25.1 String slicing

String slicing basics

Strings are a sequence type, having characters ordered by index from left to right. An ***index*** is an integer matching to a specific position in a string's sequence of characters. An individual character is read using an index surrounded by brackets. Ex: `my_str[5]` reads the character at index 5 of the string `my_str`. Indices start at 0, so index 5 is a reference to the 6th character in the string.

A programmer often needs to read more than one character at a time. Multiple consecutive characters can be read using slice notation. ***Slice notation*** has the form `my_str[start:end]`, which creates a new string whose value contains the characters of `my_str` from indices start to end - 1. If `my_str` is 'Boggle', then `my_str[0:3]` yields string 'Bog'. Other sequence types like lists and tuples also support slice notation.

Figure 25.1.1: String slicing.

```
url = 'http://en.wikipedia.org/wiki/Turing'
domain = url[7:23]  # Read 'en.wikipedia.org' from
url
print(domain)
```

```
en.wikipedia.org
```

The last character of the slice is one location *before* the specified end. Consider the string `my_str = 'John Doe'`. The slice `my_str[0:4]` includes the element at index 0 (J), 1 (o), 2 (h), and 3 (n), but *not* 4, thus yielding 'John'. The space character at index 4 is not included. Similarly, `my_str[4:7]` would yield ' Do', including the space character this time. To retrieve the last character, an end index greater than the length of the string can be used. Ex: `my_str[5:8]` or `my_str[5:10]` both yield the string 'Doe'.

Negative numbers can be used to specify an index relative to the end of the string. Ex: If the variable `my_str` is 'Jane Doe!?', then `my_str[0:-2]` yields 'Jane Doe' because the -2 refers to the second-to-last character '!' (and the character at the end index is not included in the result string).

PARTICIPATION ACTIVITY 25.1.1: Slicing.

Animation content:

```
undefined
```

Animation captions:

1. my_str[0:2] returns a substring of my_str starting at index 0 up to, but not including, index 2.
2. my_str[0:6] returns a substring of my_str starting at index 0 up to, but not including, index 6.
3. my_str[7:10] returns a substring of my_str starting at index 7 up to, but not including, index 10.

PARTICIPATION ACTIVITY 25.1.2: Slicing basics.

Determine the output of the following code:

1)
```
my_str = 'The cat in the
hat'
print(my_str[0:3])
```

Check Show answer

2)
```
my_str = 'The cat in the
hat'
print(my_str[3:7])
```

Check Show answer

Slicing and slicing operations

The Python interpreter creates a new string object for the slice. Thus, creating a slice of the string variable `my_str`, and then changing the value of `my_str`, does not also change the value of the slice.

Figure 25.1.2: A slice creates a new object.

```
my_str = "The cat jumped the brown cow"
animal = my_str[4:7]
print(f'The animal is a {animal}')

my_str = 'The fox jumped the brown llama'
print('The animal is still a', animal)  # animal
variable remains unchanged.
```

```
The animal is a
cat
The animal is
still a cat
```


A programmer often wants to read all characters that occur before or after some index in the string. Omitting a start index, such as in `my_str[:end]` yields the characters from indices 0 to end-1. Ex: `my_str[:5]` reads indices 0-4. Similarly, omitting the end index yields the characters from the start index to the end of the string. Ex: `my_str[5:]` yields all characters at and after index 5.

Use the below tool to experiment with slice notation. After using positive values only, try entering negative start or end indices. Then try omitting either the start or end index.

PARTICIPATION ACTIVITY 25.1.3: String slicing tool.

```
string_var = ' Hey folks!  '
print(string_var[ 1  : 5  ])
```

Output: 'ey f'

H	e	y		f	o	l	k	s	!
	↑				↑				
	start				end				

Variables can also be used in place of literals to specify slice notation start and end indices. Ex: `my_str[x:y]`.

zyDE 25.1.1: Slicing example: omitting start, end indices.

Run the program below.

Load default template...

```
1 usr_text = input('Enter a string: ')
2 print()
3
4 first_half = usr_text[:len(usr_text)//2]
5 last_half = usr_text[len(usr_text)//2:]
6
7 print(f'The first half of the string is "{f
8 print(f'The second half of the string is "{
9
```

```
Hello there. Nice to meet you!
```


Run

Specifying a start index beyond the end of the string, or beyond the end index (like 3:2), yields an empty string. Ex: `my_str[2:1]` is ' '. Specifying an end index beyond the end of the string is equivalent to specifying the end of the string, so if a string's end is 5, then 1:7 or 1:99 are the same as 1:6.

Table 25.1.1: Common slicing operations.

A list of common slicing operations a programmer might use.
Assume the value of my_str is 'http://en.wikipedia.org/wiki/Nasa/'

Syntax	Result	Description
`my_str[10:19]`	wikipedia	Gets the characters in indices 10-18.
`my_str[10:-5]`	wikipedia.org/wiki/	Gets the characters in indices 10-28.
`my_str[8:]`	n.wikipedia.org/wiki/Nasa/	All characters from index 8 until the end of the string.
`my_str[:23]`	http://en.wikipedia.org	Every character up to index 23, but not including my_str[23].
`my_str[:-1]`	http://en.wikipedia.org /wiki/Nasa	All but the last character.

PARTICIPATION ACTIVITY 25.1.4: Slicing.

1) What is the output?

```
my_str =
'http://reddit.com
/r/python'
print(my_str[17:])
```

Check Show answer

2) What is the output?

```
my_str = 'http://reddit.com
/r/python'
protocol = 'http://'
print(my_str[len(protocol):])
```

Check Show answer

The slice stride

Slice notation also provides for a third argument, known as the stride. The ***stride*** determines how much to increment the index after reading each element. For example, `my_str[0:10:2]` reads every other element between 0 and 10. The stride defaults to 1 if not specified.

