**Sequences: Lists and Tuples**

* Introduction

5.1 Q1: Which of the following statements is false?

a. Collections are prepackaged data structures consisting of related data items.

b. Examples of collections include your favorite songs on your smartphone, your contacts list, a library’s books, your cards in a card game, your favorite sports team’s players, the stocks in an investment portfolio, patients in a cancer study and a shopping list.

c. Lists are modifiable and tuples are not. Each can hold items of the same or different types.

d. Tuples can dynamically resize as necessary, growing and shrinking at execution time.

Answer: d. Tuples can dynamically resize as necessary, growing and shrinking at execution time. Actually, lists (not tuples) can dynamically resize as necessary, growing and shrinking at execution time.

* Lists

5.2 Q1: Lists may store \_\_\_\_\_\_\_\_ data, that is, data of many different types.

a. parallel

b. heterogeneous

c. homogeneous

d. None of the above.

Answer: b. heterogeneous

5.2 Q2: Which of the following statements about the list c are false?

c = [-45, 6, 0, 72, 1543]

a. You reference a list element by writing the list’s name followed by the element’s index (that is, its position number) enclosed in square brackets ([], known as the subscription operator).

b. The names c’s elements are c[0], c[1], c[2], c[3] and c[4].

c. The length of c is 5.

d. All of the above statements are true.

Answer: d. All of the above statements are true.

5.2 Q3: Which of the following statements is false?

a. Lists are mutable—their elements can be modified.

b. You can insert and delete list elements, changing the list’s length.

c. Python’s string and tuple sequences are immutable—they cannot be modified.

d. You can get the individual characters in a string, and you can assign a new value to one of the string’s characters.

Answer: d. You can get the individual characters in a string, and you can assign a new value to one of the string’s characters. Actually, you can get the individual characters in a string, but attempting to assign a new value to one of the characters causes a TypeError.

5.2 Q4: Which of the following statements is false?

a. You can concatenate two lists, two tuples or two strings using the + operator. The result is a new sequence of the same type containing the left operand’s elements followed by the right operand’s elements.

b. A TypeError occurs if the + operator’s operands are difference sequence types—for example, concatenating a list and a tuple is an error.

c. List elements can be accessed via their indices and the subscription operator ([]).

d. All of the above statements are true.

Answer: d. All of the above statements are *true*.

5.2 Q5: a. We’ve replaced the results of the four list comparisons below with ???. What are those four values?

In [1]: a = [1, 2, 3]  
  
In [2]: b = [1, 2, 3]  
  
In [3]: c = [1, 2, 3, 4]  
  
In [4]: a == b   
Out[4]: ???  
  
In [5]: a == c   
Out[5]: ???  
  
In [6]: a < c   
Out[6]: ???  
  
In [7]: c >= b   
Out[7]: ???

a. False, True, False, False.

b. True, False, False, True.

c. True, False, True, True.

d. True, True, True, False.

Answer: c. True, False, True, True.

* Tuples

5.3 Q1: Which of the following statements is false?

a. Tuples are immutable.

b. Tuples must store heterogeneous data.

c. A tuple’s length is its number of elements.

d. A tuple’s length cannot change during program execution.

Answer: b. Tuples must store heterogeneous data. Actually, tuples *typically* store heterogeneous data, but the data *can be* homogeneous.

5.3 Q2: Which of the following statements is false?

a. You can pack a tuple by separating its values with commas.

b. When you output a tuple, Python always displays its contents in parentheses.

c. You may surround a tuple’s comma-separated list of values with optional parentheses.

d. The following statement creates a one-element tuple:

a\_singleton\_tuple = ('red')

Answer: d. The following statement creates a one-element tuple:

a\_singleton\_tuple = ('red')

Actually, ('red') simply creates string variable. ('red',) would create a one-element tuple.

5.2 Q3: Which of the following statements is false?

a. Usually, you iterate over a tuple’s elements.

b. Like list indices, tuple indices start at 0.

c. The following code creates time\_tuple representing an hour, minute and second, displays the tuple, then uses its elements to calculate the number of seconds since midnight:

In [1]: time\_tuple = (9, 16, 1)  
  
In [2]: time\_tuple  
Out[2]: (9, 16, 1)  
  
In [3]: time\_tuple[0] \* 3600 + time\_tuple[1] \* 60 + time\_tuple[2]  
Out[3]: 33361

d. Assigning a value to a tuple element causes a TypeError.

