Functions

* Introduction

4.1 Q1: Which of the following statements is false?

a. Experience has shown that the best way to develop and maintain a large program is to construct it from a small number of large, proven pieces—this technique is called divide and conquer.

b. Using existing functions as building blocks for creating new programs is a key aspect of software reusability—it’s also a major benefit of object-oriented programming.

c. Packaging code as a function allows you to execute it from various locations in your program just by calling the function, rather than duplicating the possibly lengthy code.

d. When you change a function’s code, all calls to the function execute the updated version.

Answer: a. Experience has shown that the best way to develop and maintain a large program is to construct it from a small number of large, proven pieces—this technique is called divide and conquer. Actually, experience has shown that the best way to develop and maintain a large program is to construct it from smaller, more manageable pieces—this technique is called divide and conquer.

* Defining Functions

4.2 Q1: Which of the following statements is false?

a. Each function should perform a single, well-defined task.

b. The following code calls function square twice. We’ve replaced each of the output values with ???. The first call produces the int value 49 and the second call produces the int value 6:

In [1]: def square(number):  
 ...: """Calculate the square of number."""  
 ...: return number \*\* 2  
 ...:   
  
In [2]: square(7)  
Out[2]: ???  
  
In [3]: square(2.5)  
Out[3]: ???

c. The statements defining a function are written only once, but may be called “to do their job” from many points in a program and as often as you like.

d. Calling square with a non-numeric argument like 'hello' causes a TypeError because the exponentiation operator (\*\*) works only with numeric values.

Answer: b). The following code calls function square twice. We’ve replaced each of the output values with ???. The first call produces the int value 49 and the second call produces the int value 6:

In [1]: def square(number):  
 ...: """Calculate the square of number."""  
 ...: return number \*\* 2  
 ...:   
  
In [2]: square(7)  
Out[2]: ???  
  
In [3]: square(2.5)  
Out[3]: ???

Actually, the second call produces the float value 6.25.

4.2 Q2: Which of the following statements is false?

a. A function definition begins with the def keyword, followed by the function name, a set of parentheses and a colon (:).

b. Like variable identifiers, by convention function names should begin with a lowercase letter and in multiword names underscores should separate each word.

c. The required parentheses contain the function’s parameter list—a comma-separated list of parameters representing the data that the function needs to perform its task.

d. The indented lines after the colon (:) are the function’s suite, which consists of an optional docstring followed by the statements that perform the function’s task.

Answer: d. The indented lines after the colon (:) are the function’s suite, which consists of an optional docstring followed by the statements that perform the function’s task. Actually, The indented lines after the colon (:) are the function’s *block*, which consists of an optional docstring followed by the statements that perform the function’s task.

4.2 Q3: Which of the following statements is false?

a. The following return statement first terminates the function, then squares number and gives the result back to the caller:

return number \*\* 2

b. Function calls can be embedded in expressions.

c. The following code calls square first, then print displays the result:

print('The square of 7 is', square(7))

d. Executing a return statement without an expression terminates the function and implicitly returns the value None to the caller. When there’s no return statement in a function, it implicitly returns the value None after executing the last statement in the function’s block.

Answer: a. The following return statement first terminates the function, then squares number and gives the result back to the caller:

return number \*\* 2

Actually, the return statement first squares number, *then terminates the function* and gives the result back to the caller.

4.2 Q4: Which of the following statements is false?

a. A function’s parameter is created on each call to the function to receive the argument value, and it’s destroyed when the function returns its result to the caller.

b. Variables can be defined in a function’s block.

c. A function’s parameters and variables are all local variables—they can be used anywhere in the program that encloses the function and exist only while the function is executing.

d. Trying to access a local variable outside its function’s block causes a NameError, indicating that the variable is not defined.

Answer: c. A function’s parameters and variables are all local variables—they can be used anywhere in the program that encloses the function and exist only while the function is executing. Actually, a function’s parameters and variables are all local variables—they can be used only inside the function and exist only while the function is executing.

