### Midterm Exam 1, Sample 1

**1. Expressions**
For each expression in the left-hand column, indicate its value in the right-hand column.
Be sure to list a constant of appropriate type (e.g., 7.0 rather than 7 for a float, Strings in quotes).

Expression Value

8 + 5 \* 3 / 2 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1.5 \* 4 \* 7 // 8 + 3.4 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

73 % 10 - 6 % 10 + 28 % 3 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4 + 1 + 9 + (-3 + 10) + 11 // 3 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3 // 14 // 7 / (1.0 \* 2) + 10 // 6 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

10 > 11 == 4 / 3 > 1 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

not(2 >= 11 or 10 < 67 or 4 / 4 >= 1) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(True or not 2 < 3) and 6 == 4 / 3 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**2. Parameter Mystery**
At the bottom of the page, write the output produced by the following program.

def main():

 x = "happy"

 y = "pumpkin"

 z = "orange"

 pumpkin = "sleepy"

 orange = "vampire"

 orange(y, x, z)

 orange(x, z, y)

 orange(pumpkin, z, "y")

 z = "green"

 orange("x", "pumpkin", z)

 orange(y, z, orange)

def orange(z, y, x):
 print(y + " and " + z + " were " + x)

**3. If/Else Simulation**For each call of the function below, write the value that is returned:

def mystery(n):

 if (n < 0):

 n = n \* 3

 return n

 else:

 n = n + 3

 if (n % 2 == 1):

 n = n + n % 10

 return n

Function Call Value Returned

mystery(-5) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

mystery(0) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

mystery(7) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

mystery(18) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

mystery(49) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**4. Programming**
Write a function named month\_apart that accepts four integer parameters representing two calendar dates. Each date consists of a month (1 through 12) and a day (1 through the number of days in that month [28-31]). Assume that all dates occur during the same year. The method returns whether the dates are at least a month apart. For example, the following dates are all considered to be at least a month apart from 9/19 (September 19): 2/14, 7/25, 8/2, 8/19, 10/19, 10/20, and 11/5. The following dates are NOT at least a month apart from 9/19: 9/20, 9/28, 10/1, 10/15, and 10/18. Note that the first date could come before or after (or be the same as) the second date. Assume that all parameter values passed are valid.

 Sample calls:

month\_apart( 6, 14, 9, 21) should return True, because June 14 is at least a month before September 21

month\_apart( 4, 5, 5, 15) should return True, because April 5 is at least a month before May 15

month\_apart( 4, 15, 5, 15) should return True, because April 15 is at least a month before May 15

month\_apart( 4, 16, 5, 15) should return False, because April 16 isn't at least a month apart from May 15

month\_apart( 6, 14, 6, 8) should return False, because June 14 isn't at least a month apart from June 8

month\_apart( 7, 7, 6, 8) should return False, because July 7 isn't at least a month apart from June 8

month\_apart( 7, 8, 6, 8) should return True, because July 8 is at least a month after June 8

month\_apart(10, 14, 7, 15) should return True, because October 14 is at least a month after July 15

**5. Programming**
Write a function named print\_grid that accepts two integer parameters *rows* and *cols*. The output is a comma-separated grid of numbers where the first parameter (*rows*) represents the number of rows of the grid and the second parameter (*cols*) represents the number of columns. The numbers count up from 1 to (*rows* x *cols*). The output are displayed in column-major order, meaning that the numbers shown increase sequentially down each column and wrap to the top of the next column to the right once the bottom of the current column is reached.

Assume that *rows* and *cols* are greater than 0. Here are some example calls to your function and their expected results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Call** | print\_grid(3, 6) | print\_grid(5, 3) | print\_grid(4, 1) | print\_grid(1, 3) |
| **Output** | 1, 4, 7, 10, 13, 162, 5, 8, 11, 14, 173, 6, 9, 12, 15, 18 | 1, 6, 112, 7, 123, 8, 134, 9, 145, 10, 15 | 1234 | 1, 2, 3 |

**6. Programming**
Write a function named count\_even\_digits that accepts two integers as parameters and returns the number of even-valued digits in the first number. An even-valued digit is either 0, 2, 4, 6, or 8. The second value represents how many digits the number has. The second value is guaranteed to match the number of digits in the first number.