Answer: a. Usually, you iterate over a tuple’s elements. Actually, usually you *do not* iterate over them; rather, you access each individually.

5.3 Q4: Which of the following statements is false?

a. The += augmented assignment statement can be used with strings and tuples, even though they’re immutable.

b. In the following code, after the two assignments, tuple1 and tuple2 are two different copies of the same tuple object:

In [1]: tuple1 = (10, 20, 30)  
  
In [2]: tuple2 = tuple1  
  
In [3]: tuple2  
Out[3]: (10, 20, 30)

c. Concatenating the tuple (40, 50) to tuple1 from Part (b) creates a new tuple, then assigns a reference to it to the variable tuple1—tuple2 still refers to the original tuple.

d. For a string or tuple, the item to the right of += must be a string or tuple, respectively—mixing types causes a TypeError.

Answer: b. In the following code, after the two assignments, tuple1 and tuple2 are two different copies of the same tuple object:

In [1]: tuple1 = (10, 20, 30)  
  
In [2]: tuple2 = tuple1  
  
In [3]: tuple2  
Out[3]: (10, 20, 30)

Actually, after the two assignments, tuple1 and tuple2 refer to the *same* tuple object.

5.3 Q5: Which of the following statements is false?

a. The following code creates a student\_tuple with a first name, last name and list of grades:

student\_tuple = ('Amanda', 'Blue', [98, 75, 87])

b. Even though the tuple in Part (a) is immutable, its list element is mutable.

c. In the double-subscripted name student\_tuple[2][1], Python views student\_tuple[2] as the element of the tuple containing the list [98, 75, 87], then uses [1] to access the list element containing 75.

d. All of the above statements are true.

Answer: d. All of the above statements are *true*.

* Unpacking Sequences

5.4 Q1: Which of the following statements is false?

a. You can unpack any sequence’s elements by assigning the sequence to a comma-separated list of variables.

b. A ValueError occurs if the number of variables to the left of the assignment symbol is not identical to the number of elements in the sequence on the right.

c. The following code unpacks a sequence produced by range:

number1, number2, number3 = range(10, 40, 10)

Answer: d) All of the above are true.

5.4 Q2: Which of the following statements is false?

a. The preferred mechanism for accessing an element’s index and value is the built-in function enumerate, which receives an iterable and creates an iterator that, for each element, returns a tuple containing the element’s index and value.

b. The following code uses the built-in function list to create a list containing enumerate’s results:

colors = ['red', 'orange', 'yellow']  
colors\_list = list(enumerate(colors))

c. The following for loop unpacks each tuple returned by enumerate into the variables index and value and displays them:

for index, value in enumerate(colors):  
 print(f'{index}: {value}')

d. Each of the above is true.

Answer: d. Each of the above is true.

* Sequence Slicing

5.5 Q1: Which of the following statements is false?

a. You can slice sequences to create new sequences of the same type containing subsets of the original elements.

b. Slice operations can modify mutable sequences. Slice operations that do not modify a sequence work identically for lists, tuples and strings.

c. The following code creates a slice consisting of the elements at indices 2 through 6 of a numbers:

numbers = [2, 3, 5, 7, 11, 13, 17, 19]  
numbers2 = numbers[2:6]

d. When taking a slice of a list, the original list is not modified.

Answer: c. The following code creates a slice consisting of the elements at indices 2 through 6 of a numbers:

numbers = [2, 3, 5, 7, 11, 13, 17, 19]  
numbers2 = numbers[2:6]

Actually, the slice consists of the elements at indices 2 through 5 of the list.

5.5 Q2: Which of the following statements is false?

a. When you specify a slice and omit the starting index, 0 is assumed. So, the slice numbers[:6] is equivalent to the slice numbers[0:6]. If you omit the ending index, Python assumes the sequence’s length.

b. Omitting both the start and end indices on a slice copies the entire sequence.

c. Slices make deep copies of the sequence’s elements.

d. With slices, the new sequence’s elements refer to the same objects as the original sequence’s elements, rather than to separate copies.

