CN101

Lecture 7

Strings

Topics

- Basic String Operations
- String Slicing
- Testing, Searching, and Manipulating Strings

Basic String Operations

- Many types of programs perform operations on strings
- In Python, many tools for examining and manipulating strings
  - Strings are sequences, so many of the tools that work with sequences work with strings
- Display the character by using `print()` function
- Assigning a string into a variable can be done by quotes.

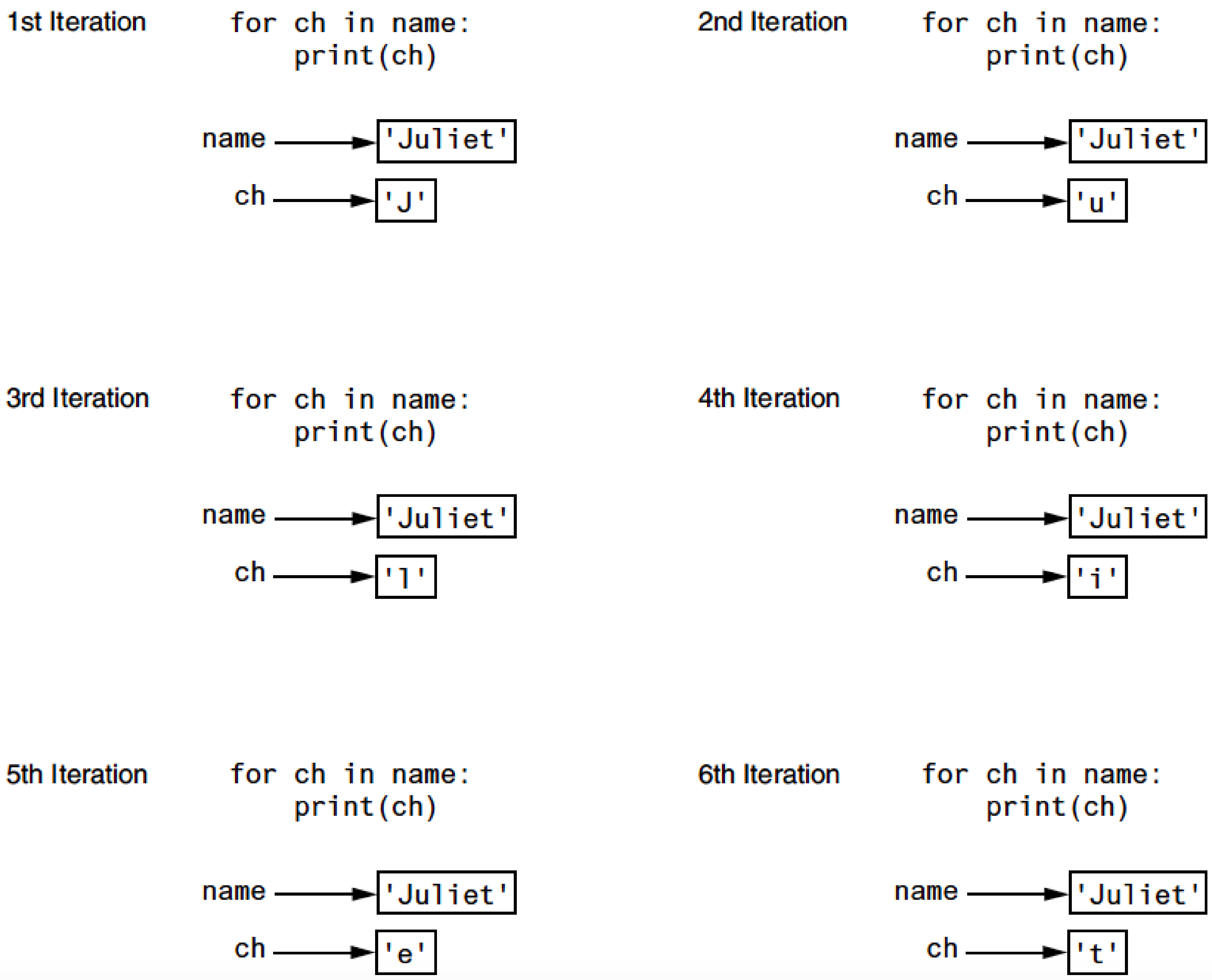
```
>>> print(“Hello”)
Hello
```

```
>>> a = “Hello”
>>> print(a)
Hello
```

