CIS 210 Fall 2019 Example Exam 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 will be on the CIS 210 final exam.

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

The final exam will be given in the regular classroom, and 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 3x5” index card of handwritten notes.

(1) What is the result of executing q1()? 

VOWELS = 'aeiou'
CONSONANTS = 'bcdfghjklmnpqrstuvwxyz'

def q1aux(st):
    v = ''
    c = ''
    for ch in st:
        if ch in VOWELS:
            v += ch
        elif ch in CONSONANTS:
            c += ch
    return v + c

def q1():
    print(q1aux('any message here'))  #STILL NEEDS TO BE FIXED
    return None

a) ''  b) None  c) 'CIS 210'  d) 'ICS 210'  e) 'ICS'

(2-5) Given the following Python code:

```python
>>> for bit in '10':
    print(bit)
    print(bit in '01')
```

(2) The value of bit the first time the for loop is executed is 

a) '1'  b) '0'  c) '10'  d) True  e) False

(3) The value of bit in '01' (second print) the first time the for loop is executed is 

a) '1'  b) '0'  c) '10'  d) True  e) False
(4) The value of bit the second time the for loop is executed is

a) '1'  b) '0'  c) '10'  d) True  e) False

(5) The value of bit in '01'(second print) the second time the for loop is executed is

a) '1'  b) '0'  c) '10'  d) True  e) False
The Taylor series expansion for $e^x$ has the following form:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

If we truncate the sum after $N$ terms, we have an approximation for $e^x$ as:

$$e^x \approx \sum_{n=0}^{N-1} \frac{x^n}{n!}$$

Given the following Python code:

```python
from math import factorial  # factorial(n) returns n!

def approx_e(x, terms):
    '''(number, int) -> float

    Generate approximation for e**x with
    n iterations of the Taylor series
    expansion for e**x. The approximate
    value of e**x is returned.
    
    >>> approx_e(1, 100)
    2.7182818284590455
    ...'''
    ae = 0
    x_to_power = 1
    for n in range(terms):
        ae += x_to_power/factorial(n)
        x_to_power *= x
    return ae
```

(6) Complete the docstring for `approx_e` below, consistent with this approximation for $e^x$.

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

(7) `approx_e` is an example of a(n) ?? pattern:

a) accumulator    b) filter    c) map    d) return
(8) What is the result of executing >>> q8(75)?

def q8(temp):
    ''' No docstring on the exam '''
    message = ''

    if temp >= 90:
        message = 'very hot'

    if temp >= 80:
        message = 'hot'

    if temp >= 70:
        message = 'ok'

    if temp >= 60:
        message = 'cool'

    return message

a) 'very hot'  b) 'hot'  c) 'ok'  d) 'cool'  e) error

(9) What is the result of executing q9()? 

def q9():
    x = 5678
    y = 0
    while x > 0:
        y += 1
        x = x // 10
    return y

a) 4  b) 3  c) 567  d) 8  e)'5678'
What is the result of executing the following UNTESTED Python code?

```python
>>> universities = ['UO', 'OSU', 'SOU']
>>> new = 'WOU'
>>> universities.append(new)
>>> print(universities)
```

(10)

a) ['UO', 'OSU', 'SOU']     b) ['UO', 'OSU', 'SOU', 'WOU']

Continuing in the same interactive Shell session:

```python
>>> new = 'PSU'
>>> universities = universities.append(new)
>>> print(universities)
```

(11)

a) ['UO', 'OSU', 'SOU']     b) ['UO', 'OSU', 'SOU', 'WOU', 'PSU']

c) ['PSU', 'WOU', 'UO', 'OSU', 'SOU']     d) None

Given the following Python code:

```python
>>> import math
>>> from math import pi
>>> dir()
```

Which of the following would you expect to see in the __main__ namespace after `dir()` is executed?

a) import     b) math     c) sqrt     d) math.pi     e) math.sqrt

A dependent child can be very loosely defined as a person under 18 years of age who does not earn $10,000 or more a year. An expression that would define a dependent child is