Figure 25.1.3: Slice stride.


```
numbers = '0123456789'

print(f'All numbers: {numbers[::]}')
print(f'Every even number: {numbers[::2]}')
print(f'Every third number between 1 and 8:
{numbers[1:9:3]}')
```

```
All numbers: 0123456789
Every even number: 02468
Every third number between
1 and 8: 147
```

PARTICIPATION ACTIVITY 25.1.5: Slice stride.

1) What is the output?

```
my_str =
'Agt2t3afc2kjMhagrds!'
print(my_str[0:5:1])
```

Check Show answer

2) What is the output?

```
my_str =
'Agt2t3afc2kjMhagrds!'
print(my_str[::2])
```

Check Show answer

CHALLENGE ACTIVITY 25.1.1: String slicing.

422102.2723990.qx3zqy7

Start

Type the program's output

```
color = 'pearl'
color_slice = color[0:4]
print(color_slice)
```

pear

1 2 3 4 5

Check Next

CHALLENGE ACTIVITY 25.1.2: Slice a rhyme.

Assign sub_lyric by slicing rhyme_lyric from start_index to end_index which are given as inputs.

Sample output with inputs: 4 7

```
cow
```

422102.2723990.qx3zqy7

```
1 start_index = int(input())
2 end_index = int(input())
3 rhyme_lyric = 'The cow jumped over the moon.'
4 sub_lyric = rhyme_lyric[''' Your solution goes here ''']
5 print(sub_lyric)
```


Run

25.2 Advanced string formatting

Field width

A program must commonly display nicely formatted output beyond the ability of basic print usage like `print(x)`. Consider a program that displays a nicely formatted table of soccer player statistics:

Figure 25.2.1: A formatted table of soccer statistics.

```
Player Name      Goals    Games Played     Goals Per Game
-----------------------------------------------------------
Sadio Mane         22          36               0.61
Mohamed Salah      22          38               0.58
Sergio Aguero      21          33               0.64
Jamie Vardy        18          34               0.53
Gabriel Jesus       7          29               0.24
```

Note in the above example how the text is formatted into columns with the contents of each column (except the leftmost column) centered under the column header. A programmer could achieve this careful formatting by placing spaces into their output strings, but each row would require different numbers of spaces depending on the player name (longer names require fewer spaces between the first and second columns).

A format specification may include a ***field width*** that defines the minimum number of characters that must be inserted into the string. If the replacement value is smaller in size than the given field width, then the string is padded with space characters. Field widths set on each column in the example above cause the output to be formatted. A field width is defined in a format specification by including an integer after the colon, as in `{name:16}` to specify a width of 16 characters.

PARTICIPATION ACTIVITY 25.2.1: Field width.

Animation content:

`undefined`

Animation captions:

1. 'Player Name' is inserted into the leftmost part of the first 16-character wide field. 'Goals' is inserted into the leftmost part of the second 8-character wide field.
2. The inserted values align themselves automatically according to the field width.

PARTICIPATION ACTIVITY 25.2.2: Format specification field widths.

1) Complete the format specification to assign a field width of 10 characters.

```
{name:    }
```

Check Show answer

2) Write a complete replacement field that assigns a field with named value "count" a field width of 5.

Check Show answer

CHALLENGE ACTIVITY 25.2.1: Field widths.

422102.2723990.qx3zqy7

Start

Type the program's output

```
name = 'Joe'
print(f'{name:10}')
```

Joe

1 2

Check Next

Aligning text

A format specification can include an ***alignment character*** that determines how a value should be aligned within the width of the field. Alignment is set in a format specification by adding a special character before the field width integer. The basic set of possible alignment options include left-aligned '<', right-aligned '>' and centered '^'.

Figure 25.2.2: Aligning strings within a field.

Consider the following code that prints a table, and how changing the alignment impacts the column organization.

```
names = ['Sadio Mane', 'Gabriel Jesus']
goals = [22, 7]

print(<f-string 1>)         #Replaced in table below
print('-' * 24)
for i in range(2):
    print(<f-string 2>)     #Replaced in table below
```

Alignment type	<f-string 1> <f-string 2>	Output

Left-aligned	`f'{"Player Name":<16}{"Goals":<8}'`
	`f'{names[i]:<16}{goals[i]:<8}'`
		`Player Name     Goals`
		`------------------------`
		`Sadio Mane      22`
		`Gabriel Jesus   7`

Right-aligned	`f'{"Player Name":>16}{"Goals":>8}'`
	`f'{names[i]:>16}{goals[i]:>8}'`
		`     Player Name   Goals`
		`------------------------`
		`      Sadio Mane      22`
		`   Gabriel Jesus       7`

Centered	`f'{"Player Name":^16}{"Goals":^8}'`
	`f'{names[i]:^16}{goals[i]:^8}'`
		`  Player Name    Goals`
		`------------------------`
		`   Sadio Mane      22`
		`Gabriel Jesus     7`

PARTICIPATION ACTIVITY 25.2.3: Aligning text in fields.

For each question, determine the value of the given expression.

1) `f'{"Bob":<5}'`

Check Show answer

2) `f'{"Bob":>5}'`

Check Show answer

3) `f'{"Bob":^5}'`

Check Show answer

4) `f'{"Bob":<5}{1:<2}'`

Check Show answer

5) `f'{"Bob":<5}{1:>2}'`

Check Show answer

Fill

The ***fill character*** is used to pad a replacement field when the string being inserted is smaller than the field width. The default fill character is an empty space `' '`. A programmer may define a different fill character in a format specification by placing the different fill character before the alignment character. Ex: `{score:0>4}` generates "0009" if score is 9 or "0250" if score is 250.