* Functions with Multiple Parameters

4.3 Q1: Which of the following statements is false?

a. The following session defines a maximum function that determines and returns the largest of three values—then calls the function three times with integers, floating-point numbers and strings, respectively.

In [1]: def maximum(value1, value2, value3):  
 ...: """Return the maximum of three values."""  
 ...: max\_value = value1  
 ...: if value2 > max\_value:  
 ...: max\_value = value2  
 ...: if value3 > max\_value:  
 ...: max\_value = value3  
 ...: return max\_value  
 ...:   
  
In [2]: maximum(12, 27, 36)  
Out[2]: 36  
  
In [3]: maximum(12.3, 45.6, 9.7)  
Out[3]: 45.6  
  
In [4]: maximum('yellow', 'red', 'orange')  
Out[4]: 'yellow'

b. You also may call maximum with mixed types, such as ints and floats:

In [5]: maximum(13.5, -3, 7)  
Out[5]: 13.5

c. The call maximum(13.5, 'hello', 7) results in TypeError because strings and numbers cannot be compared to one another with the greater-than (>) operator.

d. All of the above statements are true.

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

4.3 Q2: Which of the following statements is false?

a. The built-in max and min functions know how to determine the largest and smallest of their two or more arguments, respectively.

b. The built-in max and min functions also can receive an iterable, such as a list but not a string.

c. Using built-in functions or functions from the Python Standard Library’s modules rather than writing your own can reduce development time and increase program reliability, portability and performance.

d. All of the above statements are true.

Answer: b. The built-in max and min functions also can receive an iterable, such as a list but not a string. Actually, each of these functions also can receive an iterable, such as a list **or a string**.

* Random-Number Generation

4.4 Q1: Which of the following statements is false?

a. You can introduce the element of chance via the Python Standard Library’s random module.

b. The following code produces 10 random integers in the range 1–6 to simulate rolling a six-sided die:

import random  
  
for roll in range(10):  
 print(random.randrange(1, 7), end=' ')

c. Different values are likely to be displayed each time you run the code in Part (b).

d. Sometimes, you may want to guarantee reproducibility of a random sequence—for debugging, for example. You can do this with the random module’s repeat function.

Answer: d. Sometimes, you may want to guarantee reproducibility of a random sequence—for debugging, for example. You can do this with the random module’s repeat function. Actually, you can guarantee reproducibility of a random sequence with the random module’s seed function.

4.4 Q2: Which of the following statements is false?

a. If randrange truly produces integers at random, every number in its range has an equal probability (or chance or likelihood) of being returned each time we call it.

b. We can use Python’s underscore (\_) digit separator to make a value like 6000000 more readable as 6\_000\_000.

c. The expression range(6,000,000) would be incorrect. Commas separate arguments in function calls, so Python would treat range(6,000,000) as a call to range with the three arguments 6, 0 and 0.

d. All of the above statements are true.

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

4.4 Q3: Which of the following statements is false?

a. Function randrange actually generates pseudorandom numbers, based on an internal calculation that begins with a numeric value known as a seed.

b. When you’re debugging logic errors in programs that use randomly generated data, it can be helpful to use the same sequence of random numbers until you’ve eliminated the logic errors, before testing the program with other values.

c. You can use the random module’s seed function to seed the random-number generator yourself—this forces randrange to begin calculating its pseudorandom number sequence from the seed you specify. Choosing the same seed will cause the random number generator to generate the same sequence of random numbers.

d. In the following session, snippets [2] and [5] produce the same results purely by coincidence:

In [1]: random.seed(32)  
  
In [2]: for roll in range(10):  
 ...: print(random.randrange(1, 7), end=' ')  
 ...:   
1 2 2 3 6 2 4 1 6 1   
In [3]: for roll in range(10):  
 ...: print(random.randrange(1, 7), end=' ')  
 ...:   
1 3 5 3 1 5 6 4 3 5   
In [4]: random.seed(32)  
  
In [5]: for roll in range(10):  
 ...: print(random.randrange(1, 7), end=' ')  
 ...:   
1 2 2 3 6 2 4 1 6 1