a) `(age < 18) and (salary < 10000)`

b) `(age < 18) or (salary < 10000)`

c) `(age <= 18) and (salary < 10000)`

d) `(age <= 18) or (salary <= 10000)`
Given:

```python
import random

def createCentroids(k, datadict):
    '''(int, dict) -> list (of dict values)

    Create a starter list of k centroids
    for a k-cluster algorithm by
    randomly choosing from the items
    in datadict. The starter list is returned.
    '''
    centroids = []
    centroidCount = 0
    centroidKeys = []

    while centroidCount < k:
        rkey = random.randint(1, len(datadict))
        if rkey not in centroidKeys:
            centroids.append(datadict[rkey])
            centroidKeys.append(rkey)
            centroidCount += 1

    return centroids
```

```python
>>> d = {1: [1.1], 2: [2.9], 3: [3.4], 4: [2.7]}
>>> d[4] = [4.4]

Which could NOT be the result of executing  >>> createCentroids(2, d)

a) [[3.4], [2.9]]  
b) [[2.9], [4.4]]  
c) [[2.7], [2.9]]  
d) [[3.4], [4.4]]  
e) [[1.1], [2.9]]
```
Given the following UNTESTED Python code:

```python
def q15(astring):
    '''(str) -> ??

    Exam function
    '''
    symbols = ['@', '#', '$', '%', '&', '*']
symbols_ctr = 0

    for c in astring:
        if c in symbols:
            symbols_ctr += 1

    return (symbols_ctr >= 2)
```

(15) Complete the type contract:

a) int  b) str  c) bool  d) list  e) dict

(16) What will be the result of executing ```>>> q15('CIS210')```?

a) 0  b) 1  c) True  d) False  e) 'CIS21'

(17) Which test will reveal the bug in the code?

a) q15('CIS210')  b) q15('CIS&210')  c) q15('CIS210*')

d) q15('#CIS210')  e) q15('**CIS210**')

(18) What kind error is this?

a) documentation  b) syntax  c) runtime  d) logical
(19) The following Python code

```python
>>> x = 'hi'
>>> x = 0
>>> x = x < 0
```
demonstrates which characteristic of Python?

- a) strong typing
- b) dynamic typing
- c) operator overloading
- d) loops
- e) conditionals

(20) The following Python code

```python
>>> x = 'hi' + '-' + 'bye'
>>> y = 99 + 100
```
demonstrates which characteristic of Python?

- a) strong typing
- b) dynamic typing
- c) operator overloading
- d) loops
- e) conditionals

(21) The following Python code

```python
>>> x = 'hi' + 99
Traceback (most recent call last):
  File "<pyshell#41>", line 1, in <module>
    x = 'hi' + 99
TypeError: must be str, not int
```
demonstrates which characteristic of Python?

- a) strong typing
- b) dynamic typing
- c) operator overloading
- d) loops
- e) conditionals
(22-23) After the following Python code is executed:

```python
>>> mystr = 'hello, world'
>>> yourstr = mystr
>>> mystr = 'good morning, world'
```

(22) The value of `mystr` is:

a) 'hello, world'       b) 'good morning, world'       c) yourstr

d) ''       e) error

(23) The value of `yourstr` is:

a) 'hello, world'       b) 'good morning, world'       c) yourstr

d) ''       e) error

(24-29) After the following Python code is executed:

```python
>>> mylist = [[1, 2], 'CIS210', 99.9, 100]
>>> yourlist = mylist
>>> mylist[1] = 'CIS211'
```

(24) The value of `mylist` is:

a) [[1, 2], 'CIS210', 99.9, 100]

b) ['CIS211', 'CIS210', 99.9, 100]

c) [[1, 2], 'CIS211', 99.9, 100]

d) [[1, 2], 'CIS210', 'CIS211', 100]

e) [[1, 2], 'CIS210', 99.9, 100, 'CIS211']

(25) The value of `yourlist` is:

a) [[1, 2], 'CIS210', 99.9, 100]

b) ['CIS211', 'CIS210', 99.9, 100]

c) [[1, 2], 'CIS211', 99.9, 100]

d) [[1, 2], 'CIS210', 'CIS211', 100]

e) [[1, 2], 'CIS210', 99.9, 100, 'CIS211']
In the same interactive Shell session, after the following Python code is executed:

```python
>>> next = mylist.pop()
```

(26) The value of `mylist` is:

a) `[[1, 2], 'CIS210', 99.9]`  
b) `['CIS211', 'CIS210', 99.9]`

c) `[[1, 2], 'CIS211', 99.9]`  
d) `[1,2]`  
e) `100`

(27) The value of `yourlist` is:

a) `[[1, 2], 'CIS210', 99.9]`  
b) `['CIS211', 'CIS210', 99.9]`

c) `[[1, 2], 'CIS211', 99.9]`  
d) `[1,2]`  
e) `100`

In the same interactive Shell session, after the following Python code is executed:

```python
>>> yourlist = mylist.pop()
```