Accessing the Individual Characters in a String

- To access an individual character in a string:
  - Use a `for` loop
    - Format: `for` *`character`* `in` *`string`*`:`
    - Useful when need to iterate over the whole string, such as to count the occurrences of a specific character

```
>>> name = 'Juliet'
>>> for ch in name:
...     print(ch)
...
J
u
l
i
e
t
```



1st Iteration

```
for ch in name:
    print(ch)
```

2nd Iteration

```
for ch in name:
    print(ch)
```

3rd Iteration

```
for ch in name:
    print(ch)
```

4th Iteration

```
for ch in name:
    print(ch)
```

5th Iteration

```
for ch in name:
    print(ch)
```

6th Iteration

```
for ch in name:
    print(ch)
```

count_Ts.py

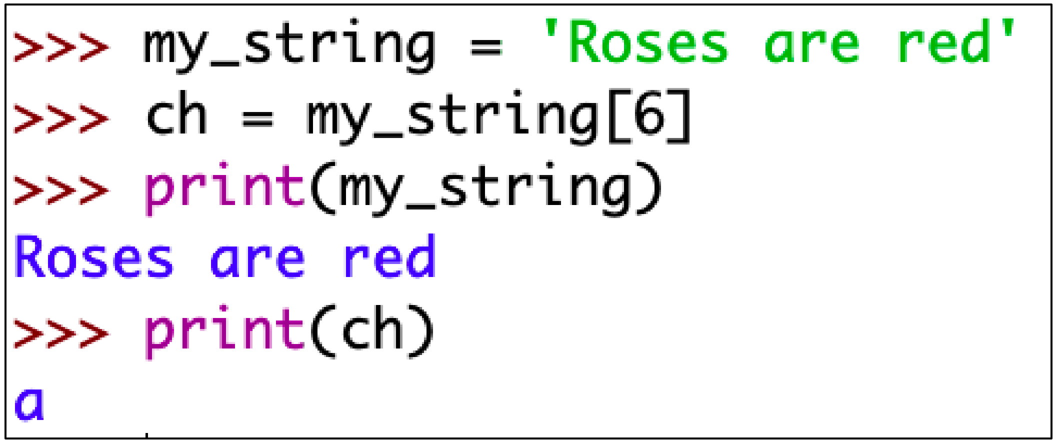
```python
def main():
    # Initialize an accumulator variable
    count = 0

    # Get a string from the user.
    my_string = input('Enter a sentence: ')

    # Count the Ts.
    for ch in my_string:
        if ch == 'T' or ch == 't’:
            count += 1

    # Print the result.
    print('The letter T appears', count, 'times.')

main()
```

Program Output

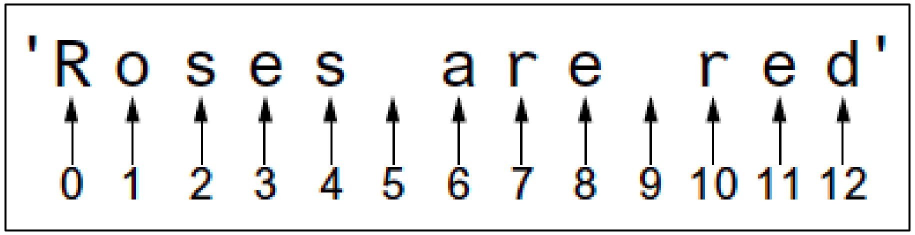
```
Enter a sentence: The tiger took the time to think
The letter T appears 7 times.
```

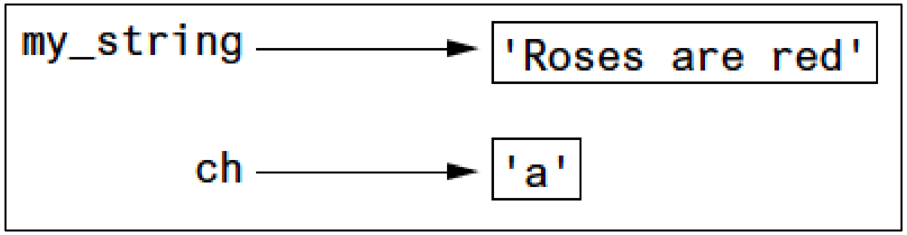
Accessing the Individual Characters in a String

- To access an individual character in a string:
  - Use indexing
    - Each character has an index specifying its position in the string, starting at 0
    - Format: `character = my_string[i]`



