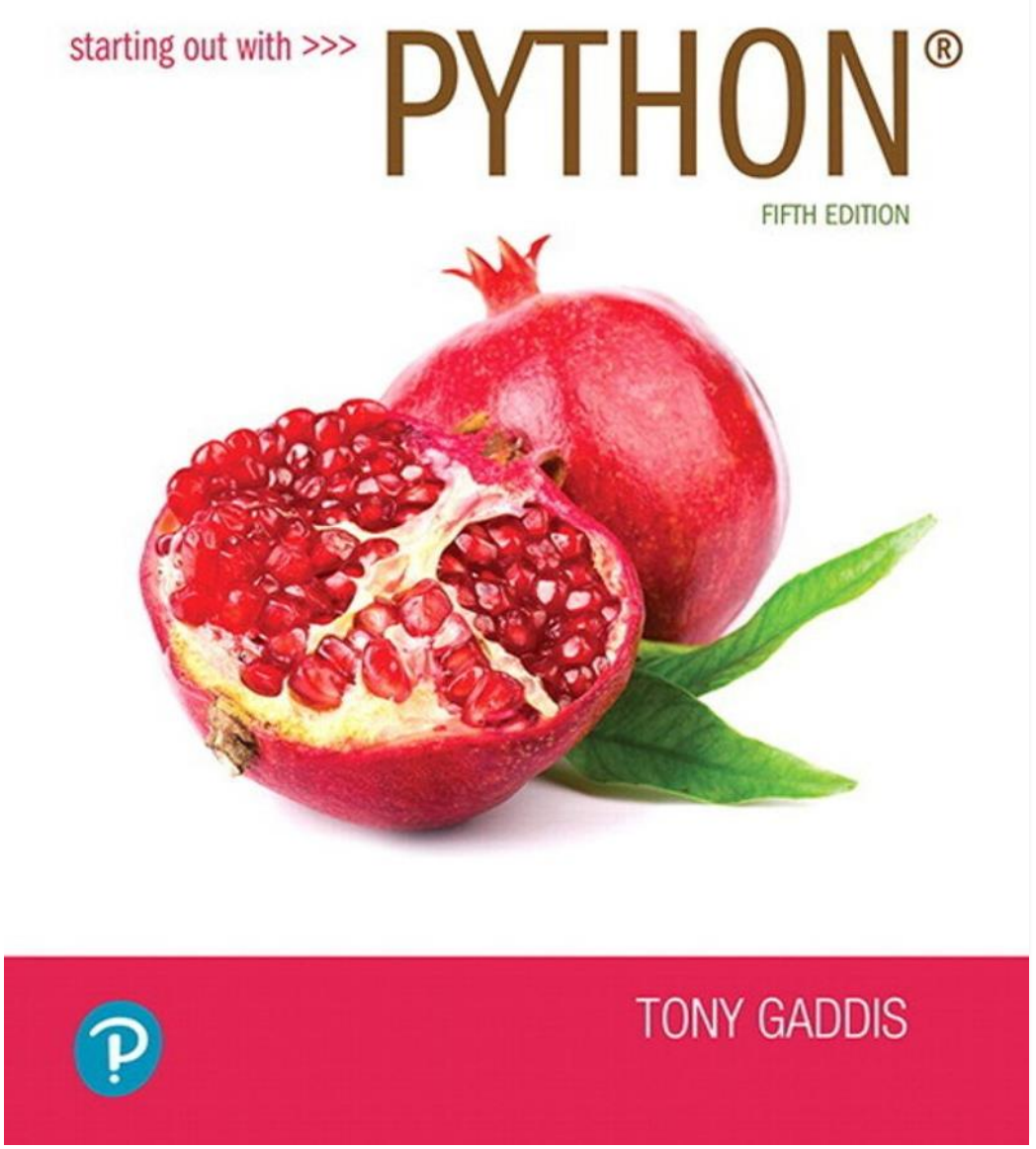
## 7.9 and Chapter 8

# Tuples and More About Strings



# Tuples

- **Tuple: an immutable sequence**
  - similar to a list, but ….
  - Once it is created it cannot be changed
  - Format: `tuple_name = (item1, item2)`
  - Notice the use of ( ) instead of [ ]
  - Tuples have operations similar to lists
    - Subscript indexing for retrieving elements
    - Methods such as `index`
    - Built in functions such as `len, min, max`
    - Slicing expressions
    - The `in`, `+`, and `*` operators

# Tuples (cont'd.)

- **Tuples do not support the methods:**
  - `append`
  - `remove`
  - `insert`
  - `reverse`
  - `sort`
  - Why not? They are immutable.

