**Strings: A Deeper Look**

* Introduction

8.1 Q1: Which of the following statements is false?

a. Strings support many of the same sequence operations as sets, lists and tuples.

b. Strings, like tuples, are immutable.

c. The re module can be used to match patterns in text.

d. All of the above statements are true.

Answer: a. Strings support many of the same sequence operations as sets, lists and tuples. Actually, strings support many of the same sequence operations as lists and tuples—*sets are not sequences*.

* Formatting Strings
* Presentation Types

8.2 Q1: Which of the following statements is false?

a. When you specify a placeholder for a value in an f-string, Python assumes the value should be displayed as a string unless you specify another type. For example, the following f-string formats the float value 17.489 rounded to the hundredths position:

f'{17.489:.2f}'

b. Python supports precision only for floating-point and Decimal values.

c. Formatting is type dependent—if you try to use .2f to format a string like 'hello', a NameError occurs.

d. The presentation type f in the format specifier .2f is required.

Answer: c. Formatting is type dependent—if you try to use .2f to format a string like 'hello', a NameError occurs. Actually, a *ValueError* occurs.

8.2 Q2 Which of the following statements is false?

a. The d presentation type in the following f-string formats strings as integer values:

f'{10:d}'

b. The integer presentation types b, o and x or X format integers using the binary, octal or hexadecimal number systems, respectively.

c. The c presentation type in the following f-string formats an integer character code as the corresponding character:

f'{65:c} {97:c}'

d. If you do not specify a presentation type, as in the second placeholder below, non-string values like the integer 7 are converted to strings:

f'{"hello":s} {7}'

Answer: a. The d presentation type in the following f-string formats strings as integer values:

f'{10:d}'

Actually, the d presentation type formats integer values as strings.

8.2 Q3: Which of the following statements is false?

a. For extremely large and small values of floating-point and Decimal types, exponential (scientific) notation can be used to format the values more compactly.

b. The following interactive session shows the difference between f and e for a large value, each with three digits of precision to the right of the decimal point:

In [1]: from decimal import Decimal  
  
In [2]: f'{Decimal("10000000000000000000000000.0"):.3f}'  
Out[2]: '10000000000000000000000000.000'  
  
In [3]: f'{Decimal("10000000000000000000000000.0"):.3e}'  
Out[3]: '1.000e+25'

c. For the e presentation type in snippet [3] of Part (c), the formatted value 1.000e+25 is equivalent to 1.000 x 1025. If you prefer a capital E for the exponent, use the E presentation type rather than e.

d. All of the above statements are true.

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

* Field Widths and Alignment

8.2 Q4: Which of the following statements is false?

a. By default, the following f-strings right-align the numeric values 27 and 3.5 and left-align the string "hello":

f'[{27:10d}]'  
f'[{3.5:10f}]'  
f'[{"hello":10}]'

b. Python formats float values with six digits of precision to the right of the decimal point by default.

c. For values that have fewer characters than the field width, the remaining character positions are filled with spaces.

d. Values with more characters than the field width cause a ValueError.

Answer: d. Values with more characters than the field width cause a ValueError. Actually, values with more characters than the field width use as many character positions as they need.

* Numeric Formatting

8.2 Q5: Which of the following statements is false?

a. Sometimes it’s desirable to force the sign on a positive number, as in the following f-string:

f'[{27:+10d}]'

b. The + before the field width in Part (a) specifies that a positive number should be preceded by a +. A negative number always starts with a -.

c. To fill the remaining characters of the field with 0s rather than spaces, place a 0 before the field width (and after the + if there is one), as in:

f'[{27:+010d}]'

d. All of the above statements are true.

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

8.2 Q6: Which of the following statements is false?

a. A space before the presentation type in an f-string indicates that positive numbers should show a space character in the sign position. This is useful for aligning positive and negative values for display purposes, as in:

In [1]: print(f'{27:d}\n{27: d}\n{-27: d}')  
27  
 27  
-27

b. If a field width is specified with a space flag, the space should appear before the field width.

c. You can format numbers with thousands separators by using a comma (,), as in the following f-strings:

f'{,12345678:d}'  
f'{,123456.78:.2f}'

d. All of the above statements are true.