Table 25.2.1: Using fill characters to pad tables.

Format specification	Value of score	Output
`{score:}`	9	`9`
`{score:4}`	9	`   9`
`{score:0>4}`	9	`0009`
`{score:0>4}`	18	`0018`
`{score:0^4}`	18	`0180`

A programmer can set different alignments, widths, and fills on each field to construct nicely formatted output, as demonstrated below.

PARTICIPATION ACTIVITY 25.2.4: Fill characters in strings.

PARTICIPATION ACTIVITY 25.2.5: Fill characters.

1) What's the fill character in the following format specification?

```
{score:*>4}
```

- score
- *
- :
- 4

2) What's the fill character in the following format specification?

```
{score:>4}
```

- ○ >
- ○ *
- ○ 4
- ○ space character

3) If `name = 'Sally'`, what is the result of: `{name:@>8}`?

- ○ Sally@@@
- ○ Sally
- ○ @Sally@@
- ○ @@@Sally
- ○ Sally>>>

Floating-point precision

A programmer commonly wants to set how many digits to the right of a floating-point number to print. The optional ***precision*** component of a format specification indicates how many digits to the right of the decimal should be included in the output of floating types. The precision follows the field width component in the format specification, if a width is specified at all, and starts with a period character. Ex: `f'{1.725:.1f}'` indicates a precision of 1, thus the resulting string would be '1.7'.

If the specified precision is greater than the number of digits available, trailing 0s are appended. Ex: `f'{1.5:.3f}'` results in the string '1.500'. If the specified precision is less than the existing precision in the given number, then the number is rounded. Ex: `f'{1.666:.2f}'` results in the string '1.67'.

Figure 25.2.3: String formatting example: Setting precision of floating-point values.

```
import math
real_pi = math.pi  # math library provides close
approximation of pi
approximate_pi = 22.0 / 7.0  # Approximately
correct pi to within 2 decimal places

print(f'pi is {real_pi}')
print(f'22/7 is {approximate_pi}')
print(f'22/7 looks better like
{approximate_pi:.2f}')
```

```
pi is
3.141592653589793
22/7 is
3.142857142857143
22/7 looks better
like 3.14
```


PARTICIPATION ACTIVITY 25.2.6: Floating-point precision in formatted strings.

Fill in the string that results from evaluating the given expression.

1) `f'{5:.1f}'`

'5.[ ]'

Check Show answer

2) `f'{5:.3f}'`

'5.[ ]'

Check Show answer

3) `f'{5.25:.3f}'`

'[ ]'

Check Show answer

4) `f'{5.2589:.3f}'`

'[ ]'

Check Show answer

5) `f'{5:4.1f}'`

' '

Check Show answer

CHALLENGE ACTIVITY 25.2.2: Format temperature output.

Print `air_temperature` with 1 decimal point followed by C.

Sample output with input: 36.4158102

```
36.4C
```

422102.2723990.qx3zqy7

```
1 air_temperature = float(input())
2
3 ''' Your solution goes here '''
4
```

Run

25.3 String methods

String objects have many useful methods to do things like replacing characters, converting to

lowercase, capitalizing the first character, etc. The methods are made possible due to a string's implementation as a *class*, which for purposes here can just be thought of as a mechanism supporting a set of methods for a particular type of object.

Finding and replacing

A common task for a programmer is to edit the contents of a string. Recall that string objects are immutable -- once created, strings can not be changed. To update a string variable, a new string object must be created and bound to the variable name, replacing the old object. The *replace* string method provides a simple way to create a new string by replacing all occurrences of a substring with a new substring.

- ***replace(old, new)*** -- Returns a copy of the string with all occurrences of the substring old replaced by the string new. The old and new arguments may be string variables or string literals.
- ***replace(old, new, count)*** -- Same as above, except only replaces the first count occurrences of old.

PARTICIPATION ACTIVITY 25.3.1: replace() string method.

Some methods are useful for finding the position of where a character or substring is located in a string:

- ***find(x)*** -- Returns the index of the first occurrence of item x in the string, else returns -1. x may be a string variable or string literal. Recall that in a string, the index of the first character is 0, not 1. If `my_str` is 'Boo Hoo!':
  - `my_str.find('!')  # Returns 7`
  - `my_str.find('Boo')  # Returns 0`
  - `my_str.find('oo')  # Returns 1 (first occurrence only)`
- ***find(x, start)*** -- Same as find(x), but begins the search at index start:
  - `my_str.find('oo', 2)  # Returns 5`
- ***find(x, start, end)*** -- Same as find(x, start), but stops the search at index end - 1:
  - `my_str.find('oo', 2, 4)  # Returns -1 (not found)`
- ***rfind(x)*** -- Same as find(x) but searches the string in reverse, returning the last occurrence in the string.

Another useful function is count, which counts the number of times a substring occurs in the string:

- ***count(x)*** -- Returns the number of times x occurs in the string.
  - `my_str.count('oo')  # Returns 2`

Note that methods such as `find()` and `rfind()` are useful only for cases where a programmer needs to know the exact location of the character or substring in the string. If the exact position is not important, then the `in` membership operator should be used to check if a character or substring is contained in the string:

Figure 25.3.1: Use 'in' to check if a character or substring is contained by another string.

```
if 'batman' in superhero_name:
    # Statements to execute if superhero_name contains 'batman' in any
position.
```

zyDE 25.3.1: String searching example: Hangman.

The following example carries out a simple guessing game, allowing a user a numbe guesses to fill out the complete word.