(28) The value of `mylist` is:

a) `[[1, 2], 99.9]`  
b) `['CIS210', 99.9]`  
c) `[[1, 2], 'CIS211']`

d) `[1,2]`  
e) `99.9`

(29) The value of `yourlist` is:

a) `[[1, 2], 99.9]`  
b) `['CIS210', 99.9]`  
c) `[[1, 2], 'CIS211']`

d) `[1,2]`  
e) `99.9`
(30-32) Given function `lowtemps` and file `'lowtemps.txt'`, with a record of the daily low temperature in Eugene from February 1 through February 10:

'lowtemps.txt' has two lines: a header line, and another line of comma separated values, which are the daily rainfall amounts:

```python
#low temperature data final exam CIS 210 W19 02/01-15/2019 Eugene 43,35,37,29,29,28,26,30,29,29
def lowtemps(f):
    '''Exam function
    '''
    with open(f) as tempf:
        tempf.readline()
        temps = tempf.readline().strip().split(',')

        tempd = {}
        day = 1
        for temp in temps:
            tempd[day] = int(temp)
            day += 1

        return tempd
```

(30) The correct type contract for `lowtemps` is:

a) (file) -> file   b) (int) -> file   c) (int) -> float
   d) (str) -> list   e) (str) -> dict

(31) The value of `temps` after `temps = tempf.readline().strip().split(',')` is executed:

a) '#low temperature data final exam CIS 210 W19 02/01-15/2019 Eugene'

b) '43,35,37,29,29,28,26,30,29,29'

c) ['43', '35', '37', '29', '29', '28', '26', '30', '29', '29']

d) {1: 43, 2: 35, 3: 37, 4: 29, 5: 29, 6: 28, 7: 26, 8: 30, 9: 29, 10: 29}

e) {43: 1, 35: 1, 37: 1, 29: 4, 28: 1, 26: 1, 30: 1}
(32) The value of tempd at the return statement is

a) 'low temperature data final exam CIS 210 W19 02/01-15/2019 Eugene'

b) '43, 35, 37, 29, 29, 28, 26, 30, 29, 29'

c) ['43', '35', '37', '29', '29', '28', '30', '29', '29']

d) {1: 43, 2: 35, 3: 37, 4: 29, 5: 29, 6: 28, 7: 26, 8: 30, 9: 29, 10: 29}

e) {43: 1, 35: 1, 37: 1, 29: 4, 28: 1, 26: 1, 30: 1}

(33-37) Given the following Python code:

```python
import math
def euclidD(point1, point2):
    '''(tuple, tuple) -> float

    Computes and returns the Euclidean distance between point1 and point2 (which must have same dimensions).
    >>> euclidD([3,0], [0,4])
    5.0
    '''
    total = 0
    for index in range(len(point1)):
        diff = (point1[index] - point2[index]) ** 2
        total = total + diff
    euclidDistance = math.sqrt(total)
    return euclidDistance
```

And the following UNTESTED main function:

```python
def main():
    '''driver'''
    points = ((1, 2), (3, 4), (5, 6))
    origin = (1, 1)
    for point in points:
        result = euclidD(origin, point)
        print(origin, point, result, point1, point2)

    return None
```
(33) The first time `euclidD` is called in `main`, the value of `point1` (in `euclidD`) is:

a) (1, 1)  
b) (1, 2)  
c) (3, 4)  
d) 1.0  
e) no value/NameError

(34) The first time `euclidD` is called in `main`, the value of `point2` (in `euclidD`) is:

a) (1, 1)  
b) (1, 2)  
c) (3, 4)  
d) 1.0  
e) no value/NameError

(35) The first time the `print` statement is called in `main`, the value of `origin` is:

a) (1, 1)  
b) (1, 2)  
c) (3, 4)  
d) 1.0  
e) no value/NameError

(36) The first time the `print` statement is called in `main`, the value of `result` is:

a) (1, 1)  
b) (1, 2)  
c) (3, 4)  
d) 1.0  
e) no value/NameError

(37) The first time the `print` statement is called in `main`, the value of `point1` is:

a) (1, 1)  
b) (1, 2)  
c) (3, 4)  
d) 1.0  
e) no value/NameError

(38-40) Given the following Python code:

```python
def q38r(s):
    '''exam function'''
    if s == '':
        return 0
    elif len(s) == 1:
        return int(s)
    else:
        return int(s[0]) + q38r(s[1:])
>>> q38r('12345')
```

(38) The second time `q38r` is called, the value of `s` will be

a) '12345'  
b) '2345'  
c) '1234'  
d) '345'  
e) ''

(39) Including the initial call to `q38r('12345')`, `q38r` will be called ?? times in all.

a) 1  
b) 2  
c) 3  
d) 4  
e) 5

(40) The value returned by `q38r('12345')` will be

a) '1'  
b) '15'  
c) 1  
d) 15  
e) None