basic code execution

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

VOWELS = 'aeiou'
CONSONANTS = 'bcdfghjklmnpqrstvwxyz'

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('CIS 210'))
    return None

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

code execution - differentiate for loop and Boolean expression

(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
'''
    acc = 0
    x_to_power = 1
    for n in range(terms):
        acc += x_to_power / factorial(n)
        x_to_power *= x
    return acc
```

(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'
lists; mutable data type; many list methods update the list as a side effect and return None

(10-11) 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']
c) ['WOU', 'UO', 'OSU', 'SOU']    d) None

Continuing in the same interactive Shell session:

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

(11)
a) ['UO', 'OSU', 'SOU']    b) ['UO', 'OSU', 'SOU', 'WOU']
c) ['WOU', 'UO', 'OSU', 'SOU']    d) None

namespaces and import

(12) 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

boolean expression

(13) 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) \((\text{age} < 18) \text{ or } (\text{salary} < 10000)\)

c) \((\text{age} \leq 18) \text{ and } (\text{salary} < 10000)\)

d) \((\text{age} \leq 18) \text{ or } (\text{salary} \leq 10000)\)

*executing code – data mining project*

(14) 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

```python
>>> createCentroids(2, d)
```

a) [[4.4], [2.9]]  
 b) [[2.9], [4.4]]  
 c) [[2.7], [2.9]]  
 d) [[3.4], [4.4]]  
 e) [[1.1], [4.4]]
Python block syntax; errors and testing

(15-18) 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*')`

b) `q15('#CIS210')`  e) `q15('**CIS210**')`

(18) What kind of error is this?

a) documentation  b) syntax  c) runtime  d) logical
dynamic typing; strong typing; overloaded operators

(19) The following Python code
>>> 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
>>> 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
>>> 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
what happens when an assignment statement is executed (reference semantics); immutable data type

(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

what happens when an assignment statement is executed (reference semantics); mutable data type; list methods update list as a side effect

(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]

(27) The value of `yourlist` is:

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

(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
executing code – file processing; type contracts; dictionaries

(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:

```
#low temperature data final exam CIS 210 W19 02/01-15/2019 Eugene
43,35,37,29,29,28,26,30,29,29
```

```python
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', '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}

namespaces; variable scope

(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).

    Called by: createClusters
    
    >>> 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(result, point1, point2)
```

return None
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

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

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

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

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

executing code; recursion

(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')
```

The second time q38r is called, the value of s will be

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

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

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

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

- a) '1'
- b) '15'
- c) 1
- d) 15
- e) None
Create name and id, both of type str, and assign them to your name and student id number:

name = 'Solution'  id = '952123456'

strings; lists; list comprehension

Replace the ?? with the result of executing the following code (indicate 'error' if the result would be a Python error message):

```python
>>> phrase = 'The quick brown fox'
>>> vowels = 'aeiou'
>>> s = phrase.split()
>>> s
['The', 'quick', 'brown', 'fox']
>>> type(s)
list
>>> v = [x for x in phrase if x in vowels]
>>> v
['e', 'u', 'i', 'o', 'o']
>>> type(v)
list
```

revising code

Mark the code to indicate changes that are needed.

Revise isInCircle to check whether point x, y is located in a circle centered at (0,0) with any radius (don't forget docstring changes):

```python
def isInCircle(x, y, r):
    '''(number, number, number) -> bool

    Returns True if point (x, y) is in the circle with radius r.
    
    >>> isInCircle(.5, .5, 1)
    True
    >>> isInCircle(1, 2, 1)
    False
    '''
    d = math.sqrt(x**2 + y**2)
    return d <= r
```
(SA4) Revise `montePi` to call revised `isInCircle` to determine whether point x, y lands on the dart board (i.e., upper right sector of a circle with origin 0,0 and radius 1):

```python
import math

def montePi(numDarts):
    
    (integer) -> float

    Uses a Monte Carlo algorithm (looping numDarts times) to generate an approximate value for pi.

    Due to randomness in the algorithm, function output may vary somewhat from function call to function call.

    >>> montePi(1000)
    3.168
    >>> montePi(1000000)
    3.141068
    ''
    inCircle = 0

    # throw the darts and track whether they landed on the dart board.

    for i in range(numDarts):
        x = random.random()
        y = random.random()

        **d = math.sqrt(x**2 + y**2)**

        if isInCircle(x, y, 1): #was: if d <= 1
            inCircle += 1

    approxPi = inCircle/numDarts * 4

    return approxPi
```