Answer: c. Slices make deep copies of the sequence’s elements. Actually, though slices create new objects, slices make *shallow* copies of the original sequence’s elements—that is, they copy the elements’ references but not the objects they point to.

5.5 Q3 Which of the following statements is false?

a. You can use a negative step to select slices in reverse order.

b. The following code concisely creates a new list in reverse order:

numbers[::-1]

c. You can modify a list by assigning to a slice of it—the rest of the list is unchanged.

d. All of the above statements are true.

Answer: d. All of the above statements are *true*.

5.5 Q4: Which of the following statements is false?

a. The following code deletes only the first three elements of numbers by assigning an empty list to the three-element slice:

numbers[0:3] = []

b. The following code assigns a list’s elements to a slice of every other element of numbers:

numbers = [2, 3, 5, 7, 11, 13, 17, 19]  
numbers[2:2:2] = [100, 100, 100, 100]

c. The following code deletes all the elements in numbers, leaving the existing list empty:

numbers[:] = []

d. When you assign a new object to a variable the original object will be garbage collected if no other variables refer to it.

Answer: b. The following code assigns a list’s elements to a slice of every other element of numbers:

numbers = [2, 3, 5, 7, 11, 13, 17, 19]  
numbers[2:2:2] = [100, 100, 100, 100]

Actually, numbers[2:2:2] s. b. numbers[::2].

* **del** Statement

5.6 Q1: Which of the following statements is false?

a. You can use del to remove the element at any valid index or the element(s) from any valid slice.

b. The following code creates a list, then uses del to remove its last element:

numbers = list(range(0, 10))  
del numbers[1]

c. The following deletes the list’s first two elements:

del numbers[0:2]

d. The following uses a step in the slice to delete every other element from the entire list:

del numbers[::2]

Answer: b. The following code creates a list, then uses del to remove its last element:

numbers = list(range(0, 10))  
del numbers[1]

Actually, snippet [3] should be del numbers[-1].

5.6 Q2: Which of the following statements is false?

a. The following code deletes all of the list’s elements:

del numbers[:]

b. The del statement can delete any variable.

c. The following code deletes the variable numbers from the interactive session, then attempt to display the variable’s value, causing a NameError:

del numbers

d. After deleting numbers from the interactive session, attempting to display it displays an empty list.

Answer: d. After deleting numbers from the interactive session, attempting to display it displays an empty list. Actually, attempting to display numbers after it has been deleted from the session causes a NameError.

* Passing Lists to Functions

5.6 Q1: Which of the following statements is false?

a. The function modify\_elements multiplies each element of its list argument by 2:

def modify\_elements(items):  
 """"Multiplies all element values in items by 2."""  
 for i in range(len(items)):  
 items[i] \*= 2

b. Part (a)’s function modify\_elements’ items parameter receives a reference to the original list, so the statement in the loop’s suite modifies each element in the original list object.

c. When you pass a tuple to a function, attempting to modify the tuple’s immutable elements results in a TypeError.

d. Tuples may contain mutable objects, such as lists, but those objects cannot be modified when a tuple is passed to a function.

Answer: d. Tuples may contain mutable objects, such as lists, but those objects cannot be modified when a tuple is passed to a function. Actually, those objects still can be modified when a tuple is passed to a function.

* Sorting Lists

5.8 Q1: Which of the following statements is false?

a. List method sort modifies a list to arrange its elements in ascending order:

numbers = [10, 3, 7, 1, 9, 4, 2, 8, 5, 6]  
numbers.sort()

b. To sort a list in descending order, call list method sort with the optional keyword argument reverse-=False.

c. Built-in function sorted returns a new list containing the sorted elements of its argument sequence—the original sequence is unmodified.

d. All of the above statements are true.

Answer: b. To sort a list in descending order, call list method sort with the optional keyword argument reverse-=False. Actually, to sort a list in descending order, call list method sort with the optional keyword argument reverse-=True—with reverse-=False, sort actually sorts in ascending order.