Answer: c. You can format numbers with thousands separators by using a comma (,), as in the following f-strings:

f'{,12345678:d}'  
f'{,123456.78:.2f}'

Actually, c. You can format numbers with thousands separators by using a comma (,), as in the following f-strings:

f'{12345678:,d}'  
f'{123456.78:,.2f}'

* String’s format Method

8.2 Q7: Which of the following statements is false?

a. You’ll encounter the format method in code written prior to Python 3.6. You’ll often see the format method in the Python documentation and in the many Python books and articles written before f-strings were introduced.

b. It’s recommend that you use the newer f-string formatting.

c. You call method format on a format string containing curly brace ({}) placeholders, possibly with format specifiers. You pass to the method the values to be formatted. In a placeholder, if there’s a format specifier, you precede it by a colon (:), as in f-strings. The result of the format call is a new string containing the formatted results.

d. The following code formats the float value 17.489 rounded to the hundredths position:

'{:.3f}'.format(17.489)

Answer: d. The following code formats the float value 17.489 rounded to the hundredths position:

'{:.3f}'.format(17.489)

Actually, the following code formats the float value 17.489 rounded to the hundredths position:

'{:.2f}'.format(17.489)

8.2 Q8: Which of the following statements is false?

a. A format string may contain multiple placeholders, in which case the format method’s arguments correspond to the placeholders from left to right:

*'*{} {}'.format('Amanda', 'Cyan')

b. The format string can reference specific arguments by their position in the format method’s argument list, starting with position 0, as in:

*'*{0} {0} {1}'.format('Happy', 'Birthday')

You can reference each argument as often as you like and in any order.

c. You can reference keyword arguments by their keys in the placeholders, as in:

'{first} {last}'.format(first='Amanda', last='Gray')  
'{last} {first}'.format(first='Amanda', last='Gray')

d. All of the above statements are true.

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

* Concatenating and Repeating Strings

8.3 Q1: Consider the following code:

In [1]: s1 = 'happy'  
  
In [2]: s2 = 'birthday'  
  
In [3]: s1 += ' ' + s2  
  
In [4]: s1  
Out[4]: 'happy birthday'  
  
In [5]: symbol = '>'  
  
In [6]: symbol \*= 5  
  
In [7]: symbol  
Out[7]: '>>>>>'

Which snippet(s) in this interactive session appear to modify existing strings, but actually create new string objects?

a. Only snippet [3].

b. Only snippet [5].

c. Both snippets [5] and [6].

d. Both snippets [3] and [6].

Answer: d. Both snippets [3] and [6].

* Stripping Whitespace from Strings

8.4 Q1: Based on the string

sentence = '\t \n This is a test string. \t\t \n'

which of the following statements is false?

a. There are several string methods for removing whitespace from the ends of a string. Because strings are immutable each returns a new string leaving the original unmodified.

b. The following code uses string method strip to remove the leading and trailing whitespace from sentence:

sentence.strip()

c. The following code snippets first use method lstrip to remove only leading whitespace from sentence:

sentence.lstrip()

then use method rstrip to remove only trailing whitespace:

sentence.rstrip()

d. Methods strip, lstrip and rstrip remove only leading and/or trailing spaces.

Answer: d. Methods strip, lstrip and rstrip remove only leading and/or trailing spaces. Actually, these methods remove *all kinds of whitespace, including spaces, newlines and tabs*.

* Changing Character Case

8.5 Q1: Which of the following statements is false?

a. String methods lower and upper can convert strings to all lowercase or all uppercase letters, respectively.

b. Method capitalize copies the original string and returns a new string with only the first letter capitalized (sometimes called sentence capitalization).

c. Method title copies the original string and returns a new string with only the first character of each word capitalized (sometimes called book-title capitalization).

d. All of the above statements are true.

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

* Comparison Operators for Strings

8.6 Q1: Which of the following statements is false?

a. Strings may be compared with the comparison operators. Recall that strings are compared based on their underlying integer numeric values. So uppercase letters compare as less than lowercase letters because uppercase letters have lower integer values.

b. You can check character codes with ord. For example, the following code displays the values 65 and 97 for A and a, respectively:

print(f'A: {ord("A")}; a: {ord("a")}')

c. In the following interactive session that compares the strings 'Orange' and 'orange', the outputs of snippets [2] and [6] (marked as ???) are False and True?