Answer: d. In the following session, snippets [2] and [5] produce the same results purely by coincidence:

In [1]: random.seed(32)  
  
In [2]: for roll in range(10):  
 ...: print(random.randrange(1, 7), end=' ')  
 ...:   
1 2 2 3 6 2 4 1 6 1   
In [3]: for roll in range(10):  
 ...: print(random.randrange(1, 7), end=' ')  
 ...:   
1 3 5 3 1 5 6 4 3 5   
In [4]: random.seed(32)  
  
In [5]: for roll in range(10):  
 ...: print(random.randrange(1, 7), end=' ')  
 ...:   
1 2 2 3 6 2 4 1 6 1

Actually, in the code session, snippets [2] and [5] produce the same results, because snippets [1] and [4] use the same seed (32).

* Case Study: A Game of Chance

4.5 Q1: Which of the following statements is false?

a. To use a tuple’s values, you can assign them to a comma-separated list of variables, which unpacks the tuple.

b. Like a list, a tuple is a sequence.

c. To avoid name collisions, variables in different functions’ blocks must have different names.

d. Each local variable is accessible only in the block that defined it.

Answer: c. To avoid name collisions, variables in different functions’ blocks must have different names. Actually, variables that belong to different functions’ blocks can have the same name without colliding.

4.5 Q2: Which of the following statements is false?

a. The in operator in the following expression tests whether the tuple (7, 11) contains sum\_of\_dice’s value. The operator’s right operand can be any iterable:

sum\_of\_dice in (7, 11)

b. There’s also a not in operator to determine whether a value is not in an iterable.

c. The concise condition in Part (a) is equivalent to

(sum\_of\_dice = 7) or (sum\_of\_dice = 11)

d. The following expression tests whether the tuple (2, 3, 12) contains sum\_of\_dice’s value.

sum\_of\_dice in (2, 3, 12)

Answer: c. The concise condition in Part (a) is equivalent to

(sum\_of\_dice = 7) or (sum\_of\_dice = 11)

Actually, the condition in part a) is equivalent to

(sum\_of\_dice == 7) or (sum\_of\_dice == 11)

* Python Standard Library

4.6 Q1: Which of the following statements is false?

a. A key programming goal is to avoid “reinventing the wheel.”

b. A module is a file that groups related functions, data and classes.

c. A package groups related modules. Every Python source-code (.py) file you create is a module. They’re typically used to organize a large library’s functionality into smaller subsets that are easier to maintain and can be imported separately for convenience.

d. The Python Standard Library module money provides arithmetic capabilities for performing monetary calculations.

Answer: d. The Python Standard Library module money provides arithmetic capabilities for performing monetary calculations. Actually, the Python Standard Library module decimal provides arithmetic capabilities for performing monetary calculations.

* **math** Module Functions

4.7 Q1: Which of the following statements is false?

a. An import statement of the following form enables you to use a module’s definitions via the module’s name and a dot (.):

import math

b. The following snippet calculates the square root of 900 by calling the math module’s sqrt function, which returns its result as a float value:

math.sqrt(900)

c. The following snippet calculates the absolute value of -10 by calling the math module’s fabs function, which returns its result as a float value:

math.fabs(-10)

d. The value of the expression floor(–3.14159) is –3.0.

Answer: d. The value of the expression floor(-3.14159) is –3.0. Actually, the value of the expression floor (-3.14159) is –4.0.

* Using IPython Tab Completion for Discovery

4.8 Q1: Which of the following statements is false?

a. You can view a module’s documentation in IPython interactive mode via tab completion—a discovery feature that speeds your coding and learning processes.

b. After you type a portion of an identifier and press Tab, IPython completes the identifier for you or provides a list of identifiers that begin with what you’ve typed so far.

c. IPython tab completion results may vary based on your operating system platform and what you have imported into your IPython session:

d. All of the above statements are true.