* Searching Sequences

5.9 Q1: Which of the following statements is false?

a. Often, you’ll want to determine whether a sequence (such as a list, tuple or string) contains a value that matches a particular key value.

b. Searching is the process of locating a key.

c. List method index takes as an argument a search key—the value to locate in the list—then searches through the list from index 1 and returns the index of the first element that matches the search key.

d. List method index raises a ValueError if the value you’re searching for is not in the list.

Answer: List method index takes as an argument a search key—the value to locate in the list—then searches through the list from index 1 and returns the index of the *first* element that matches the search key. Actually, list method index searches from index 0.

5.9 Q2: Which of the following statements is false?

a. You can use \*= to multiply a sequence—that is, append a sequence to itself multiple times.

b. After the following snippet, numbers contains two copies of the original list’s contents:

numbers \*= 2

c. The following code searches the updated list for the value 5 starting from index 7 and continuing through the end of the list:

numbers.index(5, 7)

d. All of the above statements are *true*.

Answer: d. All of the above statements are *true*.

5.9 Q3: Which of the following statements is false?

a. Specifying the starting and ending indices causes index to search from the starting index up to but not including the ending index location.

b. The call:

numbers.index(5, 7)

is equivalent to:

numbers.index(5, 7, len(numbers))

c. The following looks for the value 7 in the range of elements with indices 0 through 4:

numbers.index(7, 0, 4)

d. All of the above statements are true.

Answer: c. The following looks for the value 7 in the range of elements with indices 0 through 4:

numbers.index(7, 0, 4)

Actually, the code looks for 7 in the range of elements with indices 0 through 3.

5.9 Q4: Which of the following statements is false?

a. The built-in function any returns True if any item in its iterable argument is True.

b. The built-in function all returns True if all items in its iterable argument are True.

c. Non-empty iterable objects also evaluate to True, whereas any empty iterable evaluates to False.

d. Functions any and all are examples of external iteration in functional-style programming.

Answer: d. Functions any and all are examples of external iteration in functional-style programming. Actually, functions any and all are additional examples of *internal* iteration in functional-style programming.

* Other List Methods

5.10 Q1: Which of the following statements is false? Consider the list color\_names:

color\_names = ['orange', 'yellow', 'green']

a. Lists also have methods that add and remove elements.

b. Method insert adds a new item at a specified index. The following inserts 'red' at index 0:

color\_names.insert(0, 'red')

c. You can add a new item to the end of a list with methodappend.

d. All of the above statements are true.

Answer: d. All of the above statements are *true*.

5.10 Q2: Which of the following statements is false?

a. Use list method extend to add all the elements of another sequence to the end of a list (this is the equivalent of using +=).

b. The following code adds all the characters of a string then all the elements of a tuple to a list:

sample\_list = []  
s = 'abc'  
sample\_list.extend(s)  
t = (1, 2, 3)  
sample\_list.extend(t)

c. Rather than creating a temporary variable, like t, to store a tuple in Part (b) before appending it to a list, you might want to pass a tuple directly to extend. In this case, the tuple’s parentheses are optional, because extend expects one iterable argument.

d. All of the above statements are true.

Answer: c. Rather than creating a temporary variable, like t, to store a tuple in Part (b) before appending it to a list, you might want to pass a tuple directly to extend. In this case, the tuple’s parentheses are optional, because extend expects one iterable argument. Actually, the tuple’s parentheses are *required*.

5.10 Q3: Which of the following statements is false?

a. List method remove deletes the first element with a specified value—a NameError occurs if remove’s argument is not in the list. To delete all the elements in a list, call method clear.

b. List method count searches for its argument in a list and returns the number of times it is found.

c. List method reverse reverses the contents of a list in place, rather than creating a reversed copy, as we did with a slice previously.

d. List method copy returns a new list containing a shallow copy of the original list.

Answer: a. Method remove deletes the first element with a specified value—a NameError occurs if remove’s argument is not in the list. To delete all the elements in a list, call method clear. Actually, a ValueError occurs if remove’s argument is not in the list.

* Simulating Stacks with Lists

5.11 Q1: Which of the following statements is false?

a. Python does not have a built-in stack type, but you can think of a stack as a constrained list.

b. You push using list method append, which adds a new element to the end of the list.

c. You pop using list method pop with no arguments, which removes and returns the item at the front of the list.

d. You can run out of memory if you keep pushing items faster than you pop them.