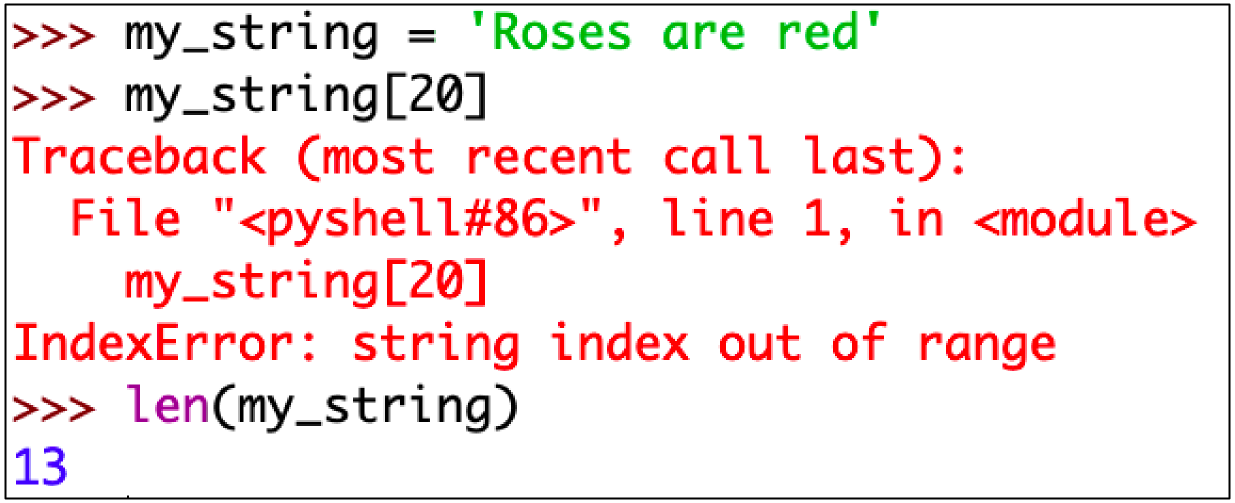
```
>>> my_string = 'Roses are red'
>>> ch = my_string[6]
>>> print(my_string)
Roses are red
>>> print(ch)
a
```





Accessing the Individual Characters in a String (cont'd.)

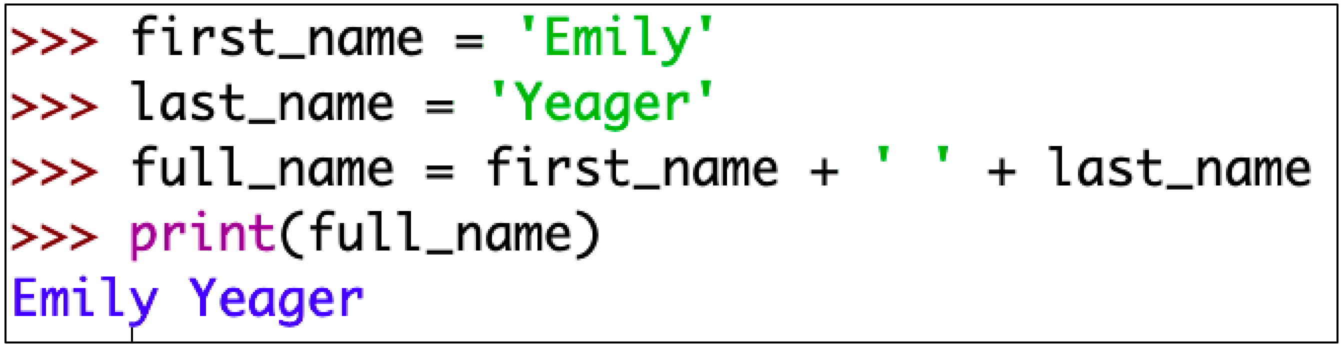
- `IndexError` exception will occur if:
  - You try to use an index that is out of range for the string
    - Likely to happen when loop iterates beyond the end of the string
- `len(`*`string`*`)` function can be used to obtain the length of a string
  - Useful to prevent loops from iterating beyond the end of a string



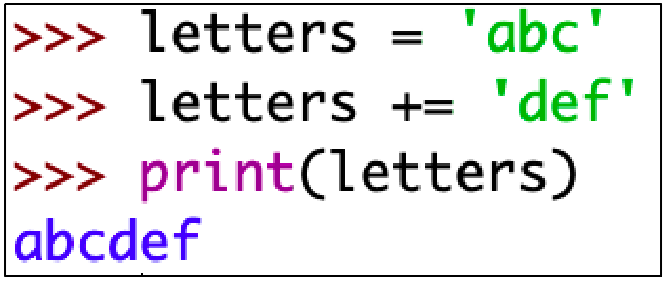
```
>>> my_string = 'Roses are red'
>>> my_string[20]
Traceback (most recent call last):
  File "<pyshell#86>", line 1, in <module>
    my_string[20]
IndexError: string index out of range
>>> len(my_string)
13
```

