

# Midterm1

● Graded

Student

Colin Cano

Total Points

8 / 25 pts

## Question 1

Expressions and the Python REPL

3 / 10 pts

1.1 `5 < P['bush'][0] / 7 < 6`

1 / 1 pt

✓ + 1 pt Correct

1.2 `T[1] / len(L[3])`

0 / 1 pt

✓ - 1 pt Wrong

1.3 `S[36:27:-3] + S[9:11] + S[48:44:-3]`

0 / 1 pt

✓ - 1 pt Wrong

1.4 `len(S.split()[5]) * str(T[-1][len(S.split()[3])+1])`

0 / 1 pt

✓ - 1 pt wrong

1.5 `T[1] + sum(T[2]) in T[-1]`

1 / 1 pt

✓ - 0 pts Correct

Well done!

1.6 `P[T[-1][9][:4].lower()][1]%L[0]`

1 / 1 pt

✓ - 0 pts Correct

Well done!

1.7 `P['reagan'][0] // T[1]`

0 / 1 pt

✓ - 0 pts Correct

✗ - 1 pt `p['reagan'] = (40)`  
The key to this problem is understanding the difference between (40) and (40,). (40) is just the INTEGER 40. (40,) is a TUPLE of length 1. You can't index an int, therefore it's an error

1.8 `len(' '.join(L[-2][0:2])) != int(43.2/6)`

0 / 1 pt

Question 1 (Parts 1-10)

✓ - 1 pt Incorrect

✗ ' '.join(L[-2][0:2]) will give you: 'iko\_day'.  
In which the length of that string is 7.

43.2 / 6 = 7.2, and then we are casting it into an int, which makes it 7.

So saying 7 != 7 would be FALSE

1.9 `str(T[2][-1])+str(L[0]/2)`

0 / 1 pt

Question 1 (Parts 1-10)

✓ - 1 pt Incorrect

✓ - 0.5 pts Half Correct

Not sure where you got 14.0, but [anything in between hard brackets should be an integer], as they are accessing values within sequences.

In which case, `T[2][-1]` should've been 11.

1.10 `(T[4][8]+P['bush'][1])%4/2`

0 / 1 pt

Question 1 (Parts 1-10)

✓ - 1 pt Incorrect

You have the right idea. But remember to follow PEMDAS in these examples

For instance, this would be:

`(15 + 43) % 4 / 2`

`=> 58 % 4 / 2`

`=> 2 / 2`

`=> 1.0`

## Question 2

`sandwich(S, i, j, U)`

1 / 3 pts

2.1 `sandwich(S, i, j, U)` arguments

1 / 1 pt

✓ - 0 pts Correct

2.2 `sandwich(S, i, j, U)`

0 / 2 pts

✓ - 0 pts perfect

✓ - 2 pts wrong

## Question 3

`uniqueEnd(S)`

3 / 4 pts

✓ - 1 pt Didn't check if `S[-1]` is a valid index. `S[-1]` will error if `s` is the empty string

## Question 4

`subsumes(w1, w2)`

1 / 4 pts

✓ - 1 pt Forgot to `lower()`

✓ - 2 pts Failed to use sets or poor use of set operations

### Question 5

vowelRatio(S)

0 / 4 pts

Right or Wrong

✓ - 3.5 pts Incorrect

Other

✓ - 0.5 pts Unknown Call

✎ unichar is going to just be a type. It does not represent an actual value.

We were looking for set and &, which would compare for values that appear in both sequences

You are not accounting for letters only, which you could get through '& set(alpha)'

using 'in' would return a boolean, making count either true or false.

Seat:

E16

Name:

Colin Cano

## CS1210 Computer Science I: Foundations

Exam 1

Monday, September 23, 2024

This test consists of 5 sections worth a total of 25 points. **Read each problem carefully.** Don't get stuck too long on any one question: you have 90 minutes. Please **write legibly**, and, where code indentation/alignment matter, **line things up cleanly and unambiguously!**

## 1. Expressions and the Python REPL [10 points]

Evaluate each of the following expressions and show the output that would be produced by the REPL. If the expression yields an error, say so and explain why. You may assume the following four statements have already been evaluated in the REPL:

T = ("Macky's back", -12, [3, 7, 11], "scarlet billows", {9:"Bush league", 8:15}).  
 S = "Midway upon the journey of [our life] I found myself within a forest dark."  
 L = [8, 'spy boy', ['your', 'grandma'], ('iko', 'day', 'hey', 'way'), 'fee nane']  
 P = { 'carter':(39), 'reagan':(40), 'bush':(41, 43), 'clinton':(42), 'ford':(38) }

>>> 5 < P['bush'][0]/7 < 6

True

>>> T[1]/len(L[3])

-1.0

>>> S[36:27:-3] + S[9:11] + S[48:44:-3]

'efileourpoesmy'

>>> len(S.split()[5]) \* str(T[-1][len(S.split()[3])+1])

Syntax error  
can't multiply string by  
string

>>> T[1]+sum(T[2]) in T[-1]

True

>>> P[T[-1][9][4].lower()][1]%L[0]

3

>>> P['reagan'][0]//T[1]

-3

*no dot key*  
 >>> len('.'.join(L[-2][0:2])) != int(43.2/6)  
*11 != 7.2*

*(3, 7, 1) + 8/2 = 4.0*  
 >>> str(T[2][-1]) + str(L[0]/2)  
*14.0*

*43004 10^3 3/2 = 1.5*  
 >>> (T[4][8] + P['bush'][1])%4/2  
*15 + 1.5*  
*16.5*

## 2. sandwich(S, i, j, U) [3 points]

Recall QotD4, where you were asked to write a function sandwich(S, i, j, U) which takes two sequences, S and U, and two non-negative integers i and j and returns a new sequence where the elements of U appear sandwiched in between the first i elements and the last j elements of S.

