More About Strings

陳建良

Basic String Operations

- In fact, strings are a type of sequence.
- Python provides several ways to access the individual characters in a string.
- Strings also have methods that allow you to perform operations on them.

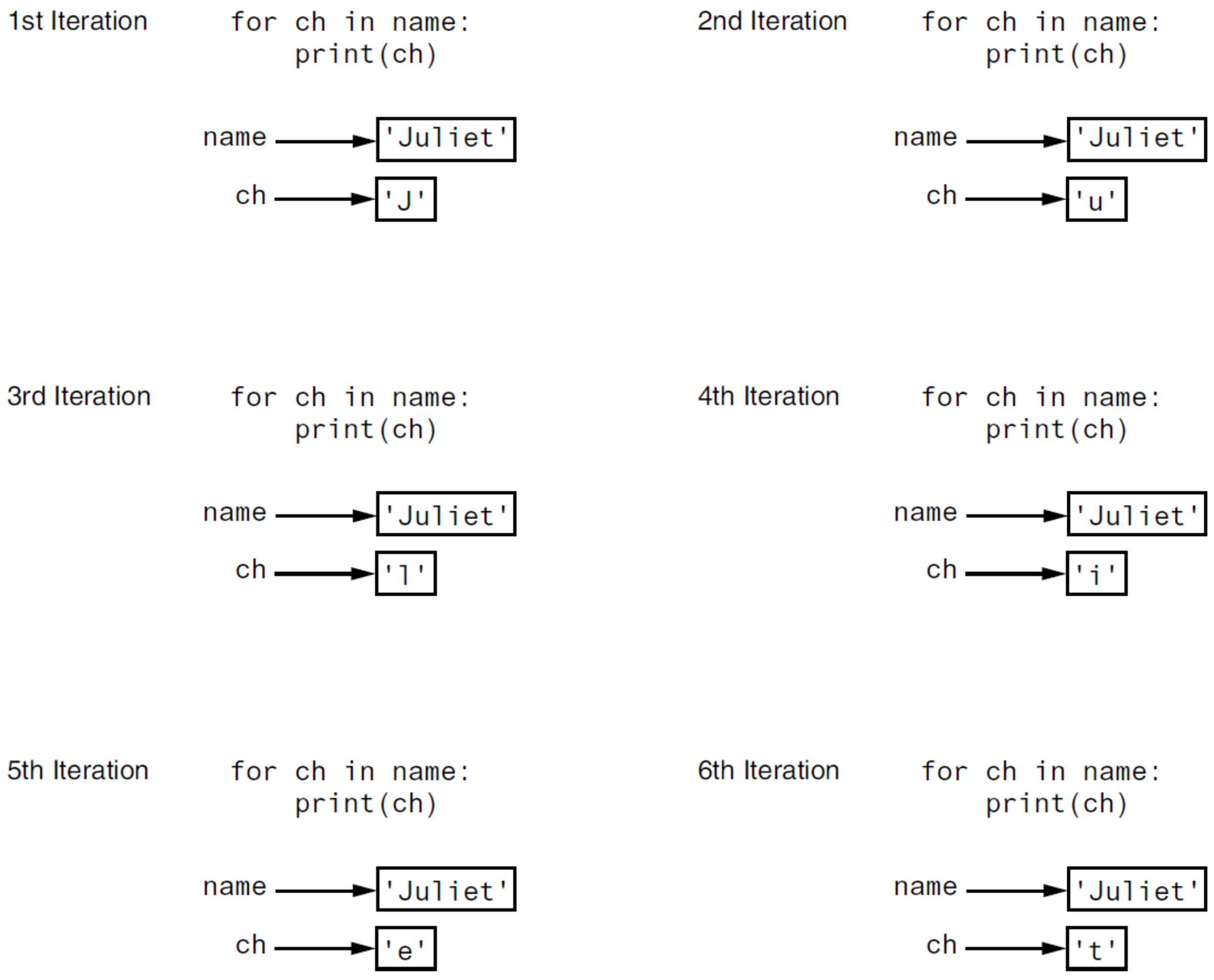
Aletheia University 資訊工程學系

Accessing the Individual Characters in a String

- Two techniques that you can use in Python to access the individual characters in a string: using the for loop, and indexing.
- To access the individual characters in a string is to use the for loop.

```
for variable in string:
    statement
    statement
    etc.
```

```
name = 'Juliet'
for ch in name:
    print(ch)
```