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

4.8 Q2: Which of the following statements is false?

a. To view a list of identifiers defined in a module, type the module’s name and a dot (.), then press Tab.

b. In the math module, pi and e represent the mathematical constants  and e, respectively.

c. Python does not have constants, so even though pi and e are real-world constants, you must not assign new values to them, because that would change their values.

d. To help distinguish “constants” from other variables, the Python style guide recommends naming your custom constants with a leading underscore (\_) and a trailing underscore.

Answer: d. To help distinguish "constants" from other variables, the Pythonstyle guide recommends naming your custom constants with a leading underscore (\_) and a trailing underscore. Actually, the Pythonstyle guide recommends naming your custom constants with all capital letters.

* Default Parameter Values

4.9 Q1: Which of the following statements is false?

a. When defining a function, you can specify that a parameter has a default parameter value.

b. When calling the function, if you omit the argument for a parameter with a default parameter value, the default value for that parameter is automatically passed.

c. The following defines a function rectangle\_area with default parameter values:

def rectangle\_area(length=2, width=3):  
 """Return a rectangle's area."""  
 return length \* width

d. The call rectangle\_area() to the function in Part (c) returns the value 0 (zero).

Answer: The call rectangle\_area() to the function in Part (c) returns the value 0 (zero). Actually the call rectangle\_area() causes Python to use the default value 2 for the length and the default value 3 for the width and returns the result 6.

4.9 Q2: Assuming the following function definition, which of the following statements is false?

def rectangle\_area(length=2, width=3):  
 """Return a rectangle's area."""  
 return length \* width

a. You specify a default parameter value by following a parameter’s name with an = and a value.

b. Any parameters with default parameter values must appear in the parameter list to the right of parameters that do not have defaults.

c. For the following call, the interpreter passes the default parameter value 3 for the width as if you had called rectangle\_area(3, 10):

rectangle\_area(10)

d. The following call to rectangle\_area has arguments for both length and width, so IPython- ignores the default parameter values:

rectangle\_area(10, 5)

Answer: c. For the following call, the interpreter passes the default parameter value 3 for the width as if you had called rectangle\_area(3, 10):

rectangle\_area(10)

Actually, interpreter passes the default parameter value 3 for the width as if you had called rectangle\_area(10, 3).

* Keyword Arguments

4.10 Q1: Based on the following function definition:

def rectangle\_area(length, width):  
 """Return a rectangle's area."""  
 return length \* width

Which of the following statements based on this function definition is false?

a. Each keyword argument in a call has the form parametername=value.

b. The following call shows that the order of keyword arguments matters—they need to match the corresponding parameters’ positions in the function definition:

rectangle\_area(width=5, length=10)

c. In each function call, you must place keyword arguments after a function’s positional arguments—that is, any arguments for which you do not specify the parameter name. Such arguments are assigned to the function’s parameters left-to-right, based on the argument’s positions in the argument list.

d. Keyword arguments can improve the readability of function calls, especially for functions with many arguments.

Answer: b. The following call shows that the order of keyword arguments matters—they need to match the corresponding parameters’ positions in the function definition:

rectangle\_area(width=5, length=10)

Actually, the call shows that the order of keyword arguments *does not* matter—they do not need to match the corresponding parameters’ positions in the function definition.

* Arbitrary Argument Lists

4.11 Q1: Which of the following statements is false?

a. The min function’s documentation states that min has two required parameters (named arg1 and arg2) and an optional third parameter of the form \*args, indicating that the function can receive any number of additional arguments.

b. The following is a valid call to function min

min(88)

c. The \* before the parameter name \*args in Part (a) tells Python to pack any remaining arguments into a tuple that’s passed to the args parameter.

d. All of the above statements are true.

Answer: b. The following is a valid call to function min

min(88)

Actually, min must be called with at least two arguments.

4.11 Q2: Based on the following function definition that can receive an arbitrary number of arguments:

In [1]: def average(\*args):  
 ...: return sum(args) / len(args)  
 ...:

which of the following statements is false?

a. The parameter name args is used by convention, but you may use any identifier.

b. If the function has multiple parameters, the \*args parameter must be the leftmost one.

c. The following session calls average several times confirming that it works with arbitrary argument lists of different lengths:

In [2]: average(5, 10)  
Out[2]: 7.5  
  
In [3]: average(5, 10, 15)  
Out[3]: 10.0  
  
In [4]: average(5, 10, 15, 20)  
Out[4]: 12.5

d. All of the above are *true*.