For example, the number 8546587 has four even digits (the two 8s, the 4, and the 6),
so the call count\_even\_digits(8346387, 7) should return 4.

 You may assume that the values passed to your function are non-negative.

### Midterm Exam 1, Sample 1 Solutions

1. Expressions

|  |  |
| --- | --- |
| Expression8 + 5 \* 3 / 2 1.5 \* 4 \* 7 // 8 + 3.4 73 % 10 - 6 % 10 + 28 % 3 4 + 1 + 9 + (-3 + 10) + 11 // 33 // 14 // 7 / (1.0 \* 2) + 10 // 6 10 > 11 == 4 / 3 > 1not (2 >= 11 or 10 < 67 or 4 / 4 >= 1) (True or not 2 < 3) and 6 == 4 / 3  | Value15.58.4-2241.0FalseFalseFalse |

2. Parameter Mystery

happy and pumpkin were orange

orange and happy were pumpkin

orange and sleepy were y

pumpkin and x were green

green and pumpkin were vampire

3. If/Else Simulation

|  |  |
| --- | --- |
| Function Call | Value Returned |
| mystery(-5)mystery(0)mystery(7)mystery(18)mystery(49) | -156102252 |

4. Programming (four solutions shown)

def month\_apart(m1, d1, m2, d2):

 if (m1 == m2):

 return False

 elif (m1 <= m2 - 2):

 return True

 elif (m1 >= m2 + 2):

 return True

 elif (m1 == m2 - 1):

 if (d1 <= d2):

 return True

 else:

 return False

 elif (m1 == m2 + 1):

 if (d1 >= d2):

 return True

 else:

 return False

 else:

 return False

def month\_apart(m1, d1, m2, d2):

 if (m1 < m2 - 1 or m1 > m2 + 1):

 return True

 elif (m1 == m2 - 1 and d1 <= d2):

 return True

 elif (m1 == m2 + 1 and d1 >= d2):

 return True

 else:

 return False

def month\_apart(m1, d1, m2, d2):

 return (m2 - m1 > 1) or (m1 - m2 > 1) or

 (m2 - m1 == 1 and d1 <= d2) or

 (m1 - m2 == 1 and d1 >= d2)

def month\_apart(m1, d1, m2, d2):

 return abs((m1 \* 31 + d1) - (m2 \* 31 + d2)) >= 31

5. Programming (two solutions shown)

def print\_grid(rows, cols):

 for i in range(1, rows + 1):

 print(i, end=’’)

 for j in range(1, cols):

 print(", " + str(i + rows \* j), end=’’)

 print()

def print\_grid(rows, cols):

 for i in range(1, rows + 1):

 for for j in range(1, cols):

 print(str(i + rows \* j) + ", ", end=’’)

 print(i + rows \* (cols - 1))

6. Programming

def count\_even\_digits(n, length):

 count = 0

 for i in range(0,length):

 digit = n % 10

 n = n // 10

 if (digit % 2 == 0):

 count += 1

 return count

# Midterm 2 Sample 1

**1. List Mystery**
Consider the following function:

def list\_mystery(list):

 int x = 0

 for i in range(0, len(list) - 1):

 if (list[i] > list[i + 1]):

 x += 1

 return x

 In the left-hand column below are specific lists of integers. Indicate in the right-hand column what value would be returned by function list\_mystery if the integer list in the left-hand column is passed as its parameter.

|  |  |
| --- | --- |
| Original Contents of List | Value Returned |
| a1 = [8]result1 = **list\_mystery(a1)**a2 = [14, 7]result2 = **list\_mystery(a2)**a3 = [7, 1, 3, 2, 0, 4]result3 = **list\_mystery(a3)**a4 = [10, 8, 9, 5, 6]result4 = **list\_mystery(a4)**a5 = [8, 10, 8, 6, 4, 2]result5 = **list\_mystery(a5)** | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

**2. Reference Semantics Mystery**
The following program produces 4 lines of output. Write the output below, as it would appear on the console.

 def main():

 y = 1

 x = 3

 a = [0] \* 4

 mystery(a, y, x)

 print(str(x) + " " + str(y) + " " + str(a))

 x = y - 1

 mystery(a, y, x)

 print(str(x) + " " + str(y) + " " + str(a))

 def mystery(a, x, y):

 if (x < y):

 x += 1

 a[x] = 17

 else:

 a[y] = 17

 print(str(x) + " " + str(y) + " " + str(a))