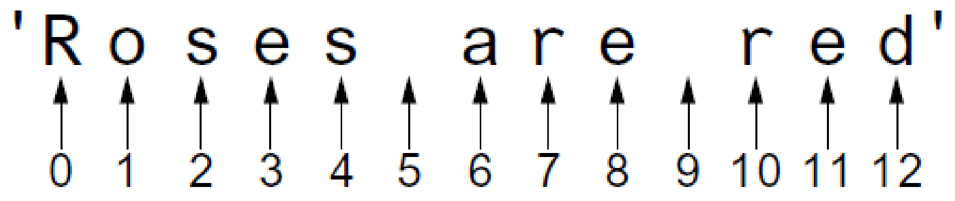
```python
# This program counts the number of times
# the letter T (uppercase or lowercase)
# appears in a string.

def main():
    # Create a variable to use to hold the count.
    # The variable must start with 0.
    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.')

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

Program Output (with input shown in bold)

```
Enter a sentence: Today we sold twenty-two toys. Enter
The letter T appears 5 times.
```

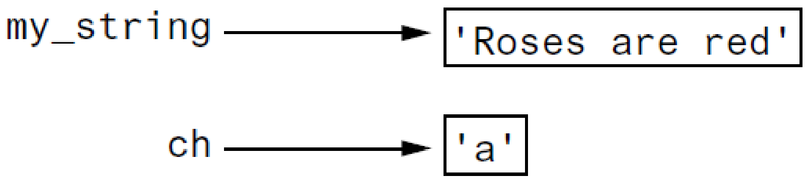
Indexing

- You can access the individual characters in a string is with an index.



my_string = 'Roses are red'

ch = my_string[6]



IndexError Exceptions

- The following is an example of code that causes an IndexError exception:

```
city = 'Boston’
print(city[6])
```

- This type of error is most likely to happen when a loop incorrectly iterates beyond the end of a string

```
city = 'Boston’
index = 0
while index < 7:
    print(city[index])
    index += 1
```

The len Function

```
city = 'Boston'
index = 0
while index < len(city):
    print(city[index])
    index += 1
```

String Concatenation

- A common operation that performed on strings is ***concatenation***, or appending one string to the end of another string.

```
>>> message = 'Hello ' + 'world' Enter
>>> print(message) Enter
Hello world
>>>
```

```
>>> name = 'Kelly' Enter          # name is 'Kelly'
>>> name += ' ' Enter             # name is 'Kelly '
>>> name += 'Yvonne' Enter        # name is 'Kelly Yvonne'
>>> name += ' ' Enter             # name is 'Kelly Yvonne '
>>> name += 'Smith' Enter         # name is 'Kelly Yvonne Smith'
>>> print(name) e
Kelly Yvonne Smith
>>>
```

Aletheia University
資訊工程學系

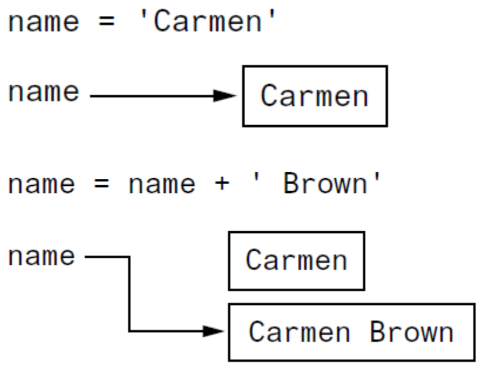
Strings Are Immutable

- In Python, strings are immutable, which means once they are created, they cannot be changed.
- Some operations, such as concatenation, give the impression that they modify strings, but in reality they do not.

```
# 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
```



- Because strings are immutable, you cannot use an expression in the form *string*[*index*] on the left side of an assignment operator.

```
# Assign 'Bill' to friend.
friend = 'Bill’
# Can we change the first character to 'J’?
friend[0] = 'J' # No, this will cause an error!
```

String Slicing

- You can use slicing expressions to select a range of characters from a string.
- String slices are also called ***substrings***.

  string[start : end]

- *start* is the index of the first character in the slice, and *end* is the index marking the end of the slice.
- The expression will return a string containing a copy of the characters from *start* up to (**but not including**) *end*.

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

- The slicing examples we have seen so far get slices of consecutive characters from strings.
- Slicing expressions can also have step value, which can cause characters to be skipped in the string.

