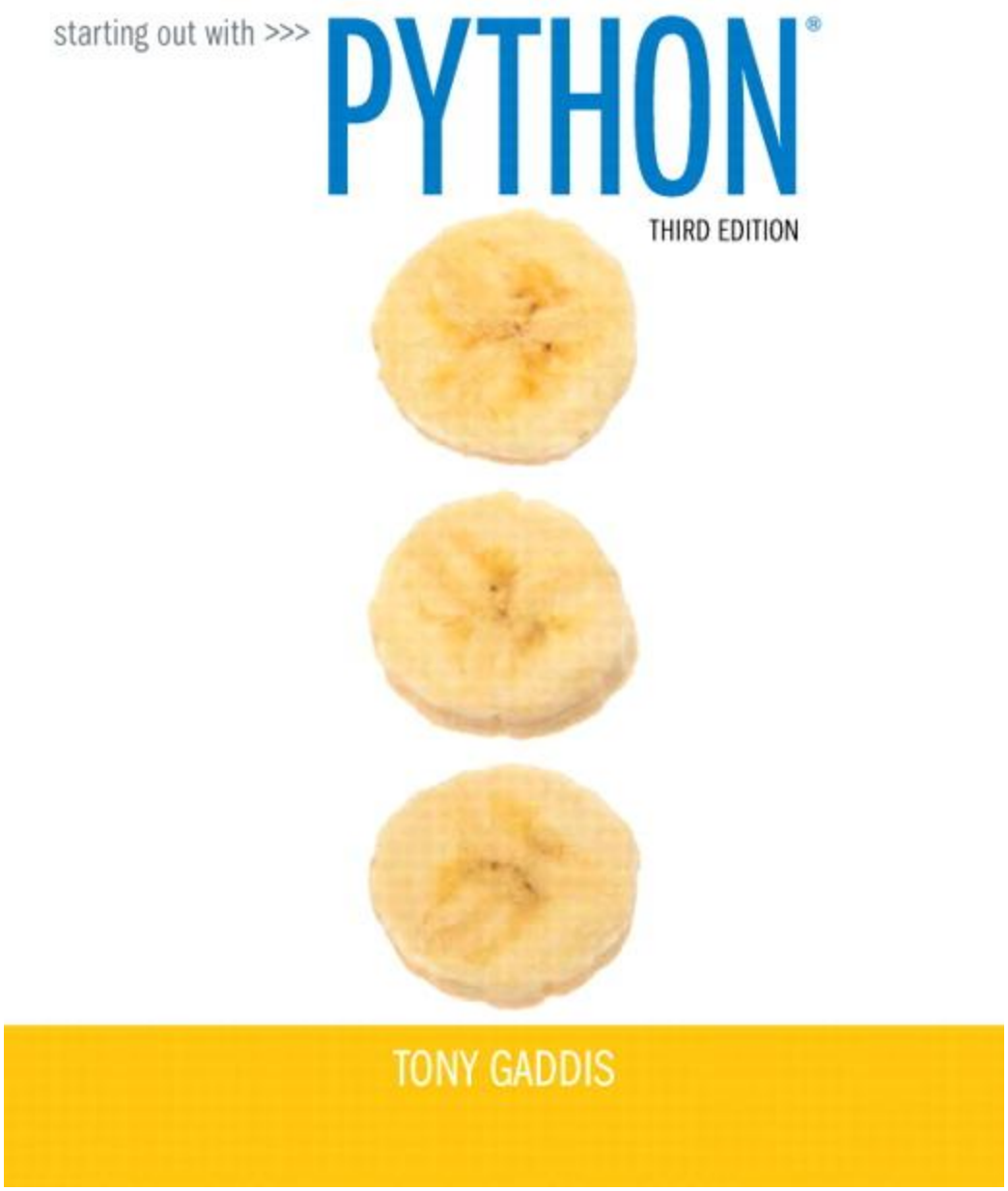
CHAPTER 8

Strings

(some parts)



Topics

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

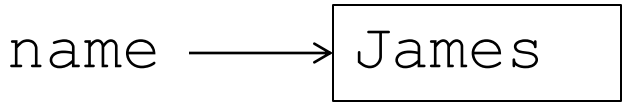
Basic String Operations

- **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

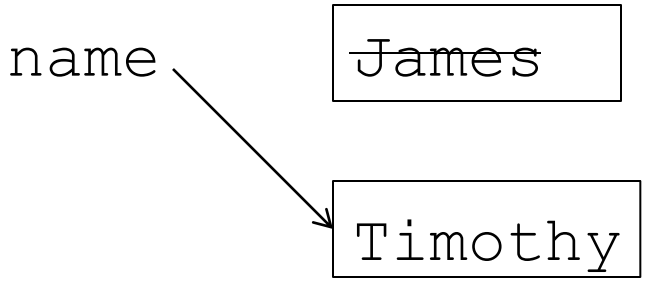
Strings Are Immutable

- **Strings are immutable**
  - Once they are created, they cannot be changed
  - Reassigning the value creates new string in memory

```
name = "James"
```




```
name = "Timothy"
```