# Tuples (cont'd.)

- **Advantages for using tuples over lists:**
  - Processing tuples is faster than processing lists
  - Tuples can be safer (immutable)
  - Some operations in Python require use of tuples
- **`list()` function: converts tuple to list**
- **`tuple()` function: converts list to tuple**
- **Fun fact, a function that returns 2 or more values returns them in a tuple**

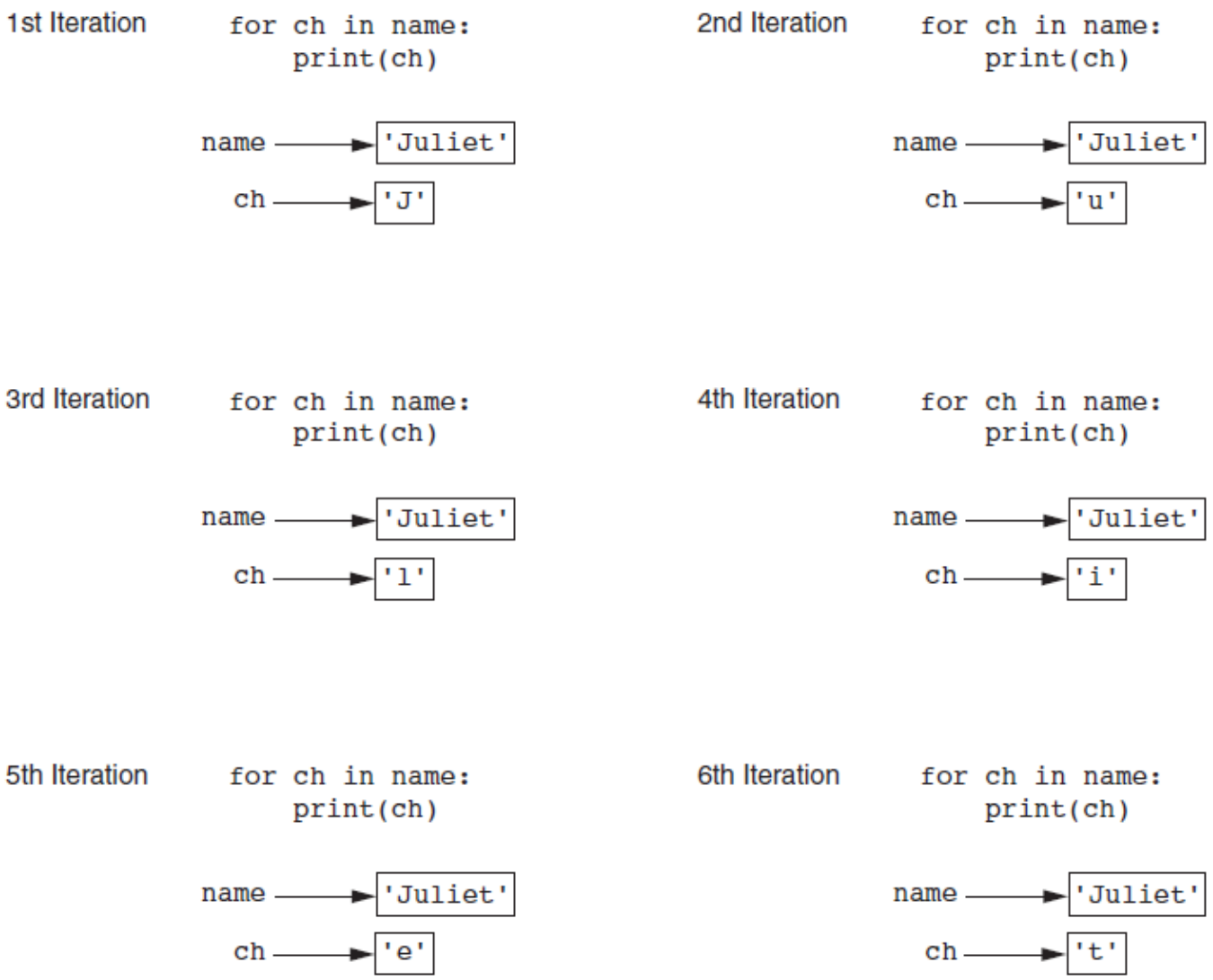
# Basic String Operations

- **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 (such as ranges, lists, and tuples) also can be used with strings