String Concatenation

- Concatenation: appending one string to the end of another string
  - Use the + operator to produce a string that is a combination of its operands
  - The augmented assignment operator += can also be used to concatenate strings
    - The operand on the left side of the += operator must be an existing variable; otherwise, an exception is raised



```
>>> first_name = 'Emily'
>>> last_name = 'Yeager'
>>> full_name = first_name + ' ' + last_name
>>> print(full_name)
Emily Yeager
```



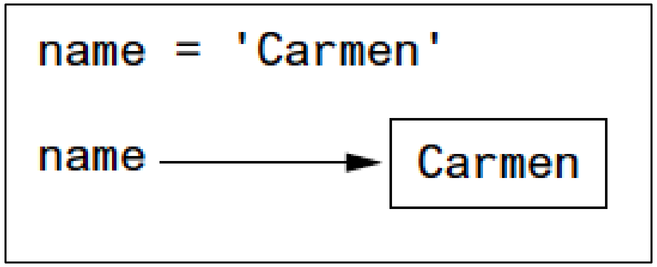
```
>>> letters = 'abc'
>>> letters += 'def'
>>> print(letters)
abcdef
```

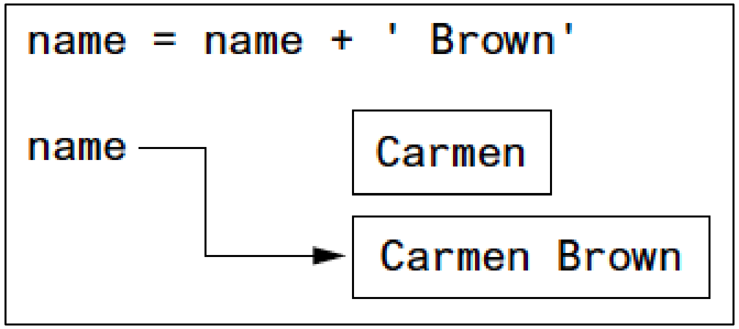
Strings Are Immutable

- Strings are immutable
  - Once they are created, they cannot be changed
    - Concatenation doesn’t actually change the existing string, but rather creates a new string and assigns the new string to the previously used variable
  - Cannot use an expression of the form
  - `string[index] = new_character`
    - Statement of this type will raise an exception

concatenate.py

```python
# This program concatenates strings.

def main():
    name = 'Carmen'
    print('The name is', name)
    name = name + ' Brown'
    print('Now the name is', name)

# Call the main function.
main()
```





Program Output

```
The name is Carmen
Now the name is Carmen Brown
```

String Slicing

- Slice: span of items taken from a sequence, known as *substring*
    - Slicing format: `string[start:end]`
        - Expression will return a string containing a copy of the characters from `start` up to, but not including, `end`
        - If `start` not specified, `0` is used for start index
        - If `end` not specified, `len(string)` is used for end index
    - Slicing expressions can include a step value and negative indexes relative to end of string

```
>>> full_name = 'Patty Lynn Smith'
>>> middle_name = full_name[6:10]
>>> middle_name
'Lynn'
>>>
>>> first_name = full_name[:5]
>>> first_name
'Patty'
>>>
>>> last_name = full_name[11:]
>>> last_name
'Smith'
>>> last_name = full_name[-5:]
>>> last_name
'Smith'
>>>
>>> my_string = full_name[:]
>>> my_string
'Patty Lynn Smith'
```

```
>>> letters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> letters[0:26:2]
'ACEGIKMOQSUWY'
>>>
>>> letters[::2]
'ACEGIKMOQSUWY'
>>>
>>> letters[::-1]
'ZYXWVUTSRQPONMLKJIHGFEDCBA'
```

Testing, Searching, and Manipulating Strings

- You can use the `in` operator to determine whether one string is contained in another string
  - General format: *`string1 in string2`*
    - *`string1`* and *`string2`* can be string literals or variables referencing strings
- Similarly you can use the `not in` operator to determine whether one string is not contained in another string

```
text = 'Four score and seven years ago'
if 'seven' in text:
    print('The string "seven" was found.')
else:
    print('The string "seven" was not found.')
```

String Methods

- Strings in Python have many types of methods, divided into different types of operations
  - General format:
    `*mystring*.*method*(*arguments*)`
- Some methods test a string for specific characteristics
  - Generally Boolean methods, that return `True` if a condition exists, and `False` otherwise

String Methods (cont'd.)

