Intro to Python for Computer Science and Data Science 1st edition Deitel Test Bank

SKU: 9780135404676-TEST-BANK

Introduction to Python Programming

2.1 Introduction

No questions.

2.2 Variables and Assignment Statements

2.2 Q1: Which of the following statements is *false*?
a. Variables store values for later use in your code.
b. The following statement creates **x** and uses the assignment symbol (**=**) to give **x** a value.

```
x = 7
```

c. Every statement stops at the end of the line in which it begins.
d. Once **x** and **y** are created and assigned values, you can use the values in expressions like:

```
x + y
```

Answer: c. Every statement stops at the end of the line in which it begins. Actually, it's possible for statements to span more than one line.

2.2 Q2: Which of the following statements is *false*?
a. The following assignment statement adds the values of variables **x** and **y** and assigns the result to the variable **total**:

```
total = x + y
```

b. The preceding snippet is read, "**total** is assigned the value of **x + y**."
c. The **=** symbol is the assignment operator.
d. The right side of the **=** symbol always executes first, then the result is assigned to the variable on the symbol's left side.
Answer: c. The = symbol is the assignment operator. Actually, the = symbol is *not* an operator in Python.

2.2 Q3: Which of statements a), b) or c) is *false*?
a. A variable name, such as **x**, is an identifier.
b. Each identifier may consist of letters, digits and underscores (_) but may not begin with a digit.
c. Python is case insensitive, so **number** and **Number** are the same identifier despite the fact that one begins with a lowercase letter and the other begins with an uppercase letter.
d. All of the above statements are true.

Answer: c. Python is case insensitive, so `number` and `Number` are the same identifier despite the fact that one begins with a lowercase letter and the other begins with an uppercase letter. Actually, Python is *case sensitive*, so `number` and `Number` are *different* identifiers because one begins with a lowercase letter and the other begins with an uppercase letter.

2.2 Q4: The value 10.5 is a(n) _____ (that is, a number with a decimal point).
a. integer
b. string
c. floating-point number
d. None of the above
Answer: c. floating-point number.

2.2 Q5: Which of the following statements a), b) and c) is *false*.
a. Python's **`type`** built-in function determines a value's type.
b. A function performs its task when you call it by writing its name, followed by square brackets, **`[]`**.
c. The parentheses after a function name in a function call contain the function's argument—the data that the **`type`** function needs to perform its task.
d. All of the above statements are *true*.
Answer: b. A function performs its task when you call it by writing its name, followed by square brackets, `[]`. Actually, a function performs its task when you call it by writing its name, followed by parentheses, `()`.

2.3 Arithmetic

2.3 Q1: Which of the following statements a), b) or c) is *false*?
a. Python uses the asterisk (**`*`**) as the multiplication operator (e.g., **`7 * 4`**).
b. The exponentiation (**`**`**) operator raises one value to the power of another (e.g., **`2 ** 10`**).
c. To square a number, you can use the exponent **`1/2`** (that is, **`0.5`**), as in:

```
9 ** (1 / 2)
```

d. All of the above statements are *true*.
Answer: c. To square a number, you can use the exponent `1/2` (that is, `0.5`). Actually, to square a number, use the exponent `2`. The exponent `1/2` takes the *square root* of the number.

2.3 Q2: Which of the following statements is *false*?
a. True division (**`/`**) divides a numerator by a denominator and yields a floating-point number with a decimal point.

b. Floor division (`//`) divides a numerator by a denominator, yielding the highest integer that's not greater than the result. Python truncates (discards) the fractional part.
c. The expression `-13 / 4` evaluates to `-3.25`.
d. The expression `-13 // 4` evaluates to `-3`.
Answer: d. The expression `-13 // 4` evaluates to `-3`. Actually, the expresion `-13 // 4` evaluates to `-4`.

2.3 Q3: Which of the following statements is *false*?
a. Dividing by zero with `/` or `//` is not allowed and results in an exception—a sign that a problem occurred:
b. Python reports an exception with a traceback.
c. Most exception names end with `Exception`.
d. In IPython interactive mode, the line that begins with `---->` shows the code that caused the exception.
Answer: c. Most exception names end with `Exception`. Actually, most exception names end with `Error`.