**3. Assertions**
For the following function, identify each of the three assertions in the table below as being either ALWAYS true, NEVER true or SOMETIMES true / sometimes false at each labeled point in the code. You may abbreviate these choices as A/N/S respectively.

def mystery():

 y = 0

 z = 1

 next = input()

 ***# Point A***

 while (next >= 0):

 ***# Point B***

 if (y > z):

 ***# Point C***

 z = y

 y += 1

 next = input()

 ***# Point D***

 ***# Point E***

 return z

|  |  |  |  |
| --- | --- | --- | --- |
|  | next < 0 | y > z | y == 0 |
| Point A |  |  |  |
| Point B |  |  |  |
| Point C |  |  |  |
| Point D |  |  |  |
| Point E |  |  |  |

**4. File Processing**
Write a function named word\_stats that accepts as its parameter the name of a file that contains a sequence of words and that reports the total number of words (as an integer) and the average word length (as an un-rounded real number). For example, suppose file contains the following words:

To be or not to be, that is the question.

 For the purposes of this problem, we will use whitespace to separate words. That means that some words include punctuation, as in "be,". For the input above, your function should produce exactly the following output:

Total words = 10

Average length = 3.2

**6. List Programming**
Write a function named min\_gap that accepts an integer list as a parameter and returns the minimum 'gap' between adjacent values in the list. The gap between two adjacent values in a list is defined as the second value minus the first value. For example, suppose a variable called list is a list of integers that stores the following sequence of values.

list = [1, 3, 6, 7, 12]

 The first gap is 2 (3 - 1), the second gap is 3 (6 - 3), the third gap is 1 (7 - 6) and the fourth gap is 5 (12 - 7). Thus, the call of min\_gap(list) should return 1 because that is the smallest gap in the list. Notice that the minimum gap could be a negative number. For example, if list stores the following sequence of values:

[3, 5, 11, 4, 8]

 The gaps would be computed as 2 (5 - 3), 6 (11 - 5), -7 (4 - 11), and 4 (8 - 4). Of these values, -7 is the smallest, so it would be returned.

 This gap information can be helpful for determining other properties of the list. For example, if the minimum gap is greater than or equal to 0, then you know the array is in sorted (nondecreasing) order. If the gap is greater than 0, then you know the list is both sorted and unique (strictly increasing).

 If you are passed an list with fewer than 2 elements, you should return 0.

**6. Programming**
Write a function named longest\_sorted\_equence that accepts a list of integers as a parameter and that returns the length of the longest sorted (nondecreasing) sequence of integers in the list. For example, if a variable named list stores the following values:

List = [3, 8, 10, 1, 9, 14, -3, 0, 14, 207, 56, 98, 12]

 then the call of longest\_sorted\_sequence(list) should return 4 because the longest sorted sequence in the array has four values in it (the sequence -3, 0, 14, 207). Notice that sorted means nondecreasing, which means that the sequence could contain duplicates. For example, if the list stores the following values:

list2 = [17, 42, 3, 5, 5, 5, 8, 2, 4, 6, 1, 19]

 Then the function would return 5 for the length of the longest sequence (the sequence 3, 5, 5, 5, 8). Your function should return 0 if passed an empty list. Your function should return 1 if passed a list that is entirely in decreasing order or contains only one element.

# Midterm 2 Sample 1 Solutions

1.

|  |  |
| --- | --- |
| Calla1 = [8]result1 = list\_mystery (a1)a2 = [14, 7]result2 = list\_mystery (a2)a3 = [7, 1, 3, 2, 0, 4]result3 = list\_mystery (a3)a4 = [10, 8, 9, 5, 6]result4 = list\_mystery (a4)a5 = [8, 10, 8, 6, 4, 2]result5 = list\_mystery (a5) | Value Returned01324 |

2.

