CIS 210 Winter 2019 Final Example Questions - KEY

Note: These questions are not a comprehensive study guide! They are given here to provide a sense of the types of questions that may be on the CIS 210 final exam.

To prepare thoroughly for the final exam you should review projects and project solutions, quizzes, midterm, class notes and exercises, lab notes and exercises, and readings and exercises from the text.

The final exam will comprise multiple choice questions, short-answer questions, and questions where the solution will require you to write Python code according to the usual CIS 210 style guidelines.

No outside resources are allowed during the exam, with the exception of one index card of handwritten notes.

Python is a programming language and Python is an interpreter program that translates Python code.

(0) In CIS 210, Python refers to

a) a programming language  b) a program that translates Python programs

C) both a) and b)  d) neither a) nor b)

Functions as parameters; mutable data types; side effects.

(1) Given the following Python code:

def q1a(i):
    '''(int) -> bool

        final exam function
        '''
    return i % 2 != 0

def q1b(i):
    '''(int) -> bool

        final exam function
        '''
    return i % 2 == 0

def q1(f, msg):
    '''(function, list) -> None

        final exam function
        '''
    for i in range(len(msg)):
        if f(i):
            msg[i] = msg[i].upper()
    return None
def main():
    '''final exam function'''
    li = ['hello', 'world', 'hello', 'CIS210']
    q1(q1a, li)
    q1(q1b, li)
    print(li)
    return None

What will be printed when >>> main() is executed?

a) ['hello', 'world', 'hello', 'CIS210']
b) ['hello', 'WORLD', 'hello', 'CIS210']
c) ['HELLO', 'world', 'HELLO', 'CIS210']
d) ['HELLO', 'WORLD', 'HELLO', 'CIS210']
e) ['Hello', 'World', 'Hello', 'CIS210']

Basic code tracing - type contract, arguments, what happens when a function is executed.

(2-8) Given the following Python code:

```python
1 def fb(n):
2     '''(int) -> ??-2
3     Play fizzbuzz up to n.
4     >>> fb(5)
5     1
6     2
7     fizz
8     4
9     buzz
10    Game over!
11    '''
12    for i in range(1, n+1):
13        m3 = (i % 3) == 0
14        m5 = (i % 5) == 0
15        if m3 and m5:
16            print('fizzbuzz')
17        elif m3:
18            print('fizz')
19        elif m5:
20            print('buzz')
21        else:
22            print(i)
23    print('Game over!')
24    return None
```
(2) Complete the type contract:

a) int  b) float  c) str  d) list  e) None

When `>>> fb(5)` is executed,

(3) the value of `i` after line 14 is executed (first time) is

a) 0  b) 1  c) 5  d) 6  e) NameError

(4) the value of `m5` after line 16 is executed (first time) is

a) 0  b) 1  c) True  d) False  e) NameError

(5) the value of `i` after line 14 is executed (last time) is

a) 0  b) 1  c) 5  d) 6  e) NameError

(6) the value of `m3` after line 15 is executed (last time) is

a) 0  b) 1  c) True  d) False  e) NameError

(7) If `n = 10` were added before line 14 and `>>> fb(5)` were called a second time

a) the result would be the same as the first time `>>> fb(5)` was executed (as in q3-6)

b) result would be different from the first time – more lines would be printed

c) result would be different from the first time – fewer lines would be printed

d) result would be different from the first time – more values would be returned

e) result would be different from the first time – fewer values would be returned

(8) Still including `n = 10` before line 14, `fb` is called a third time: `>>> fb(20)`

a) the result would be the same as the second time `>>> fb(5)` was executed in (as in q7)

b) result would be different from the second time – more lines would be printed

c) result would be different from the second time – fewer lines would be printed

d) result would be different from the second time – more values would be returned

e) result would be different from the second time – fewer values would be returned
Basic code tracing - what happens when a function is executed, namespaces.

(9-16) Given the following Python code:

```python
def dtob(n):
    '''(int) -> str
    Convert non-negative decimal integer n to a binary string.
    >>> dtob(4)
    '100'
    >>> dtob(0)
    '0'
    >>> dtob(27)
    '11011'
    '''
    if n == 0:
        b = '0'
    else:
        b = ''
        next_n = n
        while next_n > 0:
            r = next_n % 2
            b = str(r) + b
            next_n = next_n // 2
    return b
```

When `>>> dtob(6)` is executed,

(9) the function `dtob` is found in the ?? namespace.

a) local  b) global  c) built-in  d) file  e) NameError

(10) the value of `n` at line 13 is

a) 0  b) 1  c) 6  d) '6'  e) NameError

(11) the value of the expression `next_n > 0` at line 18 (first time) is

a) 0  b) 6  c) '6'  d) True  e) False

(12) the value of `b` after line 20 is executed (first time) is

a) ''  b) '0'  c) '1'  d) 0  e) 1

(13) the value of `next_n` after line 21 is executed (first time) is

a) 0  b) 1  c) 2  d) 3  e) 3.0
(14) the value of b at line 23 is

a) 0   b) 6   c) '100'   d) '110'   e) '111'