In [1]: 'Orange' == 'orange'  
Out[1]: False  
  
In [2]: 'Orange' != 'orange'  
Out[2]: ???  
  
In [3]: 'Orange' < 'orange'  
Out[3]: True  
  
In [4]: 'Orange' <= 'orange'  
Out[4]: True  
  
In [5]: 'Orange' > 'orange'  
Out[5]: False  
  
In [6]: 'Orange' >= 'orange'  
Out[6]: ???

d. All of the above are true.

Answer: c. In the following interactive session that compares the strings 'Orange' and 'orange', the outputs of snippets [2] and [6] (marked as ???) are False and True?

In [1]: 'Orange' == 'orange'  
Out[1]: False  
  
In [2]: 'Orange' != 'orange'  
Out[2]: ???  
  
In [3]: 'Orange' < 'orange'  
Out[3]: True  
  
In [4]: 'Orange' <= 'orange'  
Out[4]: True  
  
In [5]: 'Orange' > 'orange'  
Out[5]: False  
  
In [6]: 'Orange' >= 'orange'  
Out[6]: ???

Actually, the outputs of snippets [2] and [6] are True and False.

* Searching for Substrings

8.7 Q1: Which of the following statements is false?

a. String method count returns the number of times its argument occurs in the string on which the method is called.

b. If you specify as the second argument to method count a start\_index, as in

sentence.count('to', 12)

count searches only the slice string[start\_index:]—that is, from start\_index through end of the string.

c. If you specify as the second and third arguments of method count the start\_index and end\_index, as in

sentence.count('that', 12, 25)

count searches only the slice string[start\_index:end\_index]—that is, from start\_index up to, but not including, end\_index:

d. All of the above statements are true.

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

8.7 Q2: Consider this text from Shakespeare:

soliloquy = 'To be or not to be, that is the question'

Which of the following statements is false?

a. String method index searches for a substring within a string and returns the first index at which the substring is found; otherwise, a ValueError occurs. The following code returns 3:

soliloquy.index('be')

b. String method rindex performs the same operation as index, but searches from the end of the string and returns the last index at which the substring is found; otherwise, a Value-Error occurs. The following code returns 3:

soliloquy.rindex('be')

c. String methods find and rfind perform the same tasks as index and rindex but, if the substring is not found, return -1 rather than causing a Value-Error.

d. All of the above statements are true.

Answer: b. String method rindex performs the same operation as index, but searches from the end of the string and returns the *last* index at which the substring is found; otherwise, a Value-Error occurs. The following code returns 3:

soliloquy.rindex('be')

Actually, this code returns 16.

* Replacing Substrings

8.8 Q1: Which of the following statements is false?

a. Method replace takes two substrings. It searches a string for the substring in its first argument and replaces each occurrence with the substring in its second argument. The method returns a new string containing the results.

b. The following code replaces tab characters with commas:

values = '1\t2\t3\t4\t5'  
values.replace('\t', ', ')

c. Method replace can receive an optional third argument specifying the maximum number of replacements to perform.

d. All of the above statements are true.

Answer: b. The following code replaces tab characters with commas:

values = '1\t2\t3\t4\t5'  
values.replace('\t', ', ')

Actually, this code replaces tab characters with commas and spaces. The following code replaces tab characters with just commas:

values = '1\t2\t3\t4\t5'  
values.replace('\t', ',')

* Splitting and Joining Strings

8.9 Q1: Which of the following statements is false?

a. When you read a sentence, your brain breaks it into individual words, or tokens, each of which conveys meaning.

b. Interpreters like IPython tokenize statements, breaking them into individual components such as keywords, identifiers, operators and other elements of a programming language.

c. Tokens typically are separated by whitespace characters such as blank, tab and newline, though other characters may be used—the separators are known as delimiters.

d. All of the above statements are true.