Answer: b. If the function has multiple parameters, the \*args parameter must be the *leftmost* one. Actually, if the function has multiple parameters, the \*args parameter must be the *rightmost* one.

* Methods: Functions That Belong to Objects

4.12 Q1: Which of the following statements is false?

a. A method is simply a function that you call on an object using the form

object\_name.method\_name(arguments)

b. The following session calls the string object s’s object’s lower and upper methods, which produce new strings containing all-lowercase and all-uppercase versions of the original string, leaving s unchanged:

In [1]: s = 'Hello'  
  
In [2]: s.lower() # call lower method on string object s  
Out[2]: 'hello'  
  
In [3]: s.upper()  
Out[3]: 'HELLO'

c. After the preceding session, s contains 'HELLO'.

d. All of the above statements are true.

Answer: c. After the preceding session, s contains 'HELLO'. Actually, the calls to lower and upper do not modify the original string so s still contains 'Hello'.

* Scope Rules

4.13 Q1: Which of the following statements is false?

a. Each identifier has a scope that determines where you can use it in your program. For that portion of the program, the identifier is said to be “in scope.”

b. A local variable’s identifier has local scope. It’s “in scope” only from its definition to the end of the program. It “goes out of scope” when the function returns to its caller.

c. A local variable can be used only inside the function that defines it.

d. All of the above statements are correct.

Answer: b. A local variable’s identifier has local scope. It’s “in scope” only from its definition to the end of the program. It “goes out of scope” when the function returns to its caller. Actually, a local variable’s identifier has local scope. It’s “in scope” only from its definition to the end of the *function’s block*. It “goes out of scope” when the function returns to its caller.

4.13 Q2: Which of the following statements is false?.

a. Identifiers defined outside any function (or class) have script scope—these may include functions, variables and classes.

b. Variables with global scope are known as global variables.

c. Identifiers with global scope can be used in a .py file or interactive session anywhere after they’re defined.

d. All of the above statements are true.

Answer: a. Identifiers defined outside any function (or class) have *script* scope—these may include functions, variables and classes. Actually, identifiers defined outside any function (or class) have *global* scope—these may include functions, variables and classes.

4.13 Q3: Which of the following statements is false?

a. You can access a global variable’s value inside a function.

b. By default, you cannot modify a global variable in a function—when you first assign a value to a variable in a function’s block, Python creates a new local variable.

c. To modify a global variable in a function’s block, you must use a global statement to declare that the variable is defined in the global scope.

d. All of the above statements are true.

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

* **import**: A Deeper Look

4.14 Q1: Which of the following statements is false.

a. You can import all identifiers defined in a module with a wildcard import of the form

from modulename import \*

b. A wildcard import makes all of the module’s identifiers available for use in your code.

c. Importing a module’s identifiers with a wildcard import can lead to subtle errors—it’s considered a dangerous practice that you should avoid.

d. The following session is a safe use of a wildcard import:

In [1]: e = 'hello'  
  
In [2]: from math import \*  
  
In [3]: e  
Out[3]: 2.718281828459045

Answer: d. The following session is a safe use of a wildcard import:

In [1]: e = 'hello'  
  
In [2]: from math import \*  
  
In [3]: e  
Out[3]: 2.718281828459045

Actually, this session is a classic example of why you should not use wildcard imports. Initially, we assign the string 'hello' to a variable named e. After executing snippet [2] though, the variable e is replaced, possibly by accident, with the math module’s constant e, representing the mathematical floating-point value *e*.