Load default ter


```
word = 'onomatopoeia'
num_guesses = 10

hidden_word = '-' * len(word)

guess = 1

while guess <= num_guesses and '-' in hidden_word:
    print(hidden_word)
    user_input = input(f'Enter a character (guess #{guess}): ')

    if len(user_input) == 1:
        # Count the number of times the character occurs in the word
        num_occurrences = word.count(user_input)

        # Replace the appropriate position(s) in hidden_word with the actual c
        position = -1
```

```
y
m
n
```

Run

Comparing strings

String objects may be compared using relational operators (<, <=, >, >=), equality operators (==, !=), membership operators (`in`, `not in`), and identity operators (`is`, `is not`).

Evaluation of relational and equality operator comparisons occurs by first comparing the corresponding characters at element 0, then at element 1, etc., stopping as soon as a determination can be made. For an equality (==) comparison, the two strings must have the same length and every corresponding character pair must be the same. For a relational comparison (<, >, etc.), the result will be the result of comparing the ASCII/Unicode values of the first differing character pair.

Table 25.3.1: String comparisons.

Example	Expression result	Why?
`'Hello' == 'Hello'`	True	The strings are exactly identical values
`'Hello' == 'Hello!'`	False	The left hand string does not end with '!'
`'Yankee Sierra' > 'Amy Wise'`	True	The first character of the left side 'Y' is "greater than" (in ASCII value) the first character of the right side 'A'
`'Yankee Sierra' > 'Yankee Zulu'`	False	The characters of both sides match until the second word. The first character of the second word on the left 'S' is not "greater than" (in ASCII value) the first character on the right side 'Z'
`'seph' in 'Joseph'`	True	The substring 'seph' can be found starting at the 3rd position of 'Joseph'
`'jo' in 'Joseph'`	False	'jo' (with a lowercase 'j') is not in 'Joseph' (with an uppercase 'J')

The following animation shows the process of comparing two string variables character by character using their ASCII values. Recall that ASCII values are an integer value representation of a character. 'A' is represented by the integer value 65, 'B' by 66, 'C' by 67, and so on. An ***ASCII table*** provides a quick lookup of ASCII values. There are many ASCII tables available online, for example www.asciitable.com.

PARTICIPATION ACTIVITY 25.3.2: String comparison.

Animation captions:

1. Each comparison uses ASCII values.
2. Values at indexes 0-4 are the same for both student_name and teacher_name.
3. 'J' is greater than 'A', so student_name is greater than teacher_name.

If one string is shorter than the other with all corresponding characters equal, then the shorter string is considered less than the longer string.

The membership operators (`in`, `not in`) provide a simple method for detecting whether a specific substring exists in the string. The argument to the right of the operator is examined for the existence of the argument on the left. Note that reversing the arguments does not work, as 'Jo' is a substring of 'Kay, Jo', but 'Kay, Jo' is not a substring of 'Jo'.

The identity operators (`is`, `is not`) determine whether the two arguments are bound to the same object. A common error is to use an identity operator in place of an equality operator. Ex: A programmer may write `name is 'Amy Adams'`, intending to check if the value of `name` is the same as the literal 'Amy Adams'. Instead, the Python interpreter creates a new string object from the string literal on the right, and compares the identity of the new object to the `name` object, which returns False. Good practice is to always use the equality operator`==` when comparing values.

Figure 25.3.2: Identity vs. equality operators.

```
student_name = input('Enter student
name:\n')

if student_name is 'Amy Adams':
    print('Identity operator: True')
else:
    print('Identity operator: False')

if student_name == 'Amy Adams':
    print('Equality operator: True')
else:
    print('Equality operator: False')
```

```
Enter student name: Amy
Adams
Identity operator: False
Equality operator: True
```


Because comparison uses the encoded values of characters (ASCII/Unicode), comparison may not behave intuitively for some situations. Comparisons are case-sensitive, so 'Apple' does not equal 'apple'. In particular, because the encoded value for 'A' is 65, and for 'a' is 97, then 'Apple' is less-than 'apple'. Furthermore, 'Banana' is less than 'apple', because 'B' is 66 while 'a' is 97.

A number of methods are available to help manage string comparisons. The list below describes the most commonly used methods; a full list is available at docs.python.org.

- Methods to check a string value that returns a True or False Boolean value:
  - ***isalnum()*** -- Returns True if all characters in the string are lowercase or uppercase letters, or the numbers 0-9.
  - ***isdigit()*** -- Returns True if all characters are the numbers 0-9.
  - ***islower()*** -- Returns True if all cased characters are lowercase letters.
  - ***isupper()*** -- Return True if all cased characters are uppercase letters.
  - ***isspace()*** -- Return True if all characters are whitespace.
  - ***startswith(x)*** -- Return True if the string starts with x.
  - ***endswith(x)*** -- Return True if the string ends with x.

Note that the methods `islower()` and `isupper()` ignore non-cased characters. Ex: `'abc?'.islower()` returns True, ignoring the question mark.

PARTICIPATION ACTIVITY 25.3.3: String methods: Boolean string comparisons.

Determine whether the given expression evaluates to True or False.

1) `'HTTPS://google.com'.isalnum()`
   - True
   - False

2) `'HTTPS://google.com'.startswith('HTTP')`
   - True
   - False

3) `'\n \n'.isspace()`
   - True
   - False

4) `'1 2 3 4 5'.isdigit()`
   - True
   - False

5) `'LINCOLN, ABRAHAM'.isupper()`

True

Creating new strings from a string

A programmer often needs to transform two strings into similar formats to perform a comparison. The list below shows some of the more common string methods that create string copies, altering the case or amount of whitespace of the original string:

- Methods to create new strings:
  - ***capitalize()*** -- Returns a copy of the string with the first character capitalized and the rest lowercased.
  - ***lower()*** -- Returns a copy of the string with all characters lowercased.
  - ***upper()*** -- Returns a copy of the string with all characters uppercased.
  - ***strip()*** -- Returns a copy of the string with leading and trailing whitespace removed.
  - ***title()*** -- Returns a copy of the string as a title, with first letters of words capitalized.