2.3 Q4: Which of the following statements is *false*?
a. The value of the following expression is `3`:

```
17 % 5
```

b. The value of the following expression is `0.5`:

```
7.5 % 3.5
```

c. You can use the remainder operator for applications such as determining whether one number is a multiple of another.
d. You can use the remainder operator to determine whether a number is odd or even.
Answer: a. The value of the following expression is `3`:

```
17 % 5
```

Actually, the value of the expression is 2.

2.3 Q5: Which of the following statements is *false*?
a. Algebraic expressions must be typed in straight-line form using Python's operators.
b. The following code multiplies `10` times the quantity `5 + 3`, yielding the result 80:

```
10 * (5 + 3)
```

c. Parentheses in an expression are said to be redundant (unnecessary) if removing them yields the *same* result.
d. The parentheses in part b) above are redundant.

Answer: d. The parentheses in part b) above are redundant. Actually, without the parentheses, the value of the expression is *different*—it's now 53, so the parentheses are *not* redundant.

2.3 Q6: Which of the following statements is *false*?
a. Python applies the operators in arithmetic expressions according to the rules of operator precedence, which are generally the same as those in algebra.
b. Parentheses have the highest level of precedence, so expressions in parentheses evaluate first—thus, parentheses may force the order of evaluation to occur in any sequence you desire.
c. In expressions with **nested parentheses**, such as `(a / (b - c))`, the expression in the *innermost* parentheses (that is, `b - c`) evaluates first.
d. If an expression contains several exponentiation operations, Python applies them from left to right.
Answer: d. If an expression contains several exponentiation operations, Python applies them from left to right. Actually, if an expression contains several exponentiation operations, Python applies them from *right to left*.

2.3 Q7: Which of the following statements is *false*?
a. When we say that Python applies certain operators from left to right, we are referring to the operators' grouping.
b. In the following expression, the addition operators (`+`) group as if we parenthesized the expression as `a + (b + c)`:

```
a + b + c
```

c. All Python operators of the same precedence group left-to-right except for the exponentiation operator (`**`), which groups right-to-left.
d. You can use redundant parentheses to group subexpressions to make expressions clearer. For example, the second-degree polynomial

```
y = a * x ** 2 + b * x + c
```

can be parenthesized, for clarity, as

```
y = (a * (x ** 2)) + (b * x) + c
```

`Answer:` b. In the following expression, the addition operators (`+`) group as if we parenthesized the expression as `a + (b + c)`:

```
a + b + c
```

Actually, the addition operators (`+`) group from left to right as if we parenthesized the expression as `(a + b) + c`.

2.3 Q8: Which of the following statements about arithmetic operators is *false*?
a. Each arithmetic operator may be used with integers and floating-point numbers.

b. If both operands are integers, the result is an integer—except for the true-division (`/`) operator, which always yields a floating-point number.
c. If both operands are floating-point numbers, the result is a floating-point number.
d. Expressions containing an integer and a floating-point number are mixed-type expressions—these always produce an integer.
Answer: d. Expressions containing an integer and a floating-point number are mixed-type expressions—these always produce an integer.
Actually, these mixed-type expressions always produce a *floating-point number*.

2.4 Function `print` and an Intro to Single- and Double-Quoted Strings

2.4 Q1: Which of the following statements is *false*?
a. When you evaluate expressions in interactive mode, the text that **`print`** displays is preceded by **`Out[]`** with a snippet number in the square brackets.
b. **`print`** does not display a string's quotes, though there is a way to display quotes in strings.
c. You also may enclose a string in double quotes (**`"`**), as in:

```
print("Welcome to Python!")
```

but Python programmers generally prefer single quotes.
d. When print completes its task, it positions the screen cursor at the beginning of the next line.
Answer: a. When you evaluate expressions in interactive mode, the text that `print` displays is preceded by `Out[]` with a snippet number in the square brackets.
Actually, unlike when you evaluate expressions in interactive mode, the text that `print` displays *is not* preceded by `Out[]` with a snippet number in the square brackets.