Answer: c. You pop using list method pop with no arguments, which removes and returns the item at the front of the list. Actually, you pop using list method pop with no arguments, which removes and returns the item at the *end* of the list.

* List Comprehensions

5.12 Q1: Which of the following statements is false?

a. List comprehensions provide a concise and convenient notation for creating new lists.

b. List comprehensions can replace many for statements that iterate over existing sequences and create new lists, such as:

list1 = []

for item in range(1, 6):

We can accomplish the same task in a single line of code with a list comprehension:

list2 = [item for item in range(1, 6)]

c. The preceding list comprehension’s for clause iterates over the sequence produced by range(1, 6). For each item, the list comprehension evaluates the expression to the left of the for clause and places the expression’s value (in this case, the item itself) in the new list.

d. All of the above statements are true.

Answer: d. All of the above statements are *true*.

5.12 Q2: Which of the following statements is false?

a. A list comprehension’s expression can perform tasks, such as calculations, that map elements to new values (possibly of different types).

b. Mapping is a common functional-style programming operation that produces a result with more elements than the original data being mapped.

c. The following comprehension maps each value to its cube with the expression item \*\* 3:

list3 = [item \*\* 3 for item in range(1, 6)]

d. All of the above statements are true.

Answer: b. Mapping is a common functional-style programming operation that produces a result with more elements than the original data being mapped. Actually, mapping is a common functional-style programming operation that produces a result with the *same* number of elements as the original data being mapped.

5.12 Q3: Which of the following statements is false?.

a. Another common functional-style programming operation is filtering elements to select only those that match a condition.

b. Filtering typically produces a list with more elements than the data being filtered.

c. To do filtering in a list comprehension, use the if clause.

d The following includes in list4 only the even values produced by the for clause:

list4 = [item for item in range(1, 11) if item % 2 == 0]

Answer: b. Filtering typically produces a list with more elements than the data being filtered. Actually, filtering typically produces a list with *fewer* elements than the data being filtered.

* **Generator Expressions**

5.13 Q1: Which of the following statements is false?

a. A generator expression is similar to a list comprehension, but creates an iterable generator object that produces values on demand—this is an example of lazy evaluation.

b. The generator expression in the following for statement squares and returns only the odd values in numbers:

numbers = [10, 3, 7, 1, 9, 4, 2, 8, 5, 6]  
  
for value in (x \*\* 2 for x in numbers if x % 2 != 0):  
 print(value, end=' ')

c. A generator expression does not create a list.

d. All of the above are true.

Answer: d. All of the above are true.

* Filter, Map and Reduce

5.14 Q1: Which of the following statements is false?

a. The following session uses built-in function filter to obtain the odd values in numbers:

In [1]: numbers = [10, 3, 7, 1, 9, 4, 2, 8, 5, 6]

In [2]: def is\_odd(x):  
 ...: """Returns True only if x is odd."""  
 ...: return x % 2 != 0  
 ...:

b. Python functions are objects that you can assign to variables, pass to other functions and return from functions. Functions that receive other functions as arguments are a functional-style capability called higher-order functions. Higher-order functions may also return a function as a result.

c. Function filter’s first argument must be a function that receives one argument and returns True if the value should be included in the result.

d. Function filter returns an iterator, so filter’s results are produced immediately.

Answer: d. Function filter returns an iterator, so filter’s results are produced immediately. Actually, function filter returns an iterator, so filter’s results are not produced until you iterate through them—this is an example of lazy evaluation.

5.14 Q2: Which of the following statements is false?

a. For simple functions that return only a single expression’s value, you can use a lambda expression (or simply a lambda) to define the function inline where it’s needed—typically as it’s passed to another function.

b. A lambda expression is an anonymous function—that is, a function without a name.

c. In the following filter call the first argument is the lambda:

filter(lambda x: x % 2 != 0, numbers)

d. A lambda explicitly returns its expression’s value.

Answer: d. A lambda explicitly returns its expression’s value. Actually, a lambda *implicitly* returns its expression’s value.