(15) When line 13 is executed, the value of n is found in the ?? namespace.

a) local   b) global   c) built-in   d) file   e) NameError

(16) When line 20 is executed, function str is found in the ?? namespace.

a) local   b) global   c) built-in   d) file   e) NameError

**k-means cluster analysis algorithm:**

(22-23) Given data values, an initial set of randomly chosen centroids, and a value for k:

data values = 2, 8, 6, 4, 10, 12, 20, 18, 16, 14

centroids = 2, 8, 14, 20

k = 4

22) After an initial round of executing the k-means cluster analysis algorithm, using Euclid distance as the similarity measure, the data will have clustered as (clusters are indicated by <>):

a) <2>, <4>, <6, 8, 10, 12, 14, 16, 18>, <20>   b) <2, 4, 6>, <8, 10, 12>, <14, 16, 18>, <20>

c) <2, 4>, <6, 8, 10>, <12, 14, 16>, <18, 20>   d) <2> <4, 6, 8>, <10, 12, 14> <16, 18, 20>

23) The second round of centroids will then be:

a) 2, 8, 14, 20   b) **3.0, 8.0, 14.0, 19.0**   c) 2.0, 6.0, 12.0, 18.0

d) 1.0, 7.0, 13.0, 19.0

**File processing.**

(24-26) Given the following Python code:

```python
testfile = 'test.txt'
testf = open(testfile)
testdata = testf.read()
testf.close()
```
Python assignment, reference semantics, immutable and mutable data types.

Given the following Python code:

```python
>>> x = 'hello, CIS 210'
>>> y = x
>>> z = 'hello, CIS 210'
```

(28) The type of the value returned by the expression `id(x) == id(y)` is

a) `int`  
b) `str`  
c) `boolean`  
d) `list`  
e) `dict`

(29) The value returned by the expression `id(x) == id(y)` is

a) `'hello, CIS 210'`  
b) 4390452976  
c) `True`  
d) `False`  
e) `{'hello': 'CIS 210'}`

(30) The value returned by the expression `id(x) == id(z)` is

a) `'hello, CIS 210'`  
b) 4390452976  
c) `True`  
d) `False`  
e) `{'hello': 'CIS 210'}`

(31) The value returned by the expression `x == z` is

a) `'hello, CIS 210'`  
b) 4390452976  
c) `True`  
d) `False`  
e) `{'hello': 'CIS 210'}`

(32-34) After additional Python code is executed:

```python
1 >>> x = x.split()
2 >>> y = x
3 >>> y[0] = y[0][:-1]
4 >>> y
5 >>> x
```
(32) The value of x after line 1 is executed is

a) 'hello'  
b) ['hello,', 'CIS', '210']  
c) ['hello', 'CIS 210']

d) ['hello', 'CIS', '210']  
e) None

(33) The value of y at line 4 is

a) 'hello'  
b) ['hello,', 'CIS', '210']  
c) ['hello', 'CIS 210']

d) ['hello', 'CIS', '210']  
e) None

(34) The value of x at line 5 is

a) 'hello'  
b) ['hello,', 'CIS', '210']  
c) ['hello', 'CIS 210']

d) ['hello', 'CIS', '210']  
e) None

tracing code; sequential search

(35-36) Given the following Python code:

def isIn(seq, t):
    """(sequence, item) -> boolean

    Search for item t in a sorted sequence, seq.
    Return True if t is a member, else False.

    >>> isIn((10, 20, 30, 40, 50, 60, 70, 80, 90), 80)
    True
    >>> isIn((10, 20, 30, 40, 50, 60, 70, 80, 90), 100)
    False
    """
    for item in seq:
        if item == t:
            return True
    return False

(35) When >>> isIn((10, 20, 30, 40, 50, 60, 70, 80, 90), 80) is executed, how many times is the for loop executed?

a) 0  
b) 1  
c) 8  
d) 9  
e) 10

(36) When >>> isIn((10, 20, 30, 40, 50, 60, 70, 80, 90), 100) is executed, how many times is the for loop executed?

a) 0  
b) 1  
c) 8  
d) 9  
e) 10
tracing code; binary search

(37-38) Given the following Python code:
def isInb(seq, t):
    '''(sequence, item) -> boolean

    Search for item t in a sorted sequence, seq. Return True if t is a member, else False.

    Note: seq is not maintained in its original form inside this function.