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

8.9 Q2: Which of the following statements is false?

a. String method split with no arguments tokenizes a string by breaking it into substrings at each whitespace character, then returns a list of tokens.

b. To tokenize a string at a custom delimiter (such as each comma-and-space pair), specify the delimiter string (such as, ', ') that split uses to tokenize the string, as in:

letters = 'A, B, C, D'  
letters.split(', ')

c. If you provide an integer as split’s second argument, it specifies the maximum number of splits. The last token is the remainder of the string after the maximum number of splits. Assuming the string in Part (b), the code

letters.split(', ', 1)

returns

['A', 'B', 'C, D']

d. There is also an rsplit method that performs the same task as split but processes the maximum number of splits from the end of the string toward the beginning.

Answer: c. If you provide an integer as split’s second argument, it specifies the maximum number of splits. The last token is the remainder of the string after the maximum number of splits. Assuming the string in Part (b), the code

letters.split(', ', 1)

returns

['A', 'B', 'C, D']

Actually, the preceding output would be for the second argument 2. The output of the code in Part (c) is ['A', 'B, C, D']

8.9 Q3: Which of the following statements is false?

a. String method partition splits a string into a tuple of three strings based on the method’s separator argument. The three strings are the part of the original string before the separator, the separator itself, and the part of the string after the separator.

b. Consider a string representing a student’s name and grades:

'Amanda: 89, 97, 92'

The following code splits the original string into the student’s name, the separator ': ' and a string representing the list of grades:

'Amanda: 89, 97, 92'.partition(': ')

c. To search for the separator from the end of the string instead, use method rpartition to split. For example, consider the following URL string:

url = 'http://www.deitel.com/books/PyCDS/table\_of\_contents.html'

The following call to rpartition splits 'table\_of\_contents.html' from the rest of the URL:

rest\_of\_url, separator, document = url.rpartition('/')

d. All of the above statements are true.

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

8.9 Q4: Which of the following statements is false?

a. If you read large amounts of text into a string, you might want to split the string into a list of lines based on newline characters. Method splitlines returns a list of strings representing the lines of text split at each newline character in the original string.

b. Python stores multi-line strings with embedded \n characters to represent the line breaks.

c. For the string

lines = """This is line 1  
This is line2  
This is line3"""

the statement

lines.splitlines(True)

keeps the newlines and returns:

['This is line 1', '\nThis is line2', '\nThis is line3']

d. All of the above statements are true.

Answer: c. For the string

lines = """This is line 1  
This is line2  
This is line3"""

the statement

lines.splitlines(True)

keeps the newlines and returns:

['This is line 1', '\nThis is line2', '\nThis is line3']

Actually, in this case, splitlines keeps the newlines at the *end* of each line and returns:

['This is line 1\n', 'This is line2\n', 'This is line3']

* Characters and Character-Testing Methods

8.10 Q1: Which of the following statements is false?

a. Characters (digits, letters and symbols such as $, @, % and \*) are the fundamental building blocks of programs. Every program is composed of characters that, when grouped meaningfully, represent instructions and data that the interpreter uses to perform tasks.

b. Python has separate string and character types.

c. You might use string method isdigit when validating user input that must contain only digits.

d. String method isalnum returns True if the string on which you call the method is alphanumeric—that is, it contains only digits and letters.

Answer: b. Python has separate string and character types. Actually, in Python, a character is simply a **one-character string**.

* Raw Strings

8.11 Q1: Which of the following statements is false?

a. Backslash characters in strings introduce escape sequences—like \n for newline and \t for tab.

b. If you wish to include a backslash in a string, you ordinarily must use two back-slash characters \\.

c. Raw strings—preceded by the character r—treat each backslash as a regular character, rather than the beginning of an escape sequence.

d. To make your code more readable, avoid using raw strings with regular expressions.

Answer: d. To make your code more readable, avoid using raw strings with regular expressions. Actually, raw strings can make your code more readable, particularly when using regular expressions or Windows file paths like

'C:\\MyFolder\\MySubFolder\\MyFile.txt'

* Introduction to Regular Expressions

8.12 Q1: Which of the following statements is false?

a. Sometimes you’ll need to recognize patterns in text, like phone numbers, e-mail addresses, ZIP Codes, web page addresses, Social Security numbers and more. A regular expression string describes a search pattern for matching characters in other strings.

b. Regular expressions can help you extract data from unstructured text, such as social media posts.

c. Regular expressions are also important for ensuring that data is in the correct format before you attempt to process it.

d. All of the above statements are true.