```
letters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
print(letters[0:26:2])
```

- You can also use negative numbers as indexes in slicing expressions to reference positions relative to the end of the string.

```
full_name = 'Patty Lynn Smith'
last_name = full_name[-5:]
```

- At a university, each student is assigned a system login name, which the student uses to log into the campus computer system. As part of your internship with the university's Information Technology department, you have been asked to write the code that generates system login names for students. You will use the following algorithm to generate a login name:

1. *Get the first three characters of the student's first name. (If the first name is less than three characters in length, use the entire first name.)*
2. *Get the first three characters of the student's last name. (If the last name is less than three characters in length, use the entire last name.)*
3. *Get the last three characters of the student's ID number. (If the ID number is less than three characters in length, use the entire ID number.)*
4. *Concatenate the three sets of characters to generate the login name.*

- For example, if a student's name is Amanda Spencer, and her ID number is ENG6721, her login name would be AmaSpe721.

Program Output (with input shown in bold)

```
Enter your first name: **Holly** [Enter]
Enter your last name: **Gaddis** [Enter]
Enter your student ID number: **CSC34899** [Enter]
Your system login name is:
HolGad899
```

Program Output (with input shown in bold)

```
Enter your first name: **Jo** [Enter]
Enter your last name: **Cusimano** [Enter]
Enter your student ID number: **BIO4497** [Enter]
Your system login name is:
JoCus497
```

Aletheia University 資訊工程學系

Testing, Searching, and Manipulating Strings

- Python provides operators and methods for testing strings, searching the contents of strings, and getting modified copies of strings.
- Testing Strings with in and not in

```
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

- A method is a function that belongs to an object and performs some operation on that object.
- Strings in Python have numerous methods.
- We will discuss several string methods for performing the following types of operations:
  - Testing the values of strings
  - Performing various modifications
  - Searching for substrings and replacing sequences of characters
- The general format of a string method call:

  stringvar.method(*arguments*)