2 3 [0, 0, 17, 0]

3 1 [0, 0, 17, 0]

1 0 [17, 0, 17, 0]

0 1 [17, 0, 17, 0]

3.

|  |  |  |  |
| --- | --- | --- | --- |
|  | next < 0 | y > z | y == 0 |
| Point A | SOMETIMES | NEVER | ALWAYS |
| Point B | NEVER | SOMETIMES | SOMETIMES |
| Point C | NEVER | ALWAYS | NEVER |
| Point D | SOMETIMES | SOMETIMES | NEVER |
| Point E | ALWAYS | SOMETIMES | SOMETIMES |

4.

def word\_stats(file\_name):

 words = open(file\_name).read().split()

 count = 0

 sum\_length = 0

 for word in words:

 count += 1

 sum\_length += len(word)

 average = sum\_length / count

 print("Total words = " + str(count))

 print("Average length = " + str(average))

5.

def min\_gap(list):

 if (len(list) < 2):

 return 0

 else:

 min = list[1] - list[0]

 for i in range(2, len(list)):

 gap = list[i] - list[i - 1]

 if (gap < min):

 min = gap

 return min

6.

def longest\_sorted\_sequence(list):

 if (len(list) == 0):

 return 0

 max = 1

 count = 1

 for i in range(1, len(list)):

 if (list[i] >= list[i - 1]):

 count += 1

 else:

 count = 1

 if (count > max):

 max = count

 return max

### Final Sample 1

**1. While Loop Simulation**
For each call of the function below, write the output that is printed:

def mystery(i, j):

 while (i != 0 and j != 0):

 i = i // j

 j = (j - 1) // 2

 print(str(i) + " " + str(j) + " ", end='')

 print(i)

Function Call Output

mystery(5, 0) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

mystery(3, 2) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

mystery(16, 5) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

mystery(80, 9) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

mystery(1600, 40) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Inheritance Mystery

Assume that the following classes have been defined:

|  |  |
| --- | --- |
| class A(B): def method2(self): print("a 2 ", end='') self.method1()class B(C): def \_\_str\_\_(se;f): return "b"  def method2(self): print("b 2 ", end='') super(B, seld).method2() | class C: def \_\_str\_\_(self): return "c"  def method1(self): print("c 1 ", end='')  def method2(self): print("c 2 ", end='')class D(B): def method1(self): print("d 1 ", end='') self.method2() |

Given the classes above, what output is produced by the following code?

elements = [A(), B(), C(), D()]

for i in range(0, len(elements)):

 print(**elements[i]**)

 elements[i].**method1**()

 print()

 elements[i].**method2**()

 print()

 print()

**3. Collections Mystery**

Consider the following method:

 def mystery(data, pos, n):

 result = set()

 for i in range(0, n):

 for j in range(0, n):

 result.add(data[i + pos][j + pos])

 return result

 Suppose that a variable called grid has been declared as follows:

 grid = [[8, 2, 7, 8, 2, 1], [1, 5, 1, 7, 4, 7],

 [5, 9, 6, 7, 3, 2], [7, 8, 7, 7, 7, 9],

 [4, 2, 6, 9, 2, 3], [2, 2, 8, 1, 1, 3]]

 which means it will store the following 6-by-6 grid of values:

 8 2 7 8 2 1

 1 5 1 7 4 7

 5 9 6 7 3 2

 7 8 7 7 7 9

 4 2 6 9 2 3

 2 2 8 1 1 3

 For each call below, indicate what value is returned. If the function call results in an error, write "error" instead.

 Function Call Contents of Set Returned

 mystery(grid, 2, 2) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 mystery(grid, 0, 2) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 mystery(grid, 3, 3) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. List Programming

Write a function named is\_unique that takes a list of integers as a parameter and that returns a boolean value indicating whether or not the values in the list are unique (True for yes, False for no). The values in the list are considered unique if there is no pair of values that are equal. For example, if a variable called list stores the following values:

list = [3, 8, 12, 2, 9, 17, 43, -8, 46, 203, 14, 97, 10, 4]

Then the call of is\_unique(list) should return True because there are no duplicated values in this list.
If instead the list stored these values:

list = [4, 7, 2, 3, 9, 12, -47, -19, 308, 3, 74]

Then the call should return False because the value 3 appears twice in this list. Notice that given this definition, a list of 0 or 1 elements would be considered unique.