A user may enter any one of the non-equivalent values 'Bob' , 'BOB ', or 'bob' into a program that reads in names. The statement `name = input().strip().lower()` reads in the user input, strips the leading and trailing whitespace, and changes all the characters to lowercase. Thus, user input of 'Bob', 'BOB ', or 'bob' would each result in name having just the value 'bob'.

Good practice when reading user-entered strings is to apply transformations when reading in data (such as input), as opposed to later in the program. Applying transformations immediately limits the likelihood of introducing bugs because the user entered an unexpected string value. Of course, there are many examples of programs in which capitalization or whitespace should indicate a unique string -- the programmer should use discretion depending on the program being implemented.

©zyBooks 12/15/22 00:47 1361995
John Farrell
COLOSTATECS220SeaboltFall2022

zyDE 25.3.2: String methods example: Passenger database.

The example program below shows how the above methods might be used to store passenger names and travel destinations in a database. The use of `strip()`, `lower()` `upper()` standardize user-input for easy comparison.

Run the program below and add some passengers into the database. Add a duplicate passenger name, using different capitalization, and print the list again.

Load default ter

```
menu_prompt = ('Available commands:\n'
               '  (add) Add passenger\n'
               '  (del) Delete passenger\n'
               '  (print) Print passenger list\n'
               '  (exit) Exit the program\n'
               'Enter command:\n')

destinations = ['PHX', 'AUS', 'LAS']

destination_prompt = ('Available destinations:\n'
                      '(PHX) Phoenix\n'
                      '(AUS) Austin\n'
                      '(LAS) Las Vegas\n'
                      'Enter destination:\n')

passengers = {}

```

```
add
Dusty Baker
PHX
```

Run

CHALLENGE ACTIVITY 25.3.1: Find abbreviation.

Complete the if-else statement to print 'LOL means laughing out loud' if user_tweet contains 'LOL'.

Sample output with input: 'I was LOL during the whole movie!'

```
LOL means laughing out loud.
```

422102.2723990.qx3zqy7

```
user_tweet = input()

''' Your solution goes here '''

    print('LOL means laughing out loud.')
else:
    print('No abbreviation.')
```


Run

CHALLENGE ACTIVITY 25.3.2: Replace abbreviation.

Assign decoded_tweet with user_tweet, replacing any occurrence of 'TTYL' with 'talk to you later'.

Sample output with input: 'Gotta go. I will TTYL.'

```
Gotta go. I will talk to you later.
```

422102.2723990.qx3zqy7

```
user_tweet = input()

decoded_tweet = ''' Your solution goes here '''
print(decoded_tweet)
```

Run

25.4 Splitting and joining strings

The split() method

A common programming task is to break a large string down into the comprising substrings. The string method ***split()*** splits a string into a list of tokens. Each ***token*** is a substring that forms a part of a larger string. A ***separator*** is a character or sequence of characters that indicates where to split the string into tokens.

Ex: `'Martin Luther King Jr.'.split()` splits the string literal "Martin Luther King Jr." using any whitespace character as the default separator and returns the list of tokens ['Martin', 'Luther', 'King', 'Jr.'].

The separator can be changed by calling split() with a string argument. Ex: `'a#b#c'.split('#')` uses the "#" separator to split the string "a#b#c" into the three tokens ['a', 'b', 'c'].

PARTICIPATION ACTIVITY 25.4.1: Splitting a string into tokens.

Animation content:

`undefined`

Animation captions:

1. Original string contains a pathname to an mp3 of your favorite song.
2. The pathname is split using the delimiter ' / '.
3. The variable my_tokens is assigned with the 3 tokens as a list of strings.
4. When split() is called with no argument, the delimiter defaults to a space character.

Figure 25.4.1: String split example.

```
url = input('Enter URL:\n')

tokens = url.split('/')  # Uses
'/' separator
print(tokens)
```

```
Enter URL: http://en.wikipedia.org
/wiki/Lucille_ball
['http:', '', 'en.wikipedia.org', 'wiki',
'Lucille_ball']
...
Enter URL: en.wikipedia.org/wiki/ethernet/
['en.wikipedia.org', 'wiki', 'ethernet',
'']
```


The example above shows how split() might be used to find the elements of a path to a web page; the separator used is the forward slash character '/'. The split() method creates a new list, ordered from left to right, containing a new string for each sequence of characters located between '/' separators. Thus the URL http://en.wikipedia.org/wiki/Lucille_ball is split into ['http:', '', 'en.wikipedia.org', 'wiki', 'Lucille_ball']. The separator character is not included in the resulting strings.

If the split string starts or ends with the separator, or if two consecutive separators exist, then the resulting list will contain an empty string for each such occurrence. Ex: The consecutive forward slashes of 'http://' and the ending forward slash of '.../wiki/ethernet/' generate empty strings. If the separator argument is omitted from split(), thus splitting the string wherever whitespace occurs, then no empty strings are generated.

zyDE 25.4.1: More string splitting.

Run the following program and observe the output. Edit the program by changing the method separator to "//" and " " and observe the output.

Load default template... Run


```
1 file = 'C:/Users/Charles Xavier//Documents/'
2
3 separator = '/'
4 results = file.split(separator)
5 print(f'Separator ({separator}):', results)
6
```

PARTICIPATION ACTIVITY 25.4.2: String split() method.