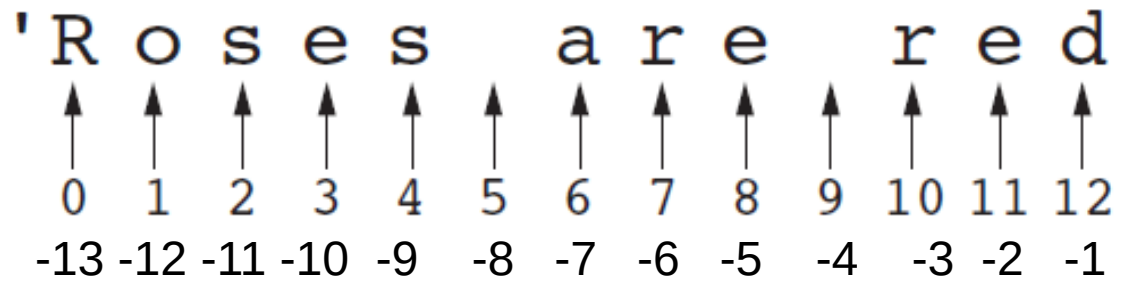
# Accessing the Individual Characters in a String

- **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
    - Each ‘character’ is simply a string of length 1
  - Use indexing
    - Each character has an index specifying its position in the string, starting at 0
    - Format: *`character = my_string[i]`*

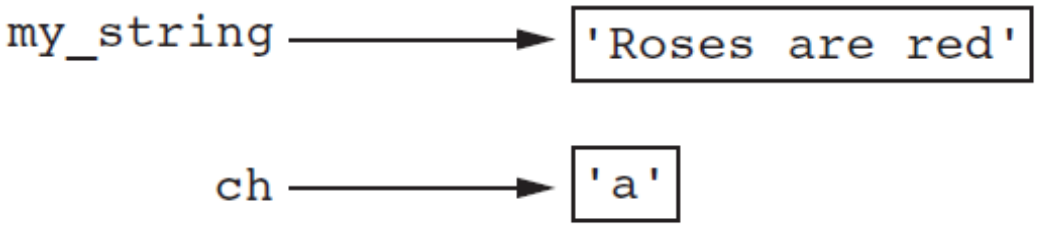
**Figure 8-1** Iterating over the string `'Juliet'`



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




## Getting a copy of a character from a string




```
ch = my_string[6]
```

# 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
- **use the `len(`*`string`*`)` function to obtain the length of a string**
  - Useful to prevent loops from iterating beyond the end of a string

# Accessing the Individual Characters in a String

- How to access the individual elements of the string using a for loop and the range function?

```
name = 'Olivia A.'
for in range(len(name)):
    print(name[i],
          type(name[i])
```

```
O <class 'str'>
l <class 'str'>
i <class 'str'>
v <class 'str'>
i <class 'str'>
a <class 'str'>
  <class 'str'>
A <class 'str'>
. <class 'str'>
```

- Or

  `for ch in string_var:`

  if we don't care about position

# 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

# 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

```
>>> name
'Olivia A.'
>>> name[7] = 'R'
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

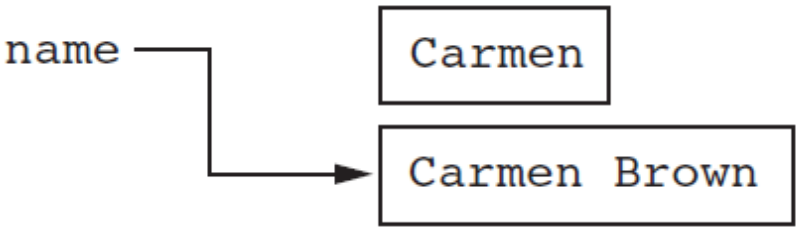
# Strings Are Immutable, Variables Are Not

## The string 'Carmen' assigned to name

```
name = 'Carmen'
```


## The string 'Carmen Brown' assigned to name

```
name = name + ' 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

# 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**

# 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.)

**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. |

Implement a function that prompts the user for an int and error checks it. Keep prompting until they enter an int

# String Methods (cont'd.)

- **Some methods create and return a modified version of the string**
  - 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 (cont'd.)

**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 (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`

# 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*

# String Methods (cont'd.)

**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*. |

# 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
  - Also referred to as *parsing* a string.

# chr and ord Functions

- Recall, the vast majority of computer systems store data in a binary form, 0's and 1's
- We have *encoding schemes* to specify what a given sequence of 0's and 1's represents, such as characters, colors, sound
- In Python, the built in chr and ord functions can be used to see the encoding for strings of length 1

```
>>> ord('A')
65
>>> ord(' ')
32
>>> ord('a')
97
>>> chr(101)
'e'
>>> chr(66)
'B'
```