Method	Description
`isalnum()`	Returns true if the string contains only alphabetic letters or digits and is at least one character in length. Returns false otherwise.
`isalpha()`	Returns true if the string contains only alphabetic letters and is at least one character in length. Returns false otherwise.
`isdigit()`	Returns true if the string contains only numeric digits and is at least one character in length. Returns false otherwise.
`islower()`	Returns true if all of the alphabetic letters in the string are lowercase, and the string contains at least one alphabetic letter. Returns false otherwise.
`isspace()`	Returns true if the string contains only whitespace characters and is at least one character in length. Returns false otherwise. (Whitespace characters are spaces, newlines (\n), and tabs (\t)).
`isupper()`	Returns true if all of the alphabetic letters in the string are uppercase, and the string contains at least one alphabetic letter. Returns false otherwise.

string_test.py

```python
def main():
    # Get a string from the user.
    user_string = input('Enter a string: ')

    print('This is what I found about that string:’)

    # Test the string.
    if user_string.isalnum():
        print('The string is alphanumeric.')
    if user_string.isdigit():
        print('The string contains only digits.')
    if user_string.isalpha():
        print('The string contains only alphabetic characters.')
    if user_string.isspace():
        print('The string contains only whitespace characters.')
    if user_string.islower():
        print('The letters in the string are all lowercase.')
    if user_string.isupper():
        print('The letters in the string are all uppercase.')

main()
```

```
Program Output
Enter a string: abc
This is what I found about that string:
The string is alphanumeric.
The string contains only alphabetic characters.
The letters in the string are all lowercase.
```

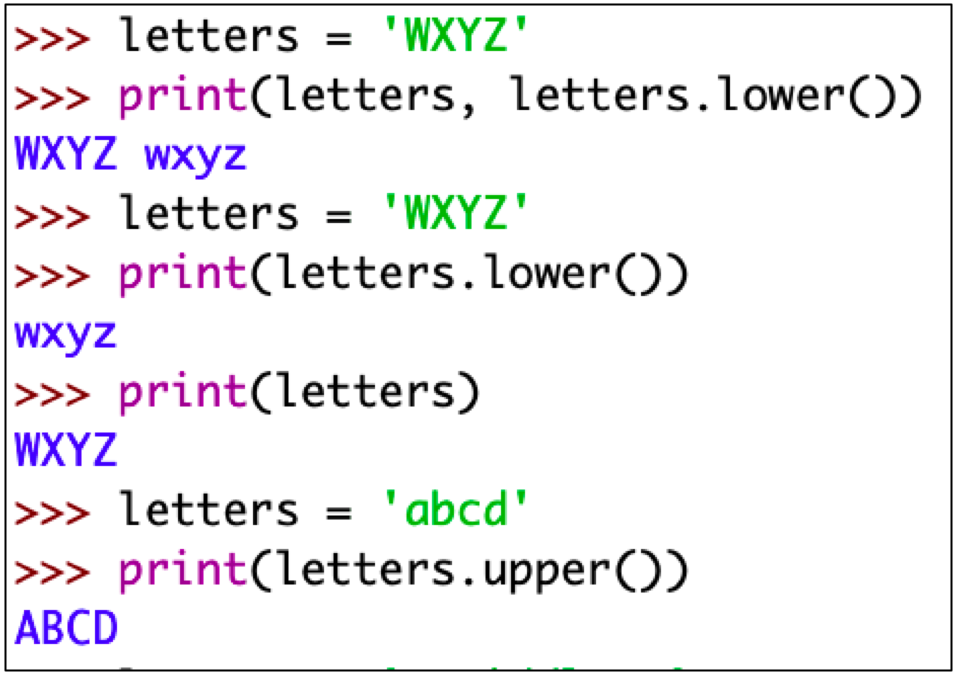
```
Program Output
Enter a string: 123
This is what I found about that string:
The string is alphanumeric.
The string contains only digits.
```

```
Program Output
Enter a string: 123ABC
This is what I found about that string:
The string is alphanumeric.
The letters in the string are all uppercase.
```