Use the variable song to answer the questions below.

```
song = "I scream; you scream; we all scream, for ice cream.\n"
```

1) What is the result of `song.split()`?

- `['I scream; you scream; we all scream, for ice cream.\n']`
- `['I scream;', 'you scream;', 'we all scream,', 'for ice cream.\n']`
- `['I', 'scream;', 'you', 'scream;', 'we', 'all', 'scream,' 'for', 'ice', 'cream.']`

2) What is the result of

`song.split('\n')`?

- `['I scream; you scream; we all scream, for ice cream.', '']`
- `['I scream; you scream;\n', 'we all scream,\n', 'for ice cream.\n']`
- `['I scream; you scream; we all scream, for ice cream']`

3) What is the result of `song.split('scream')`?

- `['I ', '; you ', '; we all ', ', for ice cream.\n']`
- `['I scream; you scream; we all scream, for ice cream.\n']`
- `['I', 'you', 'we all', 'for ice cream.\n']`

The join() method

The ***join()*** string method performs the inverse operation of split() by joining a list of strings together to create a single string. Ex: `my_str = '@'.join(['billgates', 'microsoft'])` assigns my_str with the string 'billgates@microsoft'. The separator '@' provides a join() method that accepts a single list argument. Each element in the list, from left to right, is concatenated to create a new string object with the separator placed between each list element. The separator can be any string, including multiple characters or an empty string.

PARTICIPATION ACTIVITY 25.4.3: String join() method.

Animation content:

`undefined`

Animation captions:

1. web_path is a list of strings that form the path of the webpage.
2. Create a string with the separator "/".

3. Then join() concatenates the list of strings with the separator "/".

A useful application of the join() method is to build a new string without separators. The empty string ('') is a perfectly valid string object, just with a length of 0. So the statement `''.join(['http://', 'www.', 'ebay', '.com'])` produces the string 'http://www.ebay.com'.

Figure 25.4.2: String join() example: Comparing join vs. loops.

The following programs are equivalent, but join() is a simpler approach that uses less code and is easier to read.

```
phrases = ['To be, ', 'or not to be.\n', 'That is
the question.']

sentence = ''
for phrase in phrases:
    sentence += phrase
print(sentence)
```

```
To be, or not to
be.
That is the
question.
```

```
phrases = ['To be, ', 'or not to be.\n', 'That is
the question.']

sentence = ''.join(phrases)
print(sentence)
```

```
To be, or not to
be.
That is the
question.
```

PARTICIPATION ACTIVITY 25.4.4: String join() method.

1) Write a statement that uses the join() method to set my_str to 'images.google.com', using the list x = ['images', 'google', 'com']

```
my_str =
```

Check Show answer

2) Write a statement that uses the join() method to set my_str to 'NewYork', using the list x =

['New', 'York']

my_str =

Check Show answer

Using the split() and join() methods together

The split() and join() methods are commonly used together to replace or remove specific sections of a string. Ex: A programmer may want to change 'C:/Users/Brian/report.txt' to 'C:\\Users\\Brian \\report.txt', perhaps because a different operating system uses different separators to specify file locations. The example below illustrates how split() and join() are used together.

Figure 25.4.3: Splitting and joining: Replacing separators.

```
path = input('Enter file name: ')

new_separator = input('Enter new separator:
')
tokens = path.split('/')
print(new_separator.join(tokens))
```

```
Enter file name: C:/Users/Wolfman/Documents
/report.pdf
Enter new separator: \\
C:\\Users\\Wolfman\\Documents\\report.pdf
```

A programmer may also want to add, remove, or replace specific token(s) from a string. Ex: The program below reads in a URL and checks whether the fourth token (index 3) is 'wiki', as Wikipedia URLs follow the format of http://language.wikipedia.org/wiki/topic. If 'wiki' is missing from the URL, the program uses the list method insert() (explained further elsewhere) to correct the URL by adding 'wiki' before index 3.

Figure 25.4.4: Splitting and joining: Editing tokens.

```
url = input('Enter Wikipedia URL: ')

tokens = url.split('/')

if 'wiki' != tokens[3]:
    tokens.insert(3, 'wiki')
    new_url = '/'.join(tokens)

    print(f'{url} is not a valid address.')
    print(f'Redirecting to {new_url}')
else:
    print(f'Loading {url}')
```

```
Enter Wikipedia URL: http://en.wikipedia.org
/wiki/Rome
Loading http://en.wikipedia.org/wiki/Rome
...
Enter Wikipedia URL: http://en.wikipedia.org/Rome
http://en.wikipedia.org/Rome is not a valid
address.
Redirecting to http://en.wikipedia.org/wiki/Rome
```


PARTICIPATION ACTIVITY 25.4.5: Splitting and joining strings.

1) Write a statement that replaces the separators in the string variable title from hyphens (-) to colons (:)

```
title = 'Python-Lab-
Warmup'
tokens = title.split('-')
title =
```

Check Show answer

CHALLENGE ACTIVITY 25.4.1: String split and join.

422102.2723990.qx3zqy7

Start

Type the program's output

```
item_info = 'Mug 15 20'

item_tokens = item_info.split()
item = item_tokens[0]
quantity = item_tokens[1]
price = item_tokens[2]

print(item, 'stock:', quantity)
print('Price:', price)
```

```
Mug stock: 15
Price: 20
```


1 2

Check Next

CHALLENGE ACTIVITY 25.4.2: Extract area code.

Assign number_segments with phone_number split by the hyphens.