**strings (sequential data type)**

(SA5) [2 pts.] Replace the ?? with the result of executing the following code (indicate 'error' if the result would be a Python error message):

```python
>>> name = "Prince Lucien Campbell"
>>> name = name.split()

>>> new = name[0][0] + name[1][0] + name[2][0]
>>> new
'PLC'
```
executing code

(SA6) Given the following Python code:

```python
1  def q6(astr):
2      '''(str) -> (list)
3      Exam function.
4      '''
5      countdict = {}
6      for item in astr:
7          if item in countdict:
8              countdict[item] += 1
9          else:
10             countdict[item] = 1
11
12      countlist = countdict.values()  
13      maxcount = max(countlist)
14      mli = []
15      for item in countdict:
16          if countdict[item] == maxcount:
17              mli.append(item)
18      '''
19      mli = [item for item in countdict if countdict[item] == maxcount]
20      '''
21      return mli
22
>>> teststr = 'zzzyxxxxww'
```

When `>>> q6(teststr)` is executed:

the value of `countdict` at line 11 is: `{'z': 4, 'y': 1, 'x': 4, 'w': 2}`

the value of `countlist` after line 12 is executed is: `[4, 1, 4, 2]`

the value of `maxcount` at line 14 is: 4

the value of `mli` at line 22 is: `['z', 'x']`
a structured approach to designing and implementing code

(C7) Write a function, `mypow`, that returns $x$ to the $n$th power, where $x$ and $n$, non-negative integers, are function parameters. Do not use Python's built-in power operator (** or function `pow`).

a) Write a simple example of use for `mypow`:
   
   ```python
   mypow(3, 2)
   9
   ```

b) Write one edge test case for `mypow`:
   
   ```python
   mypow(2, 0)
   1
   ```

c) Write the function header for `mypow`:
   
   ```python
def mypow(x, n):
```

d) Write the type contract for `mypow`:
   
   ```python
   (int, int) -> int
   ```

e) Write an algorithm for solving `mypow` (natural language/pseudocode):

   ```
multiply x by itself n times, e.g, 2 to 3rd is 2 * 2 * 2
   ```

f) Write the rest of (i.e., do not include header and docstring) the Python code for `mypow`:

   ```python
   result = 1
   for i in range(n):
       result = result * x
   return result
   ```
Write a program to print a message in an interesting format. Function `spring` will have one parameter, `greeting`, a string. It should return the greeting with every other word, starting with the first word, in uppercase, and the alternating words in lowercase. For example:

```python
>>> spring('Best wishes for a pleasant spring break.')
'BEST wishes FOR a PLEASANT spring BREAK.'
```

Then write a main function that calls `spring` and prints the resulting message. The program should be written using CIS 210 style guidelines. Function `spring` should include one “normal” and one “edge” example of use in the docstring, along with the type contract. (You may omit the brief description in the docstring of `spring`. The main function does not need a docstring.)

def spring(greeting):
    '''(str) -> str

    >>> spring('Best wishes for a pleasant spring break.')    # normal
    'BEST wishes FOR a PLEASANT spring BREAK.'
    >>> spring('')    # edge
    ''
    ...
    newgr = ''
    uc = True

    words = greeting.split()
    for word in words:
        if uc:
            word = word.upper()
            uc = False
        else:
            word = word.lower()
            uc = True

        newgr += word + ' ' 
    return newgr[:-1]    # return newgr.strip()
```
def main():
    '''driver'''
    print(spring('Best wishes for a pleasant spring break.'))
    return None