5.14 Q3: Which of the following statements is false?

a. The following code uses built-in function map with a lambda to square each value in the list numbers:

list(map(lambda x: x \*\* 2, numbers))

b. Function map’s first argument is a function that receives one value and returns a new value—in Part (a), a lambda that squares its argument. The second argument is an iterable of values to map.

c. Function map uses eager evaluation.

d. The equivalent list comprehension to Part (a) is:

[item \*\* 2 for item in numbers]

Answer: c. Function map uses eager evaluation. Actually, function map uses lazy evaluation.

5.14 Q4: Which of the following statements is false?

a. Reductions process a sequence’s elements into a small number of values.

b. The built-in functions len, sum, min and max perform reductions.

c. You also can create custom reductions using the functools module’s reduce function.

d. In the worlds of big data and Hadoop, MapReduce programming is based on the filter, map and reduce operations in functional-style programming.

Answer: a. Reductions process a sequence’s elements into a small number of values. Actually, reductions process a sequence’s elements into a *single* value.

* Other Sequence Processing Functions

5.15 Q1: Which of the following statements is false?

a. Sometimes you’ll need to find the minimum and maximum of more complex objects, such as strings.

b. Consider the following comparison:

'Red' < 'orange'

The letter 'R' “comes after” 'o' in the alphabet, so 'Red' is greater than 'orange' and the condition above is False.

c. Built-in function ord returns the numerical value of a character.

d. All of the above statements are true.

Answer: b. Consider the following comparison:

'Red' < 'orange'

The letter 'R' “comes after” 'o' in the alphabet, so 'Red' is greater than 'orange' and the condition above is False. Actually, strings are compared by their characters’ underlying *numerical values*, and lowercase letters have *higher* numerical values than uppercase letters so the condition is true.

5.15 Q2: Which of the following statements is false?

a. Built-in function reversed returns an iterator that enables you to iterate over a sequence’s values backward.

b. The following list comprehension creates a new list containing the squares of numbers’ values in the same oder as the list:

numbers = [10, 3, 7, 1, 9, 4, 2, 8, 5, 6]  
reversed\_numbers = [item for item in reversed(numbers)]

c. Built-in function zip enables you to iterate over multiple iterables of data at the same time. The function zip receives as arguments any number of iterables and returns an iterator that produces tuples containing the elements at the same index in each.

d. The following call to zip below produces the tuples ('Bob', 3.5), ('Sue', 4.0) and ('Amanda', 3.75) consisting of the elements at index 0, 1 and 2 of each list, respectively:

names = ['Bob', 'Sue', 'Amanda']  
grade\_point\_averages = [3.5, 4.0, 3.75]

zip(names, grade\_point\_averages)

Answer: b. The following list comprehension creates a new list containing the squares of numbers’ values in the same order as the list:

numbers = [10, 3, 7, 1, 9, 4, 2, 8, 5, 6]  
reversed\_numbers = [item for item in reversed(numbers)]

Actually, b. The the list comprehension creates a new list containing the squares of numbers’ values in reverse order.

* Two-Dimensional Lists

5.16 Q1: Which of the following statements a), b) or c) is false?

a. Lists can contain other lists as elements.

b. To identify a particular table element, we specify two indices—by convention, the first identifies the element’s column, the second the element’s row.

c. Multidimensional lists can have more than two indices.

d. Each of the above statements is true.

Answer: b. To identify a particular table element, we specify *two* indices—by convention, the first identifies the element’s column, the second the element’s row. Actually, the first identifies the element’s row, the second the element’s column.

5.16 Q2: Consider a two-dimensional list with three rows and four columns (i.e., a 3-by-4 list) that might represent the grades of three students who each took four exams in a course:

a = [[77, 68, 86, 73], [96, 87, 89, 81], [70, 90, 86, 81]]

Which of the following statements is false?

a. Writing the list as follows makes its row and column tabular structure clearer:

a = [77, 68, 86, 73], # first student's grades  
 [96, 87, 89, 81], # second student's grades   
 [70, 90, 86, 81] # third student's grades

b. The element names in the last column all have 3 as the second index.

c. The following nested for statement outputs the rows of the two-dimensional list a one row at a time:

for row in a:  
 for item in row:  
 print(item, end=' ')  
 print()

d. All of the above statements are true.