    >>> isInb((10, 20, 30, 40, 50, 60, 70, 80, 90), 80)
    True
    >>> isInb((10, 20, 30, 40, 50, 60, 70, 80, 90), 100)
    False
    '''
    if len(seq) == 0:
        return False
    while len(seq) > 0:
        mid = len(seq) // 2
        if seq[mid] == t:
            return True
        elif seq[mid] > t:
            seq = seq[:mid]
        else:
            seq = seq[mid+1:]
    return False

(37) When >>> isInb((10, 20, 30, 40, 50, 60, 70, 80, 90), 80) is executed, how many times is the while loop executed?

a) 0   b) 1   c) 2   d) 3   e) 10

(38) When >>> isInb((10, 20, 30, 40, 50, 60, 70, 80, 90), 100) is executed, how many times is the while loop executed?

a) 0   b) 1   c) 2   d) 3   e) 10
(39-47) Given the following Python code:

```python
def q39(s):
    '''(sequence) \rightarrow \text{??-39}
    test function.'''
    if len(s) == 0:
        return 0
    else:
        match = s[0]
        match_ct = 1
        longest = 1
        idx = 1
        while idx < len(s):
            nextch = s[idx]
            if nextch == match:
                match_ct += 1
            else:
                if match_ct > longest:
                    longest = match_ct
                match = nextch
                match_ct = 1
            idx += 1
        return max(match_ct, longest)
```

(39) Replace \text{??-39} with the appropriate value: \underline{\text{int} \underline{\underline{\text{int}}}}

When >>> q39('aabbcc') is executed,

(40) the value of \text{s} at line 6 is: \underline{\text{'aabbcc'}}

(41) the value of \text{match} after line 9 is executed is: \underline{\text{'a'}}

(42) the first time the \text{while} loop is executed, the value of \text{idx} at line 15 is: \underline{\underline{1}}

(43) the first time the \text{while} loop is executed, the value of \text{nextch} is: \underline{\text{'a'}}

(44) the first time the \text{while} loop is executed, the value of \text{nextch} == \text{match} is: \underline{\text{True}}

(45) when line 26 is executed, the value of \text{match}_ct is: \underline{\underline{1}}

(46) when line 26 is executed, the value of \text{longest} is: \underline{3}

(47) if line 14 were changed to \text{while idx} <= \text{len(s)}, this would cause an error at line: \underline{15}
Write a function, `word_ct`, to generate a frequency dictionary with the number of occurrences of each word in a string of words. `word_ct` will have one parameter, `text`, an unpunctuated (except for spaces between words) string of words, possibly empty. `word_ct` will return the dictionary of each word and the number of times it occurs in `text`. If `text` is an empty string, `word_ct` should return an empty dictionary.

For example,

```python
>>> word_ct('Monday morning Tuesday morning')
{'Monday': 1, 'morning': 2, 'Tuesday': 1}
```

The function should be written using CIS 210 style guidelines. The docstring examples of use should include at least one example with a "normal" value argument and at least one example with a "boundary" value argument. Python code should clearly reflect the underlying algorithm. Code should use the most appropriate Python tools for solving the problem.

```python
def word_ct(text):
    '''(str) -> dict

    Generate a frequency dictionary with number of occurrences of each word
    in (unpunctuated except for spaces) text.
    
    >>> word_ct('It was the best of times it was the worst of times')
    {'It': 1, 'was': 2, 'the': 2, 'best': 1, 'of': 2, 'times': 2, 'it': 1, 'worst': 1}
    >>> word_ct('')
    {}
    '''
    freqD = {}
    wordli = text.split()
    for word in wordli:
        if word in freqD:
            freqD[word] += 1
        else:
            freqD[word] = 1
    # better - use dict setdefault method
    for word in wordli:
        freqD.setdefault(word, 0)
        freqD[word] += 1
    return freqD
```
Write a function, `find_last`, which returns the position of the last occurrence of a character in a string. `find_last` will have two parameters, `s`, a string, and `ch`, a single character string. `find_last` will return the position of the last occurrence of `ch` in `s`, or -1 if `ch` does not occur in `s`.

For example,

```python
>>> find_last('mississippi', 'i')
10
```

The function should be written using CIS 210 style guidelines. The docstring examples of use should include at least one example with a "normal" value argument and at least one example with a "boundary" value argument. Python code should clearly reflect the underlying algorithm. Code should use only the most appropriate Python "tools" for solving the problem.

```python
def find_last(s, ch):
    '''(str, str) -> int
    return position of last occurrence of ch s, or -1 if ch does not occur in s
    >>> find_last('', '')
    -1
    >>> find_last('', 'a')
    -1
    >>> find_last('a', 'a')
    0
    >>> find_last('a', 'b')
    -1
    >>> find_last('aaa', 'a')
    2
    >>> find_last('abb', 'a')
    0
    >>> find_last('example', 'm')
    3
    >>> find_last('mississippi', 'i')
    10
    '''
    lastpos = -1
    for idx in range(len(s)):
        if s[idx] == ch:
            lastpos = idx
    return lastpos
```

```python
def find_last(s, ch):
    '''another approach - no need for lastpos var this way'''
    for idx in range(len(s)-1, -1, -1):
        if s[idx] == ch:
            return idx
    return -1
```