4.14 Q2: Which of the following statements is false?

a. Sometimes it’s helpful to import a module and use an abbreviation for it to simplify your code. The import statement’s as clause allows you to specify the name used to reference the module’s identifiers. For example, we can import the statistics module and access its mean function as follows:

In [1]: import statistics as stats  
  
In [2]: grades = [85, 93, 45, 87, 93]  
  
In [3]: stats.mean(grades)  
Out[3]: 80.6

b. import…as is frequently used to import Python libraries with convenient abbreviations, like stats for the statistics module.

c. The numpy module is typically imported with

import numpy as npy

d. Typically, when importing a module, you should use import or import…as statements, then access the module through the module name or the abbreviation following the as keyword, respectively. This ensures that you do not accidentally import an identifier that conflicts with one in your code.

Answer: c. The numpy module is typically imported with

import numpy as npy

Actually, the numpy module is typically imported with

import numpy as np

* Passing Arguments to Functions: A Deeper Look

4.15 Q1: Which of the following statements is false?

a. With pass-by-value, the called function receives a copy of the argument’s value and works exclusively with that copy. Changes to the called function’s copy do not affect the original variable’s value in the caller.

b. With pass-by-reference, the called function can access the argument’s value in the caller directly and modify the value if it’s mutable.

c. Python arguments are always passed by value.

d. When a function call provides an argument, Python copies a reference to the argument object—not the object itself—into the corresponding parameter—functions often manipulate large objects—frequently copying them would consume large amounts of computer memory and significantly slow program performance.

Answer: c. Python arguments are always passed by value. Actually, Python arguments are always passed by reference—some people call this pass-by-object-reference, because “everything in Python is an object.”

4.15 Q2: Which of the following statements is false?

a. You interact with an object via a reference, which behind the scenes is that object’s address (or location) in the computer’s memory—sometimes called a “pointer” in other languages.

b. After the following assignment the variable x contains the value 7:

x = 7

c. After the assignment in part b), you might say that x “points to” (that is, "references") the object containing 7.

d) All of the above statements are false.

Answer: b. After the following assignment the variable x contains the value 7:

x = 7

Actually, the variable x contains a *reference* to the value 7.

4.15 Q3: Which of the following statements is false?

a. Two objects can reside at the same memory address.

b. Though we can’t see an object’s address, we can use the built-in id function to obtain a unique int value which identifies only that object while it remains in memory.

c. The integer result of calling id is known as the object’s identity.

d. No two objects in memory can have the same identity.

Answer: a. Two objects can reside at the same memory address. Actually, no two separate objects can reside at the same address in memory, so every object in memory has a *unique address*.

* Function-Call Stack

4.16 Q1: Which of the following statements is false?

a. For each function call, the interpreter pushes a stack frame onto the stack. This entry contains the return location that the called function needs so it can return control to its caller. When the function finishes executing, the interpreter pops the function’s stack frame, and control- transfers to the return location that was popped.

b. The top stack frame always contains the information the currently executing function needs to return control to its caller.

c. If before a function returns it makes a call to another function, the interpreter pushes a stack frame for that function call onto the stack. Thus, the return address required by the newly called function to return to its caller is now on top of the stack.

d. All of the above statements are true.

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

4.16 Q2: Which of the following statements is false?

a. Most functions have one or more parameters and possibly local variables that need to exist while the function is executing, remain active if the function makes calls to other functions, and “go away” when the function returns to its caller.

b. A called function’s stack frame is the perfect place to reserve memory for the function’s local variables.

c. A called function’s stack frame is popped when the function is called and exists while the function is executing.

d. When a function returns, it no longer needs its local variables, so its stack frame is popped from the stack, and its local variables no longer exist.

Answer: c. A called functions stack frame is popped when the function is called and exists while the function is executing. Actually, a called functions stack frame is *pushed* when the function is called and exists while the function is executing.

4.16 Q3: Which of the following statements is false?

a. The principle of least privilege states that code should be granted only the amount of privilege and access that it needs to accomplish its designated task, but no more.

b. An example of the principle of least privilege is the scope of a local variable, which should not be visible when it’s not needed. This is why a function’s local variables are placed in stack frames on the function-call stack, so they can be used by that function while it executes and go away when it returns.

c. Once a stack frame is popped, the memory that was occupied by it can be reused for new stack frames.

d. Functions can see each other’s local variables by accessing each other’s stack frames.