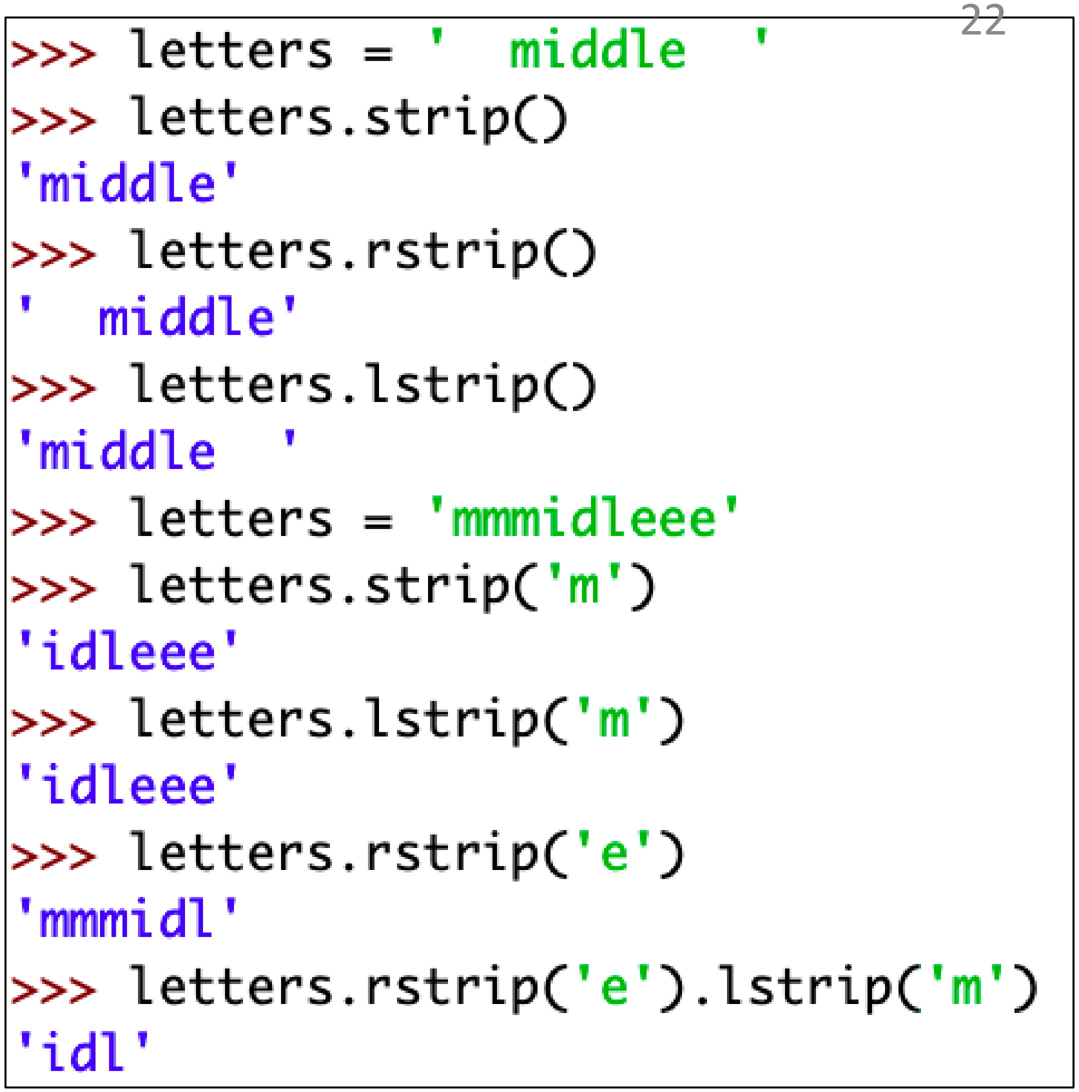
String Methods (cont'd.)

- Some methods return a copy of the string, to which modifications have been made
  - Simulate strings as mutable objects
- String comparisons are case-sensitive
  - Uppercase characters are distinguished from lowercase characters
  - `lower` and `upper` methods can be used for making case-insensitive string comparisons