**5. Dictionary/Set Programming**

Write a function called count\_in\_area\_code that accepts two parameters, a dictionary from names (strings) to phone numbers (strings) and an area code (as a string), and returns how many unique phone numbers in the map use that area code. For example, if a map m contains these pairs:

{Marty=206-685-2181, Rick=520-206-6126, Beekto=206-685-2181,

 Jenny=253-867-5309, Stuart=206-685-9138, DirecTV=800-494-4388,

 Bob=206-685-9138, Benson=206-616-1246, Hottline=900-520-2767}

The call of count\_in\_area\_code(m, "206") should return 3, because there are 3 unique phone numbers that use the 206 area code: Marty/Beekto's number of "206-685-2181", Stuart/Bob's number of "206-685-9138", and Benson's number of "206-616-1246".

You may assume that every phone number value string in the dictionary will begin with a 3-digit numeric area code, and that the area code string passed will be a numeric string exactly 3 characters in length. If the dictionary is empty or contains no phone numbers with the given area code, your function should return 0.

You may create one collection (list, dictionary, set) of your choice as auxiliary storage to solve this problem. You can have as many simple variables as you like. You should not modify the contents of the dictionary passed to your function.

**6. Programming**

 Write a function called same\_pattern that returns true or false depending upon whether two strings have the same pattern of characters. More precisely, two strings have the same pattern if they are of the same length and if two characters in the first string are equal if and only if the characters in the corresponding positions in the second string are also equal. Below are some examples of patterns that are the same and patterns that differ (keep in mind that the method should return the same value no matter what order the two strings are passed).

 1st String 2nd String Same Pattern?

 ------------ -------------- -------------

 "" "" True

 "a" "x" True

 "a" "ab" False

 "ab" "ab" True

 "aa" "xy" False

 "aba" "+-+" True

 "---" "aba" False

 "abcabc" "zodzod" True

 "abcabd" "zodzoe" True

 "abcabc" "xxxxxx" False

 "aaassscccn" "aaabbbcccd" True

 "asasasasas" "xyxyxyxyxy" True

 "ascneencsa" "aeiouuoiea" True

 "aaassscccn" "aaabbbcccd" True

 "asasasasas" "xxxxxyyyyy" False

 "ascneencsa" "aeiouaeiou" False

 "aaassscccn" "xxxyyyzzzz" False

 "aaasssiiii" "gggdddfffh" False

Your function should take two parameters: the two strings to compare. You are allowed to create new strings, but otherwise you are not allowed to construct extra data structures to solve this problem (no list, set, dictionary, etc). You are limited to the string functions on the cheat sheet.

7. 2-d Lists

Write a function called find\_max that takes a two dimensional list as a parameter and returns the number of the row that sums to the greatest value. For example if you had the following list of lists:

list = [[1, 2, 3], [2, 3, 3], [1, 3, 3]]

 The first row would be 6, the second 8 and the third 7. The function would therefore return 1.

You can assume the passed in list of lists has at least one row and one column. You cannot assume that it is square.

8. Critters

Write a class Ostrich that extends the Critter class from the Critters assignment, including its get\_move and get\_color methods. An Ostrich object first stays in the same place for 10 moves, then moves 10 steps to either the WEST or the EAST, then repeats. In other words, after sitting still for 10 moves, the ostrich randomly picks to go west or east, then walks 10 steps in that same direction. Then it stops and sits still for 10 moves and repeats. Whenever an Ostrich is moving (that is, whenever its last call to get\_move returned a direction other than DIRECTION\_CENTER), its color should be white ("white"). As soon as it stops moving, and initially when it first appears in the critter world, its color should be cyan ("cyan"). When randomly choosing west vs. east, the two directions should be equally likely.

 You may add anything needed (fields, other methods) to implement the above behavior appropriately. All other critter behavior not discussed here uses the default values.

9. Classes and Objects

|  |  |
| --- | --- |
| Suppose that you are provided with a pre-written class Date as described at right. (The headings are shown, but not the method bodies, to save space.) Assume that the fields, constructor, and methods shown are already implemented. You may refer to them or use them in solving this problem if necessary.Write an instance method named **compare** that will be placed inside the Date class to become a part of each Date object's behavior. The compare method accepts another Date as a parameter and compares the two dates to see which comes first in chronological order. It returns an integer with one of the following values:* a negative integer (such as -1) if the date represented by this Date comes before that of the parameter
* 0 if the two Date objects represent the same month and day
* a positive integer (such as 1) if the date represented by this Date comes after that of the parameter

