So we call the deposit method on John which takes one parameter, the amount.

john.deposit(2.75)

**Question 38: D**

For time complexity we calculate an approximate formula for the time it takes to run a function and then take the most rapidly increasing subfunction as the indicator for Big O notation.

def has\_repeats(xs):  
 size = len(xs) # Constant time complexity = 1  
 for i in range(size-1): # Approximately linear time complexity = len(xs)  
 e = xs[i] # Constant time complexity = 1  
 for j in range(i+1, size): # This is the tricky part, as i increases by 1 the body of this statement runs 1 less time so approximately run half linear = len(xs)/2  
 if e == xs[j]: # Constant = 1  
 return True # Constant = 1  
 return False # Constant = 1

So from that information and remembering loops essentially multiply the inner statments we can work out an approximate formula for the program (I'm not sure this is accurate):

1 (The size = len(xs) statement is run only once)

+ len(xs) (This is from the e = xs[i] statement of constant complexity being run len(xs) times)

+ (len(xs)/2) \* (len(xs)) (From the constant if statement being run len(xs) times then len(xs)/2 times)

So since we are determining the complexity for length of the list, lets let len(xs) = n:

1 + n + n\*(n/2)

Now we work out what the most rapidly increasing subfunction is, we know that when n is really big 1 is negligible and the same thing happens when n is really really big for the n component and for the division by 2 of n.

Thus the most rapidly increasing subfunction is: n\*n = n^2

Thus, the time complexity is quadratic.

**Question 39: C**

I described tried to explain how the concept of time complexity works in the above question so look to there for guidance but note that it is not a topic this semester.

Here I will go over similar steps to develop an equation and then determine the biggest factor.

def isPalindrome(string):  
 size = len(string) # Constant time = 1  
 half = size/2 # Constant time = 1  
 i = 0 # Constant time = 1  
 while i < half: # Combined with the information that i increases by 1 in the body: linear time = len(string)/2  
 if string[i] != string[size-1-i]: # Constant time = 1  
 return False # Constant time = 1  
 i += 1 # Constant time = 1  
 return True # Constant time = 1

So from the above information and the fact we are determining for the length of the string we let len(string) = n:

Note that conditional statements of constant time (i.e. the return statements) are excluded, this is because they vary in execution and are at most a complexity of 1 so they aren't worth working out here.

1 + 1 + 1 + n/2 + n/2

Which has the most rapidly increasing factor of n which is linear.

Here there are some tricks that are designed to confuse us, firstly they have n/2 which as I briefly went over in the previous example does not actually have too much of a bearing on the big O notation, in fact even 0.00000001n would be considered simply n since when n gets ridiculously big it doesn't matter what coefficient it has.

Another trick to notice is that n/2 + n/2 is not the reason that this is linear complexity, it would still be linear complexity if it were just n.

**Question 40: C**

The following code is equivalent to the code in the question but may be easier to understand.

def g(x, y):  
 return (x-y)  
  
def f(x, y):  
 return x > y  
  
xs = [1,2,3,4]  
  
y = []  
  
for x in xs:  
 for y in xs:  
 if f(x, y)  
 y.append(g(x,y))

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | y | f(x,y) | g(x,y) | y |
| 1 | 1 | False |  |  |
| 1 | 2 | False |  |  |
| 1 | 3 | False |  |  |
| 1 | 4 | False |  |  |
| 2 | 1 | True | 1 | [1] |
| 2 | 2 | False |  |  |
| 2 | 3 | False |  |  |
| 2 | 4 | False |  |  |
| 3 | 1 | True | 2 | [1,2] |
| 3 | 2 | True | 1 | [1,2,1] |
| 3 | 3 | False |  |  |
| 3 | 4 | False |  |  |
| 4 | 1 | True | 3 | [1,2,1,3] |
| 4 | 2 | True | 2 | [1,2,1,3,2] |
| 4 | 3 | True | 1 | [1,2,1,3,2,1] |
| 4 | 4 | False |  |  |