Method	Description
`lower()`	Returns a copy of the string with all alphabetic letters converted to lowercase. Any character that is already lowercase, or is not an alphabetic letter, is unchanged.
`lstrip()`	Returns a copy of the string with all leading whitespace characters removed. Leading whitespace characters are spaces, newlines (\n), and tabs (\t) that appear at the beginning of the string.
`lstrip(`*`char`*`)`	The *char* argument is a string containing a character. Returns a copy of the string with all instances of *char* that appear at the beginning of the string removed.
`rstrip()`	Returns a copy of the string with all trailing whitespace characters removed. Trailing whitespace characters are spaces, newlines (\n), and tabs (\t) that appear at the end of the string.
`rstrip(`*`char`*`)`	The *char* argument is a string containing a character. The method returns a copy of the string with all instances of *char* that appear at the end of the string removed.
`strip()`	Returns a copy of the string with all leading and trailing whitespace characters removed.
`strip(`*`char`*`)`	Returns a copy of the string with all instances of *char* that appear at the beginning and the end of the string removed.
`upper()`	Returns a copy of the string with all alphabetic letters converted to uppercase. Any character that is already uppercase, or is not an alphabetic letter, is unchanged.



```
>>> letters = 'WXYZ'
>>> print(letters, letters.lower())
WXYZ wxyz
>>> letters = 'WXYZ'
>>> print(letters.lower())
wxyz
>>> print(letters)
WXYZ
>>> letters = 'abcd'
>>> print(letters.upper())
ABCD
```



```
>>> letters = '  middle  '
>>> letters.strip()
'middle'
>>> letters.rstrip()
'  middle'
>>> letters.lstrip()
'middle  '
>>> letters = 'mmmidleee'
>>> letters.strip('m')
'idleee'
>>> letters.lstrip('m')
'idleee'
>>> letters.rstrip('e')
'mmmidl'
>>> letters.rstrip('e').lstrip('m')
'idl'
```

String Methods (cont’d.)

Method	Description
`endswith(substring)`	The substring argument is a string. The method returns true if the string ends with substring.
`find(substring)`	The substring argument is a string. The method returns the lowest index in the string where substring is found. If substring is not found, the method returns -1.
`replace(old, new)`	The old and new arguments are both strings. The method returns a copy of the string with all instances of old replaced by new.
`startswith(substring)`	The substring argument is a string. The method returns true if the string starts with substring.

String Methods (cont'd.)

- Programs commonly need to search for substrings
- Several methods to accomplish this:
  - `endswith(`*`substring`*`)`: checks if the string ends with *`substring`*
    - Returns `True` or `False`
  - `startswith(`*`substring`*`)`: checks if the string starts with *`substring`*
    - Returns `True` or `False`

file_extension.py

```python
filenames = ["photo.jpg", "document.pdf", "image.png", "video.mp4", "graphic.gif"]
image_files = []

# Iterate over each file in the list of filenames
for file in filenames:
    # Check if the file ends with one of the specified image extensions
    if file.endswith(('.jpg', '.png', '.gif')):
        image_files.append(file)

print("Image files:", image_files)
```

Program Output

```
Image files: ['photo.jpg', 'image.png', 'graphic.gif']
```

secure_url.py

```python
urls = [
    "https://example.com",
    "http://example.org",
    "https://secure-site.net",
    "ftp://fileserver.com",
    "http://insecure.net"
]
secure_urls = []

# Iterate over each URL in the list
for url in urls:
    # Check if the URL starts with 'https://'
    if url.startswith("https://"):
        secure_urls.append(url)

print("Secure URLs:", secure_urls)
```

Program Output

```
Secure URLs: ['https://example.com', 'https://secure-site.net']
```

String Methods (cont'd.)

- Several methods to accomplish this (cont'd):
  - `find(`*`substring`*`)`: searches for *`substring`* within the string
    - Returns lowest index of the substring, or if the substring is not contained in the string, returns -1
  - `replace(`*`substring, new_string`*`)`:
    - Returns a copy of the string where every occurrence of *`substring`* is replaced with *`new_string`*

domain.py

```python
# input email address
email = input("Enter your email address: ")

# Use the find() method to locate the position of the '@' symbol
at_position = email.find('@')

# Check if the '@' symbol was found
if at_position != -1:
# Extract the domain part by slicing
    domain = email[at_position + 1:]
    print(f"The domain of the email address is: {domain}")
else:
    print("Invalid email address. No '@' symbol found.")
```

Program Output

```
Enter your email address: somsak@gmail.com
The domain of the email address is: gmail.com
```

The Repetition Operator

- Repetition operator: makes multiple copies of a string and joins them together
    - The * symbol is a repetition operator when applied to a string and an integer
        - String is left operand; number is right
    - General format: *`string_to_copy * n`*
    - Variable references a new string which contains multiple copies of the original string

```
>>> my_string = 'w' * 5
>>> my_string
'wwwww'
>>> print('Hello' * 5)
HelloHelloHelloHelloHello
```

censored_words.py

```python
# Original text with sensitive words
text = "This is a bad example of a text with offensive language."

# Words to be censored
censored_words = ["bad", "offensive"]

# Replace each censored word with asterisks
for word in censored_words:
    text = text.replace(word, "*" * len(word))

print("Censored text:", text)