2.4 Q2: Which of the following statements a), b) and c) is *false*?
a. When a backslash (\) appears in a string, it's known as the escape character.
b. The backslash and the character immediately following it form an escape sequence. For example, **`\n`** represents the newline character escape sequence, which tells **`print`** to move the output cursor to the next line.
c. Placing two *backslashes* back-to-back tells **`print`** to display a blank line.
d. All of the above statements are *true*.
Answer: c. Placing two *backslashes* back-to-back tells `print` to display a blank line.
Actually, placing two *backslashes* back-to-back tells print to display a *backslash*.

2.5 Triple-Quoted Strings

2.5 Q1: Triple-quoted strings are used to create:
a. multiline strings,
b. strings containing single or double quotes
c. docstrings, which are the recommended way to document the purposes of certain program components.
d. All of the above
Answer: d. All of the above.

2.5 Q2: Which of the following statements is *false*?
a. A string delimited by single quotes may include double-quote characters:

```
print('Display "hi" in quotes')
```

b. A string delimited by double quotes may include single quote characters:

```
print("Display the name O'Brien")
```

c. A string delimited by double quotes may not include double quotes.
d. To avoid using **\'** and **\"** inside strings, you can enclose such strings in triple quotes:
Answer: c. A string delimited by double quotes may not include double quotes. Actually, a string delimited by double quotes may not include double quotes, *unless* you use the \" escape sequence:

```
print("Display \"hi\" in quotes")
```

2.6 Getting Input from the User

2.6 Q1: When Python "adds" the *string* values **'7'** and **'3'**, it produces:
a. the integer **10**
b. the string **'10'**
c. the string **'73'**
d. the integer **73**.
Answer: c. the string '73'

2.6 Q2: What does the **int** function attempt to do in the following code?

```
value = input('Enter an integer: ')
value = int(value)
```

a. Convert an integer to a string.
b. Convert a string to an integer.
c. Convert a non-object to an object.
d. None of the above.
Answer: b. Convert a string to an integer.

2.7 Decision Making: The `if` Statement and Comparison Operators

2.7 Q1: Which of the following statements a), b) and c) is *false*?
a. A condition is a Boolean expression with the value **`True`** or **`False`**.
b. **`True`** and **`False`** are keywords—words that Python reserves for its language features.
c. Using a keyword as an identifier causes a **`ValueError`**.
d. None of the above statements is false.
Answer: c. Using a keyword as an identifier causes a `ValueError`. Actually, using a keyword as an identifier causes a `SyntaxError`.

2.7 Q2 Which of the following statements is *true*?
a. The **`if`** statement uses a condition to decide whether to execute a statement (or a group of statements).
b IPython interactive mode is helpful for executing brief code snippets and seeing immediate results.
c. When you have many statements to execute as a group, you typically write them as a script stored in a file with the **`.py`** (short for Python) extension.
d. All of the above statements are *true*.
Answer: d. All of the above statements are *true*.

2.7 Q3: Which of the following statements is *false*?
a. A line that begins with the hash character (**`#`**) is a comment.
b. A comment must begin the line with a hash character (**`#`**).
c. Comments do not cause the computer to perform any action when the code executes.
d. The *Style Guide for Python Code* states that each script should start with a docstring that explains the script's purpose,
Answer: b. A comment must begin the line with a hash character (#). Actually, a comment also can begin to the right of the code on a given line and continue until the end of that line. Such a comment documents the code to its left.

2.7 Q4: Which of the following statements is *false*?
a. You use blank lines and space characters to make code easier to read.
b. Together, blank lines, space characters and tab characters are known as white space.
c. Python ignores all white space.
d. Python allows you to split long code lines in parentheses without using continuation characters.

Answer: c. Python ignores all white space. Actually, Python ignores *most* white space—some indentation (which is created with whitespace characters) is required. [Also, white space in strings is significant.]

2.7 Q5: You may spread a lengthy statement over several lines with the _____ continuation character.
a. **@**
b. ****
c. **%**
d. **^**
**Answer: b. **

2.7 Q6:Which of the following statements is *false*?
a. The following **if** statement uses the **==** comparison operator to determine whether the values of variables **number1** and **number2** are equal:

```
if number1 == number2:
    print(number1, 'is equal to', number2)
```

b. Each **if** statement consists of the keyword **if**, the condition to test, and a colon (**:**) followed by an indented body called a suite.
c. Each suite contains zero or more statements.
d. Forgetting the colon (**:**) after the condition is a common syntax error.
Answer: c. Each suite contains zero or more statements. Actually, each suite must contain *one or more* statements.