Answer: a. Writing the list as follows makes its row and column tabular structure clearer:

a = [77, 68, 86, 73], # first student's grades  
 [96, 87, 89, 81], # second student's grades   
 [70, 90, 86, 81] # third student's grades

Actually, writing the list as follows makes its row and column tabular structure clearer:

a = [[77, 68, 86, 73], # first student's grades  
 [96, 87, 89, 81], # second student's grades   
 [70, 90, 86, 81]] # third student's grades

* Intro to Data Science: Simulation and Static Visualizations

5.17 Q1: Which of the following statements is false?

a. Visualizations help you “get to know” your data.

b. Visualizations give you a powerful way to understand data that goes beyond simply looking at raw data.

c. The Matplotlib visualization library is built over the Seaborn visualization library and simplifies many Seaborn operations.

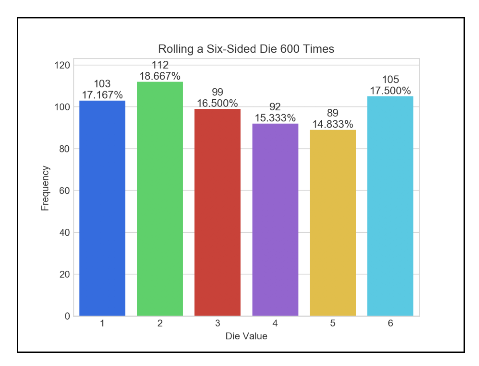
d. All of the above statements are true.

Answer: c. The Matplotlib visualization library is built over the Seaborn visualization library and simplifies many Seaborn operations. Actually, the Seaborn visualization library is built over the Matplotlib visualization library and simplifies many Matplotlib operations.

* Sample Graphs for 600, 60,000 and 6,000,000 Die Rolls

5.17 Q2: Which of the following statements is false?

a. Seaborn refers to the following type of graph as a bar plot:



b. For 600 rolls, we expect about 100 occurrences of each die face. As we run the simulation for 60,000 die rolls, the bars will become much closer in size. At 6,000,000 die rolls, they’ll appear to be exactly the same size. This is the “principal of least privilege” at work.

c. The second screen capture below shows the results for 6,000,000 rolls—surely something you’d never do by hand! In this case, we expect about 1,000,000 of each face, and the frequency bars appear to be identical in length (they’re close but not exactly the same length).

d. With large numbers of die rolls, the frequency percentages are much closer to the expected 16.667%.

Answer: b. For 600 rolls, we expect about 100 occurrences of each die face. As we run the simulation for 60,000 die rolls, the bars will become much closer in size. At 6,000,000 die rolls, they’ll appear to be exactly the same size. This is the “principal of least privilege” at work. Actually, b. Here we expect about 100 occurrences of each die face. As we run the simulation for 60,000 die rolls, the bars will become much closer in size. At 6,000,000 die rolls, they’ll appear to be exactly the same size. This is the *law of large numbers* at work. Self Check

* Visualizing Die-Roll Frequencies and Percentages

5.17 Q3: Which of the following statements is false?

a. The following code uses a list comprehension to create a list of 600 random die values, then uses NumPy’s unique function to determine the unique roll values (guaranteed to include all six possible face values) and their frequencies:

rolls = [random.randrange(1, 7) for i in range(600)]  
values, frequencies = np.unique(rolls, return\_counts=True)

b. The NumPy library provides the high-performance ndarray collection, which is typically much faster than lists.

c. If you pass a list (like rolls) into np.unique, NumPy converts it to an ndarray for better performance.

d. Specifying the keyword argument return\_counts=True tells unique to count each unique value’s number of occurrences.

Answer: a. The following code uses a list comprehension to create a list of 600 random die values, then uses NumPy’s unique function to determine the unique roll values (guaranteed to include all six possible face values) and their frequencies:

rolls = [random.randrange(1, 7) for i in range(600)]  
values, frequencies = np.unique(rolls, return\_counts=True)

Actually, this code uses a list comprehension to create a list of 600 random die values, then uses NumPy’s unique function to determine the unique roll values (*most likely* all six possible face values) and their frequencies.