Sample output with input: '977-555-3221'

```
Area code: 977
```

422102.2723990.qx3zqy7

```
1 phone_number = input()
2 number_segments = ''' Your solution goes here '''
3 area_code = number_segments[0]
4 print('Area code:', area_code)
```

Run

25.5 LAB: Checker for integer string

Forms often allow a user to enter an integer. Write a program that takes in a string representing an integer as input, and outputs `Yes` if every character is a digit 0-9 or `No` otherwise.

Ex: If the input is:

```
1995
```

the output is:

```
Yes
```

Ex: If the input is:

```
42,000
```

or any string with a non-integer character, the output is:

```
No
```

422102.2723990.qx3zqy7

LAB ACTIVITY | 25.5.1: LAB: Checker for integer string | 0 / 10

main.py Load default template...

```
1 user_string = input()
2
3 ''' Type your code here. '''
4
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program Input (from above) → **main.py** (Your program) → O

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin
working on this zyLab.
```

25.6 LAB: Name format

Many documents use a specific format for a person's name. Write a program whose input is:

firstName middleName lastName

and whose output is:

lastName, firstInitial.middleInitial.

Ex: If the input is:


```
Pat Silly Doe
```

the output is:

```
Doe, P.S.
```

If the input has the form:

firstName lastName

the output is:

lastName, firstInitial.

Ex: If the input is:


```
Julia Clark
```

the output is:

```
Clark, J.
```

422102.2723990.qx3zqy7

LAB ACTIVITY 25.6.1: LAB: Name format 0 / 10

main.py Load default template...

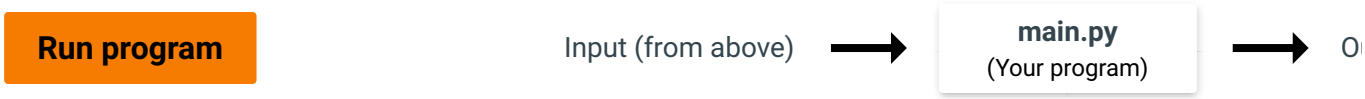
```
1 ''' Type your code here. '''
```

Develop mode | **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.



Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin
working on this zyLab.
```

25.7 LAB: Count characters

Write a program whose input is a string which contains a character and a phrase, and whose output indicates the number of times the character appears in the phrase. The output should include the input character and use the plural form, n's, if the number of times the characters appears is not exactly 1.

Ex: If the input is:

```
n Monday
```

the output is:

```
1 n
```

Ex: If the input is:

```
z Today is Monday
```

the output is:


```
0 z's
```

Ex: If the input is:

```
n It's a sunny day
```

the output is:

```
2 n's
```

Case matters. n is different than N.

Ex: If the input is:

```
n Nobody
```


the output is:

```
0 n's
```

422102.2723990.qx3zqy7

LAB ACTIVITY 25.7.1: LAB: Count characters 0 / 10

main.py Load default template...

```
1 ''' Type your code here. '''
```

Develop mode **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → main.py (Your program) → O

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin
working on this zyLab.
```

25.8 LAB: Mad Lib - loops

Mad Libs are activities that have a person provide various words, which are then used to complete a short story in unexpected (and hopefully funny) ways.

Write a program that takes a string and an integer as input, and outputs a sentence using the input values as shown in the example below. The program repeats until the input string is `quit` and disregards the integer input that follows.

Ex: If the input is:

```
apples 5
shoes 2
quit 0
```

the output is:

```
Eating 5 apples a day keeps the doctor away.
Eating 2 shoes a day keeps the doctor away.
```

422102.2723990.qx3zqy7

LAB ACTIVITY | 25.8.1: LAB: Mad Lib - loops 0 / 10

main.py Load default template...

```
1 ''' Type your code here. '''
```

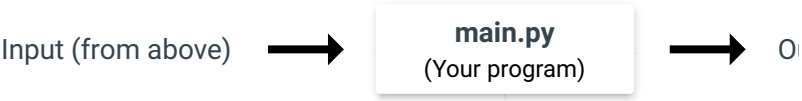
Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program



Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin
working on this zyLab.
```

©zyBooks 12/15/22 00:47 1361995
John Farrell
COLOSTATECS220SeaboltFall2022

25.9 LAB: Palindrome

A palindrome is a word or a phrase that is the same when read both forward and backward. Examples are: "bob," "sees," or "never odd or even" (ignoring spaces). Write a program whose input is a word or phrase, and that outputs whether the input is a palindrome.

Ex: If the input is:

```
bob
```

the output is:

```
bob is a palindrome
```


Ex: If the input is:

```
bobby
```

the output is:

```
bobby is not a palindrome
```

Hint: Start by removing spaces. Then check if a string is equivalent to it's reverse.

422102.2723990.qx3zqy7

LAB ACTIVITY | 25.9.1: LAB: Palindrome | 0 / 10

main.py — Load default template...

```
1 ''' Type your code here. '''
2
```

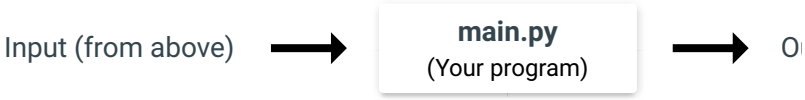

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program



Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin
working on this zyLab.
```

25.10 LAB: Acronyms

An acronym is a word formed from the initial letters of words in a set phrase. Write a program whose input is a phrase and whose output is an acronym of the input. Append a period (.) after each letter in the acronym. If a word begins with a lower case letter, don't include that letter in the acronym. Assume the input has at least one upper case letter.

Ex: If the input is:

```
Institute of Electrical and Electronics Engineers
```

the output is:

```
I.E.E.E.
```

Ex: If the input is:


```
Association for computing MACHINERY
```

the output is:

```
A.M.
```

The letters ACHINERY in MACHINERY don't start a word, so those letters are omitted.

Hint: Use isupper() to check if a letter is upper case.