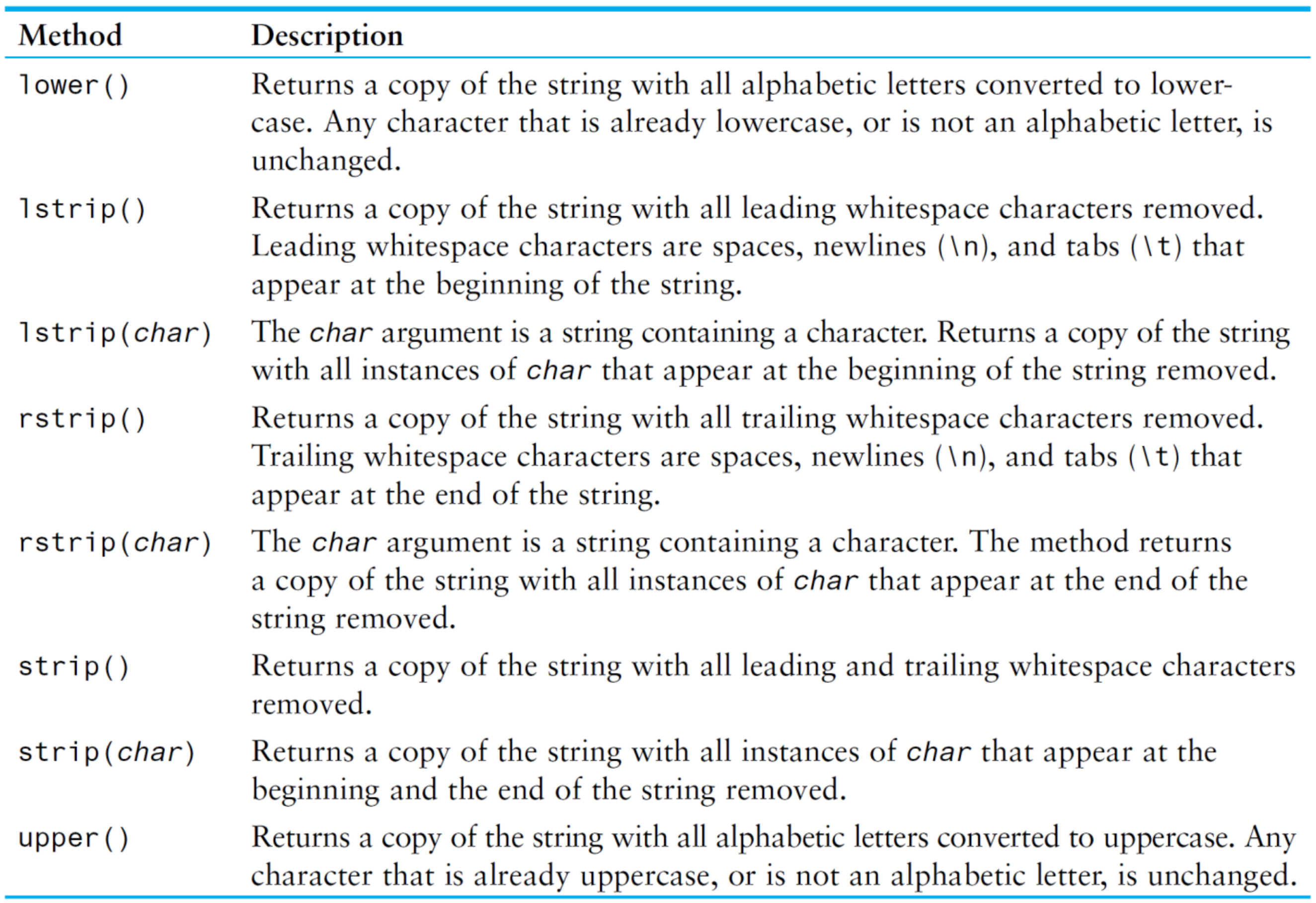
真理大學 Aletheia University 資訊工程學系

String Testing Methods

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.

```
string1 = '1200’
if string1.isdigit():
    print(string1, 'contains only digits.’)
else:
    print(string1, 'contains characters other than digits.’)

string2 = '123abc’
if string2.isdigit():
    print(string2, 'contains only digits.’)
else:
    print(string2, 'contains characters other than digits.')
```

Modification Methods



Method	Description
lower()	Returns a copy of the string with all alphabetic letters converted to lower-case. 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.

Aletheia University
資訊工程學系

```
again = 'y’
while again.lower() == 'y’:
    print('Hello’)
    print('Do you want to see that again?’)
    again = input('y = yes, anything else = no: ')
```

Searching and Replacing

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`*.

Aletheia University
資訊工程學系

```
string = 'Four score and seven years ago’
position = string.find('seven’)
if position != −1:
    print('The word "seven" was found at index', position)
else:
    print('The word "seven" was not found.')
```

- At the university, passwords for the campus computer system must meet the following requirements:
  - The password must be at least seven characters long.
  - It must contain at least one uppercase letter.
  - It must contain at least one lowercase letter.
  - It must contain at least one numeric digit.
- When a student sets up his or her password, the password must be validated to ensure it meets these requirements.
- You have been asked to write the code that performs this validation.
- You decide to write a function named valid_password that accepts the password as an argument and returns either true or false, to indicate whether it is valid.

- Here is the algorithm for the function, in pseudocode:

```
valid_password function:
   Set the correct_length variable to false
   Set the has_uppercase variable to false
   Set the has_lowercase variable to false
   Set the has_digit variable to false
   If the password's length is seven characters or greater:
      Set the correct_length variable to true
      for each character in the password:
         if the character is an uppercase letter:
            Set the has_uppercase variable to true
         if the character is a lowercase letter:
            Set the has_lowercase variable to true
         if the character isa digit:
            Set the has_digit variable to true

   If correct_length and has_uppercase and has_lowercase and has_digit:
      Set the is_valid variable to true
   else:
      Set the is_valid variable to false

   Return the is_valid variable
```

Program Output (with input shown in bold)

```
Enter your password: bozo [Enter]
That password is not valid.
Enter your password: kangaroo [Enter]
That password is not valid.
Enter your password: Tiger9 [Enter]
That password is not valid.
Enter your password: Leopard6 [Enter]
That is a valid password.
```