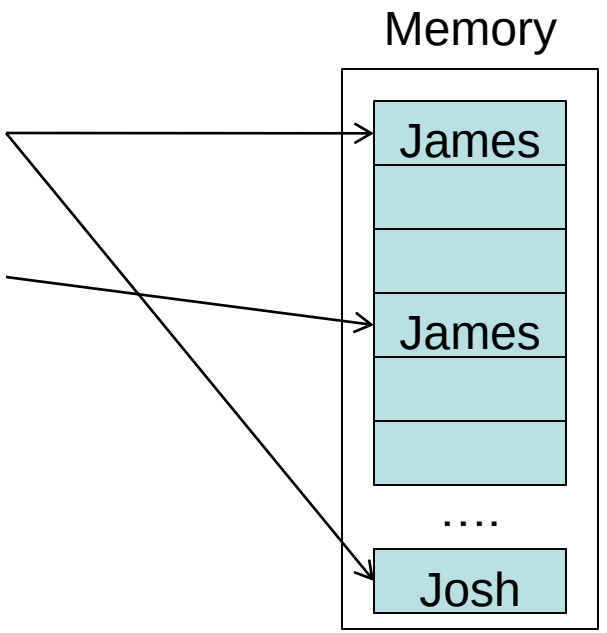


Strings Are Immutable

cName = “James”

kName = cName

cName = “Josh”



Strings Are Immutable

Figure 8-4 The string 'Carmen' assigned to `name`

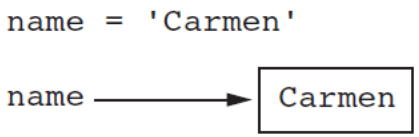
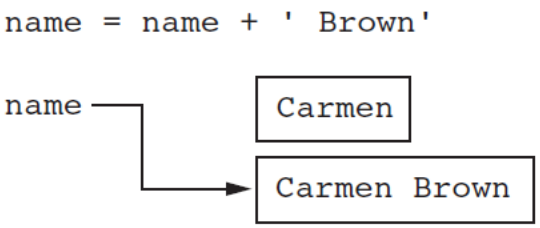
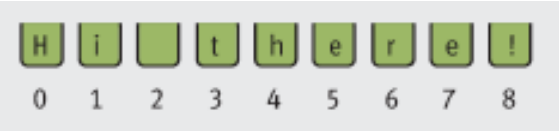


Figure 8-5 The string 'Carmen Brown' assigned to `name`



Accessing the Individual Characters in a String

- **To access an individual character in a string:**
  - Use a `for` loop (later)
    - Format: `for ch in string:`
    - Useful when need to iterate over the whole string, such as to count the occurrences of a specific character
  - Use indexing
    - Each character has an index specifying its position in the string, starting at 0
    - Format: `ch = string[i]`



Accessing the Individual Character 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

```
ch = string[13] -> Index Error
```

- **`len(`*`string`*`)` function can be used to obtain the length of a string**

```
strLen = len(string) -> 12
```

Accessing the Individual Character in a String

***word* = "nice"**

Character	n	i	c	e
Index	0	1	2	3

length = 4

sentence = "I like Python"

I		l	i	k	e		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11	12

length = 13

Accessing the Individual Character in a String

Figure 9-2 String indexes

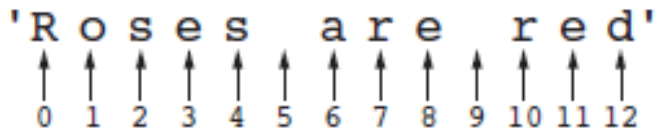
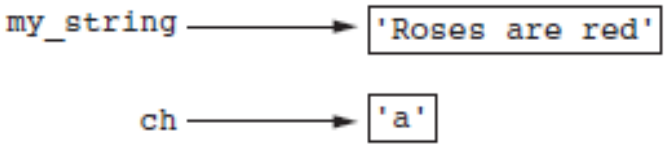


Figure 9-3 Getting a copy of a character from a string




```
ch = my_string[1]    o
ch = my_string[5]    ' '   -> empty space
ch = my_string[8]    e
ch = my_string[13]   Index Error
```

stringIndex.py

Accessing the Individual Characters in a String

- **`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

String Concatenation

- **<u>Concatenation</u>: appending one string to the end of another string**
  - Use the **+** operator to produce a string that is a combination of its operands

```
fName = "Joe"          lName = "Doe"

flName = fName + lName              -> JoeDoe
flName = fName + ' ' + lName        -> Joe Doe

fName = fName + ' ' + lName         -> Joe Doe
lName = lName + ' ' + lName         -> Doe Doe
```

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

String Slicing

myStr = "I like Python"

I		l	i	k	e		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11	12

```
newStr = myStr[0 : len(myStr)]
OUTPUT:I like Python
newStr = myStr[1 : 8]
OUTPUT: like P
newStr = myStr[2 : 9]
OUTPUT:like Py
```

stringConcatenateSliceSearch.py

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`***
- **Similarly you can use the `not in` operator to determine whether one string is not contained in another string**
  - General format: ***`string1`*** **`not in`** ***`string2`***

String Methods

- **Strings in Python have many types of methods, divided into different types of operations**
  - General format:

```
mystring.method(arguments)
```

```
name.isalpha()
```

- **Some methods test a string for specific characteristics**
  - Generally Boolean methods, that return `True` if a condition exists, and `False` otherwise

String Methods: return True or False

Table 8-1 Some 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.

String Methods

- **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

String Methods: create new string

Table 8-2 String Modification Methods

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.

String Methods

- **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`

String Methods

- **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*

String Methods

Table 8-3 Search and replace methods

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*.

stringMethods.py

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

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

Splitting a String

```
# Create a string with a date.
myStr = "this is.a/long.string"

# Split the date.
myList = myStr.split('.')

# Display each piece of the date.
print(myList[0])  -> this is
print(myList[1])  -> a/long
print(myList[2])  -> string
```

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