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

8.12 Q2: Which of the following statements is false?

a. Before working with text data, you’ll often use regular expressions to validate the data. b. A U.S. Social Security number contains three digits, a hyphen, two digits, a hyphen and four digits, and adheres to other rules about the specific numbers that can be used in each group of digits.

c. You must create your own regular expressions to ensure that they’ll meet your needs.

d. Many sites provide interfaces in which you can test regular expressions.

Answer: c. You must create your own regular expressions to ensure that they’ll meet your needs. Actually, you’ll rarely need to create your own regular expressions for common items. Websites like https://regex101.com, http://www.regexlib.com and https://www.regular-expressions.info (and others) offer repositories of existing regular expressions that you can copy and use.

8.12 Q3: Which of the following statements is false? In addition to validating data, regular expressions often are used to:

a. Extract data from text (sometimes known as scraping)—e.g., locating all URLs in a web page. [Though, tools like BeautifulSoup, XPath and lxml are preferred for scraping.]

b. Clean data—for example, removing data that’s not required, removing duplicate data, handling incomplete data, fixing typos, ensuring consistent data formats, dealing with outliers and more.

c. Transform data into other formats—for example, reformatting data that was collected as tab-separated or space-separated values into comma-separated values (CSV) format.

d. All of the above statements are true.

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

* re Module and Function fullmatch

8.12 Q4: The following code demonstrates regular expressions containing literal characters—that is, characters that match themselves:

In [1]: import re  
  
In [2]: pattern = '02215'  
  
In [3]: 'Match' if re.fullmatch(pattern, '02215') else 'No match'  
Out[3]: 'Match'  
  
In [4]: 'Match' if re.fullmatch(pattern, '51220') else 'No match'  
Out[4]: 'No match'

Given this code, which of the following statements is false?

a. The fullmatch function’s first argument is the regular expression pattern to match. Any string can be a regular expression. The variable pattern’s value, '02215', contains only literal digits that match themselves in the specified order.

b. The second argument to fullmatch is the string that should entirely match the pattern. If the second argument matches the pattern in the first argument, fullmatch returns an object containing the matching text, which evaluates to True.

c. In snippet [4], the second argument contains the same digits as the regular expression, but in a different order. This is still a match and fullmatch returns an object containing the matching text, which evaluates to True.

d. All of the above statements are true.

Answer: c. In snippet [4], the second argument contains the same digits as the regular expression, but in a different order. This is still a match and fullmatch returns an object containing the matching text, which evaluates to True. Actually, this is not a match, because the digits are not in the correct order, so fullmatch returns None, which evaluates to False.

8.12 Q5: Which of the following statements is false?

a. The \ metacharacter begins each of the predefined character classes, which each match a specific set of characters.

b. The following code ensures that a string contains five consecutive digits:

re.fullmatch(r'\d{5}', '02215')

c. A character class is a regular expression escape sequence that matches one or more character.

d. The following code returns None because '9876' contains only four consecutive digit characters.

re.fullmatch(r'\d{5}', '9876')

Answer: c. A character class is a regular expression escape sequence that matches one or more characters. Actually, a character class is a regular expression escape sequence that matches *exactly one* character.

8.12 Q6: Which of the following statements is false?

a. The regular-expression quantifiers \* and + are greedy—they match as many characters as possible. The regular expression [A-Z][a-z]+ matches 'Al', 'Eva', 'Samantha', 'Benjamin' and any other words that begin with a capital letter followed at least one lowercase letter.

b. The ? quantifier matches zero or one occurrences of a subexpression.

c. The regular expression \d{3,} matches strings containing at least three digits.

d. You can match between n and m (exclusive) occurrences of a subexpression with the {n,m} quantifier. The regular expression \d{3,6} matches strings containing 3 to 5 digits.

Answer: d. You can match *between n and m (exclusive) occurrences* of a subexpression with the {*n*,*m*} quantifier. The regular expression \d{3,6} matches strings containing 3 to 5 digits. Actually, the given regular expression matches strings containing *3 to* *6* digits, the value of m is included in the range.