真理大學 Aletheia University 資訊工程學系

The Repetition Operator

- You learned how to duplicate a list with the repetition operator (*). The repetition operator works with strings as well. Here is the general format:

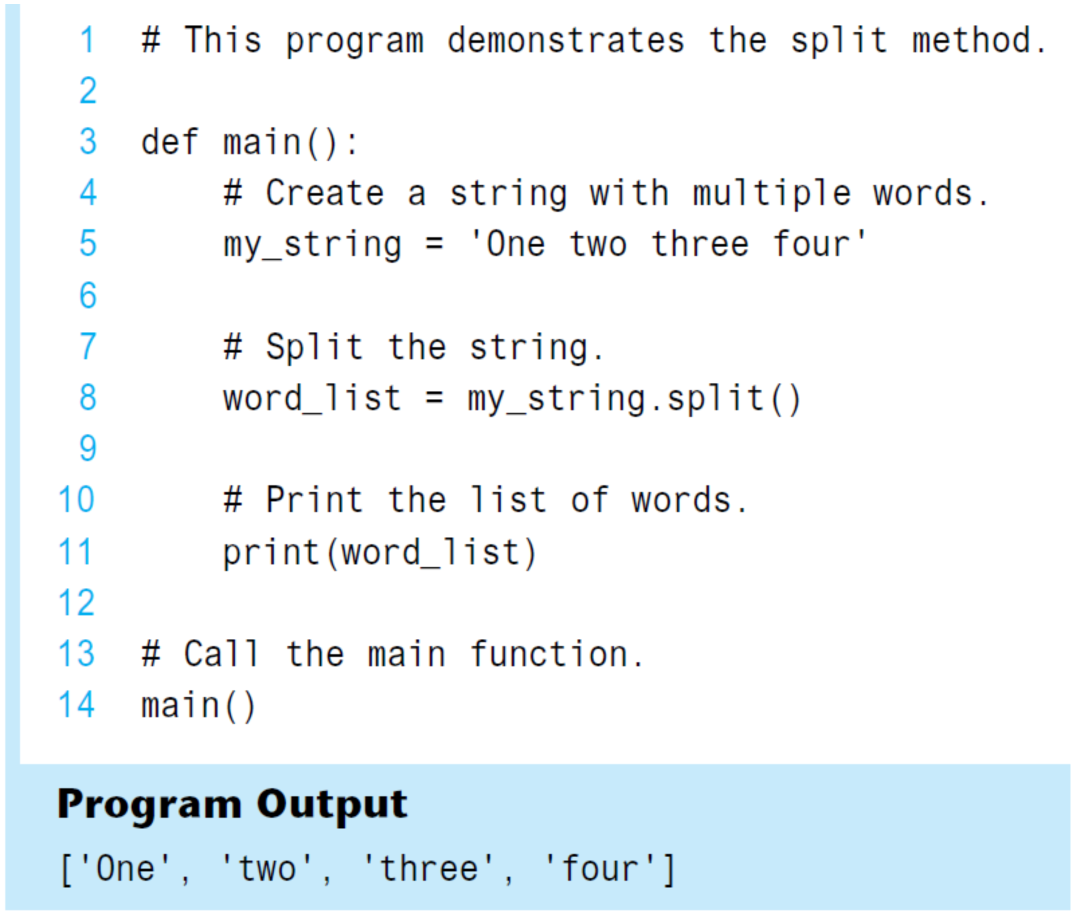
  string_to_copy * *n*

- The repetition operator creates a string that contains *n* repeated copies of *string_to_copy*.
- Example:

```
print('Hello' * 5)
```

Splitting a String

- Python have a method named **split** that returns a list containing the words in the string.



```python
# This program demonstrates the split method.

def main():
    # Create a string with multiple words.
    my_string = 'One two three four'

    # Split the string.
    word_list = my_string.split()

    # Print the list of words.
    print(word_list)

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

Program Output

```
['One', 'two', 'three', 'four']
```

```python
# This program calls the split method, using the
# '/' character as a separator.

def main():
    # Create a string with a date.
    date_string = '11/26/2018'

    # Split the date.
    date_list = date_string.split('/')

    # Display each piece of the date.
    print('Month:', date_list[0])
    print('Day:', date_list[1])
    print('Year:', date_list[2])

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

Program Output

```
Month: 11
Day: 26
Year: 2018
```