Answer: d. Functions can see each other’s local variables by accessing each other’s stack frames. Actually, there is no access between stack frames, so functions cannot see each other’s local variables.

* Functional-Style Programming

4.17 Q1: Which of the following statements is false?

a. Like other popular languages, such as Java and C#, Python is a purely functional language.

b. Functional-style programs can be easier to parallelize to get better performance on today’s multi-core processors.

c. You’ve already used list, string and built-in function range iterators with the for statement, and several reductions (functions sum, len, min and max).

d. All of the above statements are true.

Answer: a. Like other popular languages, such as Java and C#, Python is a purely functional language. Actually, like other popular languages, such as Java and C#, Python *is not* a purely functional language—rather, it offers “functional-style” features that help you write code which is less likely to contain errors, more concise and easier to read, debug and modify.

4.17 Q2: Which of the following statements is false?

a. Functional-style programming lets you simply say what you want to do. It hides many details of how to perform each task.

b. Typically, library code handles the how for you. As you’ll see, this can eliminate many errors.

c. Consider the for statement in many other programming languages. Typically, you must specify all the details of counter-controlled iteration: a control variable, its initial value, how to increment it and a loop-continuation condition that uses the control variable to determine whether to continue iterating. This style of iteration is known as external iteration and is error-prone.

d. Functional-style programming emphasizes mutability—it uses only operations that modify variables’ values.

Answer: d. Functional-style programming emphasizes mutability—it uses only operations that modify variables’ values. Actually, functional-style programming emphasizes immutability—it avoids operations that modify variables’ values.

4.17 Q3: Which of the following statements is false?

a. In pure functional programming languages you focus on writing pure functions. A pure function’s result depends only on the argument(s) you pass to it. Also, given particular arguments, a pure function always produces the same result. For example, built-in function sum’s return value depends only on the iterable you pass to it.

b. Pure functions can have side effects—for example, if you pass a mutable list to a pure function, the list can contain different values before and after the function call.

c. The following session demonstrates that when you call the pure function sum, it does not modify its argument.

In [1]: values = [1, 2, 3]  
  
In [2]: sum(values)  
Out[2]: 6  
  
In [3]: sum(values) # same call always returns same result  
Out[3]: 6  
  
In [4]: values  
Out[4]: [1, 2, 3]

d. Functions are objects that you can pass to other functions as data.

Answer: b. Pure functions can have side effects—for example, if you pass a mutable list to a pure function, the list can contain different values before and after the function call. Actually, pure functions do not have side effects. For example, even if you pass a mutable list to a pure function, the list will contain the same values before and after the function call.

* Intro to Data Science: Measures of Dispersion

4.18 Q1: Which of the following statements is false?

a. When we’re talking about a group, the entire group is called the population.

b. Sometimes a population is quite large, such as the people likely to vote in the next U.S. presidential election—in excess of 100,000,000 people.

c. For practical reasons, polling organizations trying to predict who will become the next president work with carefully selected small subsets of the population known as samples. Many of the polls in the 2016 election had sample sizes of about 1000 people.

d. Measures of dispersion help you understand how concentrated the values are.

Answer: d. Measures of dispersion help you understand how concentrated the values are. Actually, measures of dispersion help you understand how spread out the values are.

4.18 Q2: Which of the following statements is false?

a. The standard deviation is the square root of the variance (in this case, 1.5), which tones down the effect of the outliers.

b. The smaller the variance and standard deviation are, the further the data values are from the mean and the greater overall dispersion (that is, spread) there is between the values and the mean.

c. The following code calculates the population standard deviation with the statistics module’s pstdev function:

statistics.pstdev([1, 3, 4, 2, 6, 5, 3, 4, 5, 2])

d. All of the above statements are true.

Answer: b. The smaller the variance and standard deviation are, the further the data values are from the mean and the greater overall dispersion (that is, spread) there is between the values and the mean. Actually, the smaller the variance and standard deviation are, the closer the data values are to the mean and the **less overall dispersion** (that is, spread) there is between the values and the mean.