* Replacing Substrings and Splitting Strings

8.12 Q7: Which of the following statements is false?

a. By default, the re module’s sub function replaces only the first occurrence of a pattern with the replacement text you specify. The statement

re.sub(r'\t', ', ', '1\t2\t3\t4')

returns

'1, 2\t3\t4'

b. The sub function receives three required arguments—the pattern to match (the tab character '\t'), the replacement text (', ') and the string to be searched ('1\t2\t3\t4') and returns a new string.

c. The sub function’s keyword argument count can be used to specify the maximum number of replacements.

d. All of the above statements are true.

Answer: a. By default, the re module’s sub function replaces only the first occurrence of a pattern with the replacement text you specify. The statement

re.sub(r'\t', ', ', '1\t2\t3\t4')

returns

'1, 2\t3\t4'

Actually, by default, the re module’s sub function replaces *all* occurrences of a pattern with the replacement text you specify, so the statement returns

'1, 2, 3, 4'

8.12 Q8: Which of the following statements is false?

a. The split function tokenizes a string, using a regular expression to specify the delimiter, and returns a list of strings.

b. The following code tokenizes a string by splitting it at any comma that’s followed by 0 or more whitespace characters—\s is the whitespace character class and \* indicates zero or more occurrences of the preceding subexpression:

re.split(r',\s\*', '1, 2, 3,4, 5,6,7,8')

c. The following code uses the keyword argument maxsplit to specify the maximum number of splits (in this case, after the 3 splits, the fourth string contains the rest of the original string):

re.split(r',\s\*', '1, 2, 3,4, 5,6,7,8', maxsplit=3)

d. All of the above statements are true.

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

* Other Search Functions; Accessing Matches

8.12 Q9: Which of the following statements is false?

a. Function search looks in a string for the first occurrence of a substring that matches a regular expression and returns a match object (of type SRE\_Match) that contains the matching substring. The match object’s group method returns that substring, as in the following session:

In [1]: import re  
  
In [2]: result = re.search('Python', 'Python is fun')  
  
In [3]: result.group() if result else 'not found'  
Out[3]: 'Python'

b. Function search returns None if the string does not contain the pattern.

c. You can search for a match only at the beginning of a string with function match.

d. All of the above statements are true.

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

8.12 Q10: Which of the following statements is false?

a. Function findall finds every matching substring in a string and returns a list of the matching substrings.

b. The following code extracts all phone numbers of the form ###-###-#### from a string:

contact = 'Wally White, Home: 555-555-1234, Work: 555-555-4321'  
re.findall(r'\d{3}-\d{3}-\d{4}', contact)

c. Function finditer works like findall, but returns a greedy iterable of match objects.

d. For large numbers of matches, using finditer can save memory because it returns one match at a time, whereas findall returns all the matches at once.

Answer: c. Function finditer works like findall, but returns a greedy iterable of match objects. Actually, function finditer works like findall, but returns a *lazy* iterable of match objects.

* Intro to Data Science: Pandas, Regular Expressions and Data Munging

8.13 Q1:Preparing data for analysis is called          or         .

a. data wrangling

b. data munging

c. Both (a) and (b).

d. Neither (a) nor (b).

Answer: c. Both (a) and (b).

8.13 Q2: Which of the following statements is false?

Assume the following two-dimensional list:

contacts = [['Mike Green', 'demo1@deitel.com', '5555555555'],  
 ['Sue Brown', 'demo2@deitel.com', '5555551234']]

a. The following code creates a DataFrame from the list:

contactsdf = pd.DataFrame(contacts,  
 columns=['Name', 'Email', 'Phone'])

b. Part (a) specified column indices via the columns keyword argument but did not specify row indices, so the rows are indexed from 0.

c. The following output shows how the DataFrame is displayed when you evaluate the variable contactsdf IPython—all column values are right aligned by default, as they are in Python:

Name Email Phone  
0 Mike Green demo1@deitel.com 5555555555  
1 Sue Brown demo2@deitel.com 5555551234

d. All of the above statements are true.

Answer: c. The following output shows how the DataFrame is displayed when you evaluate the variable contactsdf IPython—all column values are right aligned by default, as they are in Python:

Name Email Phone  
0 Mike Green demo1@deitel.com 5555555555  
1 Sue Brown demo2@deitel.com 5555551234

Actually, this differs from Python formatting in which numbers in a field are right aligned by default but non-numeric values are left aligned by default.