Our first solution attempt was:

```
def sandwich(S, i, j, U):
    return(S[:i] + U + S[-j:])
```

*Print 2, 3 Sandwich  
 Pascendwich*

Assuming we now restrict both i and j to be less than len(S), can you think of a set of arguments (values of S, i, j and U) that still cause this solution to fail? Explain the nature of the failure, if any.

*If j is 0, it will fail because there is not negative zero, so it would go from start and not the end*

If you have identified a failure, modify the given solution so that it now meets the specification.

```
def sandwich(S, i, j, U):
```

*# Use dots to keep your code aligned*

```
    .. return(S[:i] + U + (j > 0 and S[-j:]))
```

```
    ..
    ..
    ..
    ..
```

**3. uniqueEnd(S) [4 points]**

Specification: `uniqueEnd(S)` takes a sequence, `S`, and returns `True` if and only if the last element of sequence `S` is unique within `S`; otherwise, it returns `False`.

```
>>> uniqueEnd([1, 2, 1, 3, 1])
False           # There are several 1's
>>> uniqueEnd(())
False
>>> uniqueEnd(range(10))
True            # range(10)[-1] is 9, which is unique
>>> uniqueEnd('alberto')
True           # 'o' is unique in 'alberto'
```

return(  
 $S[-1]$   
 $s \leftarrow s + (s)[-1]$   
 $s[1, 2, 3]$   
 $S[-1]$   
 $S[-1]$   
 // not in

```
def uniqueEnd(S): # Use dots to keep your code aligned
```

```
.. return(S[-1] not in S[:-1]) ..
```

*For full credit, your solution should be legible, concise, elegant, properly indented, and should avoid extraneous statements or constructs not yet covered in class or the assigned readings.*

**4. subsumes(w1, w2) [4 points]**

Specification: `subsumes(w1, w2)` takes two strings, `w1` and `w2`, composed only of upper and lower case letters (no digits, spaces, or punctuation: these are essentially "words"), and returns `True` if either `w1` subsumes `w2` or `w2` is subsumed by `w1`. Here, "is subsumed by" means "has its elements contained entirely within the other, independent of order or case."

```
>>> subsumes('Alberto', 'tart')
True      # Because 'tart' is subsumed by 'Alberto'
>>> subsumes('file', 'fulfill')
False     # Because neither subsumes the other
```

*list(w1) in list(w2) or list(w2) in list(w1)*

```
def subsumes(w1, w2):
```

```
    .. return(list(w1) in list(w2) or list(w2) in list(w1)) ..
```

*For full credit, your solution should be legible, concise, elegant, properly indented, and should avoid extraneous statements or constructs not yet covered in class or the assigned readings.*



un: char in vowels

return(vowels/un:char)

### 5. vowelRatio(S) [4 points]

Specification: vowelRatio(S) takes a string, S, and returns an integer representing the ratio of unique vowels in S (where vowels are 'a', 'e', 'i', 'o', and 'u') to unique characters in S.

```
>>> vowelRatio('are')
66 # 2 (unique) vowels, 3 (unique) characters
>>> vowelRatio('vowel')
40 # 2 (unique) vowels, 5 (unique) characters
```

int( 66 / 100 )  
2/5 = 0.4

Note that duplicate characters are ignored in both the vowel count and the overall character count in S. Note also that the function signature provides for a second parameter, alpha, which defaults to a string containing all 26 lower case characters in the Roman alphabet. Finally, upper and lower case characters should be treated as being the same, and non-alphabetic characters should be ignored. So:

```
>>> vowelRatio('BookKeeper') == vowelRatio('bookkeeper')
True # case is ignored
>>> vowelRatio('Articolo 31')
42 # 3 vowels, 7 characters, excluding ' ', '3', and 'I'
```

0/0 unique vowels  
10/100

vowels divided characters

where Articolo 31 is, of course, the well known 90's Italian rap duo who released a new album in 2024, nearly 20 years after their last.

len(list(s)) = 3

```
def vowelRatio(w, alpha='abcdefghijklmnopqrstuvwxyz'): # Use dots to keep your code aligned
```

```
    .. vowels = 'a', 'e', 'i', 'o', 'u'
    .. un:char = len(list(s))
    .. return(int((vowels.in un:char) / un:char) * 100)
```

For full credit, your solution should be legible, concise, elegant, properly indented, and should avoid extraneous statements or constructs not yet covered in class or the assigned readings.

*feel free to use as scratch paper*