```

Program Output

```
Censored text: This is a *** example of a text with *********
language.
```

repetition_operator.py

```python
def main():
    # Print nine rows increasing in length.
    for count in range(1, 10):
        print('Z' * count)

    # Print nine rows decreasing in length.
    for count in range(8, 0, -1):
        print('Z' * count)

    # Call the main function.
main()
```

Program Output

```
Z
ZZ
ZZZ
ZZZZ
ZZZZZ
ZZZZZZ
ZZZZZZZ
ZZZZZZZZ
ZZZZZZZZZ
ZZZZZZZZ
ZZZZZZZ
ZZZZZZ
ZZZZZ
ZZZZ
ZZZ
ZZ
Z
```

Splitting a String

- `split` method: returns a list containing the words in the string
  - By default, uses space as separator
  - Can specify a different separator by passing it as an argument to the `split` method

```
>>> date_string = "10/08/2567"
>>> date_list = date_string.split("/")
>>> date_list
['10', '08', '2567']
```

paragraphs.py

```python
# Example paragraph
paragraph = "Python is a powerful programming language. It is widely used in web development, data science, and automation. Python's simplicity makes it accessible to beginners."

# Split the paragraph into sentences using the period as the delimiter
sentences = paragraph.split(". ")

# Print each sentence
for sentence in sentences:
    print(sentence)
```

Program Output

```
Python is a powerful programming language
It is widely used in web development, data science, and automation
Python's simplicity makes it accessible to beginners.
```

csv_split.py

```python
# Example CSV line representing a record
csv_line = "John Doe,35,New York"

# Split the line into individual fields
fields = csv_line.split(",")

# Assign the fields to variables for easier access
name = fields[0]
age = fields[1]
city = fields[2]

print("Name:", name)
print("Age:", age)
print("City:", city)
```

Program Output

```
Name: John Doe
Age: 35
City: New York
```

String Join

- Join: method takes an iterable (objects capable of returning its members one at a time) as its parameter.
    - The Join method returns a string created by joining the elements of an iterable by string separator.

```
>>> list1 = ['1', '2', '3', '4']
>>> separator = ', '
>>> print(separator.join(list1))
1, 2, 3, 4
```

csv_join.py

```python
# List of values representing a data record
data = ["John Doe", "35", "New York"]

# Join the list elements into a CSV formatted string
csv_line = ",".join(data)

print("CSV Line:", csv_line)
```

Program Output

```
CSV Line: John Doe,35,New York
```

Escape Character

- To insert characters that are illegal in a string, use an escape character.
- An escape character is a backslash \ followed by the character you want to insert.

```
# This will cause an error!!!
txt = "I will get "A" from CN101"
```

```
# Using Escape Character
txt = "I will get \"A\" from CN101"
```

Escape Character (cont'd.)

Code	Result
`\'`	Single Quote
`\"`	Double Quote
`\\`	Backslash
`\n`	New Line
`\r`	Carriage Return
`\t`	Tab
`\b`	Backspace
`\ooo`	Octal value
`\xhh`	Hex value

Escape Character (cont'd.)

- Example of other escape characters used in Python:

```
>>> txt = 'It\'s a good subject.'
>>> print(txt)
It's a good subject.
```

```
>>> txt = "This will insert two \\\\ (backslash).”
>>> print(txt)
This will insert two \\ (backslash).
```

Escape Character (cont’d.)

- Example of other escape characters used in Python:

```
>>> txt = "Hello\nWorld!"
>>> print(txt)
Hello
World!
```

```
>>> txt = "Hello\tWorld!"
>>> print(txt)
Hello   World!
```

Escape Character (cont'd.)

- Example of other escape characters used in Python:

```
>>> txt = "\110\145\154\154\157"  # Octal value
>>> print(txt)
Hello
```

```
>>> txt = "\x48\x65\x6c\x6c\x6f"  # Hex value
>>> print(txt)
Hello
```

Summary

- This chapter covered:
  - String operations, including:
    - Methods for iterating over strings
    - Repetition and concatenation operators
    - Strings as immutable objects
    - Slicing strings and testing strings
    - String methods
    - Splitting a string
    - Escape character