2.7 Q7: Which of the following statements is *false*?
a. Python requires you to indent the statements in suites.
b. Incorrect indentation of the statements in a suite can cause errors.
c. Using the assignment symbol (**=**) instead of the equality operator (**==**) in an **if** statement's condition is a common logic error.
d. Using **==** in place of **=** in an assignment statement can lead to subtle problems.
Answer: c. Using the assignment symbol (=) instead of the equality operator (==) in an if statement's condition is a common logic error. Actually, Using the assignment symbol (=) instead of the equality operator (==) in an if statement's condition is a common *syntax* error.

2.7 Q8: The chained comparison **3 < y <= 11** is *false* for which of the following values of **y**:
a. 11
b. 10
c. 4
d. 3
Answer: d. 3

2.8 Objects and Dynamic Typing

2.8 Q1: Assume **x** is **3**, **y** is **7.1** and **z** is **'11.5'**. Which of the following statements is *incorrect*?
a. **type(x)** is **int**
b. the value of **z** is **11.5**
c. the value of **y** is **7.1**
d. **type(z)** is **str**
Answer: b. the value of z is 11.5. Actually, the value of z is '11.5'.

2.8 Q2: Which of the following statements is *false*?
a. Assigning an object to a variable binds (associates) that variable's name to the object. You can then use the variable in your code to access the object's value.
b. After the following snippet's assignment, the variable **x** refers to the integer object containing **7**.

```
x = 7
```

c. The following statement changes **x**'s value:

```
x + 10
```

d. The following statement changes **x**'s value:

```
x = x + 10
```

Answer: c. The following statement changes x's value:

```
x + 10
```

Actually, this statement *does not* change x's value—you can change x's value via assignment as shown in Part (d).

2.8 Q3: Which of the following Python concepts are demonstrated directly or indirectly by the following code:

```
In [1]: type(x)
Out[1]: int

In [2]: x = 4.1

In [3]: type(x)
Out[3]: float

In [4]: x = 'dog'

In [5]: type(x)
Out[5]: str
```

a. Python uses dynamic typing—it determines the type of the object a variable refers to while executing your code.
b. Over its lifetime a variable can be bound to different variables, even variables of different types.
c. Python creates objects in memory and removes them from memory as necessary. This removal process—called garbage collection—helps ensure that memory is available for new objects you create.
d. All of the above.
Answer: d. All of the above.

2.9 Intro to Data Science: Basic Descriptive Statistics

2.9 Q1: Which of the following statements about descriptive statistics is *false*?
a. The minimum is the smallest value in a collection of values.
b. The range is the values starting with the minimum and ending with the maximum.
c. The count is the number of values in a collection.
d. Measures of dispersion (also called measures of variability), such as count, help determine how spread out values are.
Answer: d. Measures of dispersion (also called measures of variability), such as count, help determine how spread out values are. Actually, measures of dispersion (also called measures of variability), such as *range*, help determine how spread out values are.

2.9 Q2: Which of the following statements is *incorrect*:
a. `min(17, 19, 23, 29, 31, 37, 43)` is a valid call to built-in function `min`,
b. The *range* of values is simply the minimum through the maximum value.
c. Much of data science is devoted to getting to know your data.
d. All of the above are correct.
Answer: d. All of the above are correct.

2.9 Q3: Which of the following statements about functional-style programming is *false*?
a. With functional-style programming capabilities you can write code that is more concise, clearer and easier to debug—that is, find and correct errors.
b. The `min` and `max` functions are examples of a functional-style programming concept called reduction. They reduce a collection of values to a single value.
c. Other reductions you'll see include the sum, average, variance and standard deviation of a collection of values.
d. All reductions are built into Python—you cannot define custom reductions.
Answer: d. All reductions are built into Python—you cannot define custom reductions.
Actually, you can define custom reductions.