422102.2723990.qx3zqy7

LAB ACTIVITY | 25.10.1: LAB: Acronyms | 0 / 10

main.py Load default template...

```
1 ''' Type your code here. '''
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program Input (from above) → **main.py** (Your program) → O

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin
working on this zyLab.
```

25.11 LAB: Contains the character

Write a program that reads a character, then reads in a list of words. The output of the program is every word in the list that contains the character at least once. Assume at least one word in the list will contain the given character.

Ex: If the input is:

```
z
hello zoo sleep drizzle
```

the output is:

```
zoo
drizzle
```

Keep in mind that the character 'a' is not equal to the character 'A'.

422102.2723990.qx3zqy7

LAB ACTIVITY 25.11.1: LAB: Contains the character 0 / 10

main.py Load default template...

```
1 ''' Type your code here. '''
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → main.py (Your program) → O

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin
working on this zyLab.
```

25.12 LAB: Warm up: Parsing strings

(1) Prompt the user for a string that contains two strings separated by a comma. (1 pt)

- Examples of strings that can be accepted:
  - Jill, Allen
  - Jill , Allen
  - Jill,Allen

Ex:

```
Enter input string:
Jill, Allen
```

(2) Report an error if the input string does not contain a comma. Continue to prompt until a valid

string is entered. *Note: If the input contains a comma, then assume that the input also contains two strings.* (2 pts)

Ex:

```
Enter input string:
Jill Allen
Error: No comma in string.

Enter input string: Jill, Allen
```


(3) Using string splitting, extract the two words from the input string and then remove any spaces. Output the two words. (2 pts)

Ex:

```
Enter input string:
Jill, Allen
First word: Jill
Second word: Allen
```

(4) Using a loop, extend the program to handle multiple lines of input. Continue until the user enters q to quit. (2 pts)

Ex:

```
Enter input string:
Jill, Allen
First word: Jill
Second word: Allen

Enter input string:
Golden , Monkey
First word: Golden
Second word: Monkey

Enter input string:
Washington,DC
First word: Washington
Second word: DC
```

```
Enter input string:
q
```

422102.2723990.qx3zqy7

LAB ACTIVITY | 25.12.1: LAB: Warm up: Parsing strings | 0 / 7

main.py Load default template...

```
1 # Type your code here
```

Develop mode | **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → **main.py** (Your program) → O

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin
working on this zyLab.
```

25.13 LAB*: Program: Data visualization

(1) Prompt the user for a title for data. Output the title. (1 pt)

Ex:

```
Enter a title for the data:
Number of Novels Authored
You entered: Number of Novels Authored
```

(2) Prompt the user for the headers of two columns of a table. Output the column headers. (1 pt)

Ex:

```
Enter the column 1 header:
Author name
You entered: Author name

Enter the column 2 header:
Number of novels
You entered: Number of novels
```

(3) Prompt the user for data points. Data points must be in this format: *string, int*. Store the information before the comma into a string variable and the information after the comma into an integer. The user will enter `-1` when they have finished entering data points. Output the data points. Store the string components of the data points in a list of strings. Store the integer components of the data points in a list of integers. (4 pts)

Ex:

```
Enter a data point (-1 to stop input):
Jane Austen, 6
Data string: Jane Austen
```

```
Data integer: 6
```

(4) Perform error checking for the data point entries. If any of the following errors occurs, output the appropriate error message and prompt again for a valid data point.

- If entry has no comma
  - Output: `Error: No comma in string.` (1 pt)
- If entry has more than one comma
  - Output: `Error: Too many commas in input.` (1 pt)
- If entry after the comma is not an integer
  - Output: `Error: Comma not followed by an integer.` (2 pts)

Ex:

```
Enter a data point (-1 to stop input):
Ernest Hemingway 9
Error: No comma in string.

Enter a data point (-1 to stop input):
Ernest, Hemingway, 9
Error: Too many commas in input.

Enter a data point (-1 to stop input):
Ernest Hemingway, nine
Error: Comma not followed by an integer.

Enter a data point (-1 to stop input):
Ernest Hemingway, 9
Data string: Ernest Hemingway
Data integer: 9
```

(5) Output the information in a formatted table. The title is right justified with a minimum field width value of 33. Column 1 has a minimum field width value of 20. Column 2 has a minimum field width value of 23. (3 pts)

Ex:

```
        Number of Novels Authored
Author name         |       Number of novels
-------------------------------------------
```

```
Jane Austen         |                  6
Charles Dickens     |                 20
Ernest Hemingway    |                  9
Jack Kerouac        |                 22
F. Scott Fitzgerald |                  8
Mary Shelley        |                  7
Charlotte Bronte    |                  5
Mark Twain          |                 11
Agatha Christie     |                 73
Ian Flemming        |                 14
Stephen King        |                 54
Oscar Wilde         |                  1
```


(6) Output the information as a formatted histogram. Each name is right justified with a minimum field width value of 20. (4 pts)

Ex:

```
         Jane Austen ******
     Charles Dickens ********************
    Ernest Hemingway *********
        Jack Kerouac ************************
 F. Scott Fitzgerald ********
        Mary Shelley *******
    Charlotte Bronte *****
          Mark Twain ***********
     Agatha Christie
************************************************************************
        Ian Flemming **************
        Stephen King
******************************************************
         Oscar Wilde *
```

422102.2723990.qx3zqy7

LAB ACTIVITY 25.13.1: LAB*: Program: Data visualization 0 / 17

main.py Load default template...

```
1 # Type your code here
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → **main.py** (Your program) → O

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin
working on this zyLab.
```


25.14 LAB: Remove all non-alpha characters

This section's content is not available for print.