For example, if these Date objects are declared in client code:sep19 = Date(9, 19)dec15 = Date(12, 15)temp = Date(9, 19)sep11 = Date(9, 11)The following boolean expressions should have True results.sep19.compare(sep11) > 0sep11.compare(sep19) < 0temp.compare(sep19) == 0dec15.compare(sep11) > 0Your method should not modify the state of either Date object (such as by changing their day or month field values). | # Each Date object stores a single# month/day such as September 19.# This class ignores leap years.class Date:  # Constructs a date with # the given month and day. def **\_\_init\_\_**(self, m, d): self.\_\_ **month** = m self.\_\_ **day** = d # Returns the date's day. def **get\_day**(self) # Returns the date's month. def **get\_month**(self) # Returns the number of days # in this date's month. def **days\_in\_month**(self) # Modifies this date's state # so that it has moved forward # in time by 1 day, wrapping # around into the next month # or year if necessary. # example: 9/19 -> 9/20 # example: 9/30 -> 10/1 # example: 12/31 -> 1/1 def **next\_day**()  *# your method would go here* |
|  |  |

### Final Sample 1 Solutions

1. While Loop Simulation

|  |  |
| --- | --- |
| Function Call | Output |
| mystery(5, 0)mystery(3, 2)mystery(16, 5)mystery(80, 9)mystery(1600, 40) | 51 0 13 2 1 0 18 4 2 1 2 0 240 19 2 9 0 4 0 |

**2. Inheritance Mystery**

b

c 1

a 2 c 1

b

c 1

b 2 c 2

c

c 1

c 2

b

d 1 b 2 c 2

b 2 c 2

**3. Collections Mystery**

 Function Call Contents of Set Returned

 -----------------------------------------------

 mystery(grid, 2, 2) [6, 7]

 mystery(grid, 0, 2) [1, 2, 5, 8]

 mystery(grid, 3, 3) [1, 2, 3, 7, 9]

**4. List Programming**

def is\_unique(list):

 for i in range(1, len(list)):

 for j in range(i, len(list)):

 if (list[i - 1] == list[j]):

 return False

 return True

**5. Collections Programming**

def count\_in\_area\_code(numbers, area\_code):

 unique\_numbers = set()

 for name, phone in numbers.items():

 if (phone[0:3] == area\_code):

 unique\_numbers.add(phone)

 return len(unique\_numbers)

**6. Programming**

 def same\_pattern(s1, s2):

 if (len(s1) != len(s2)):

 return False

 for i in range(0, len(s1)):

 for j in range(i + 1, len(s1)):

 if (s1[i] == s1[j] and s2[i] != s2[j]):

 return False

 if (s2[i] == s2[j] and s1[i] != s1[j]):

 return False

 return True

**7. 2d Lists**

def find\_max(lis):

 max\_sum = 0

 max\_row = 0

 for i in range(0, len(lis)):

 cur\_sum = 0

 cur\_row = i

 for j in range(0, len(lis[i])):

 cur\_sum += lis[i][j]

 if cur\_sum > max\_sum:

 max\_sum = cur\_sum

 max\_row = cur\_row

 return max\_row

**8. Critters**

class Ostrich(Critter):

 def \_\_init\_\_(self):

 super(Ostrich, self).\_\_init\_\_()

 self.\_\_hiding = True

 self.\_\_steps = 0

 self.\_\_west = randint(0, 1) == 0

 def get\_color(self):

 if (self.\_\_hiding):

 return "cyan"

 else:

 return "white"

 def get\_move(self):

 if (self.\_\_steps == 10):

 self.\_\_steps = 0 # Pick a new direction and re-set the steps counter

 self.\_\_hiding = not self.\_\_hiding

 self.\_\_west = randint(0, 1) == 0

 self.\_\_steps += 1

 if (self.\_\_hiding):

 return DIRECTION\_CENTER

 elif (self.\_\_west):

 return DIRECTION\_WEST

 else:

 return DIRECTION\_EAST

**9. Classes**

def compare(other):

 if (self.\_\_month < other.\_\_month or (self.\_\_month == other.\_\_month and

 self.\_\_day < other.\_\_day)):

 return -1

 elif (self.\_\_month == other.\_\_month and self.\_\_day == other.\_\_day):

 return 0

 else:

 return 1