COMP 141

Strings II

Rhodes College

Announcements

Reminders:

Program 6 - due Tuesday, March 27th

Practice From Last Time

- Write a loop to count the number of capital letter A's in a string.
- Write a loop to count capital or lowercase A's.
- Write a loop to print all the letters in a string in reverse order
- Write a loop to print every other character in a string, starting with the first.

String Testing Methods

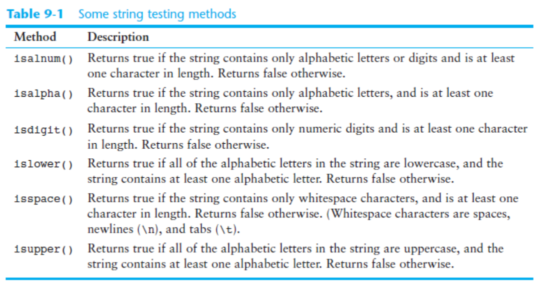


Table 9-1 Some string testing methods

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.

Example using isupper()

```
# This program counts the number of times
# the an uppercase letter 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 uppercase letters
    for ch in my_string:
        if ch.isupper():
            count += 1

    # Print the result.
    print(count, 'of the letters were uppercase.')

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

String Modification Methods

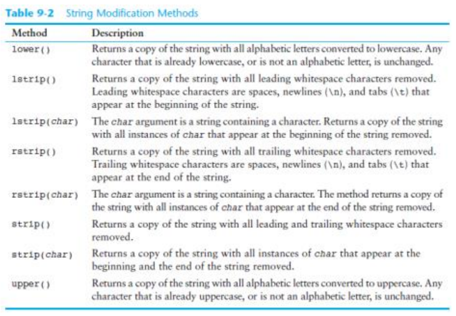


Table 9-2 String Modification Methods

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.

6

Example

```
shape = input(“Enter shape: Sphere or Cube ”)

#Ensures that all letters in shape are lowercase
shape = shape.lower()

if shape == ‘sphere’ or shape == ‘cube’:
      validShape = True
else:
      validShape = False
```

7

Using len function

# Prints 1 letter of city on each line

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

# Equivalent Code

```
city = ‘Boston’
for index in range(0, len(city)):
      print(city[index])
```

8

Accessing Characters Review

Strings are stored character by character.
Each character in a string is numbered by its position:

0	1	2	3	4	5	6	7
"C"	"o"	"m"	"p"	"u"	"t"	"e"	"r"

The numbers shown here above the characters are called ***indices*** (singular: index) or ***positions***.

Negative Indices

Negative indexing can be used.
Particularly useful for getting characters near the end of a string.

0	1	2	3	4	5	6	7
-8	-7	-6	-5	-4	-3	-2	-1
"C"	"o"	"m"	"p"	"u"	"t"	"e"	"r"

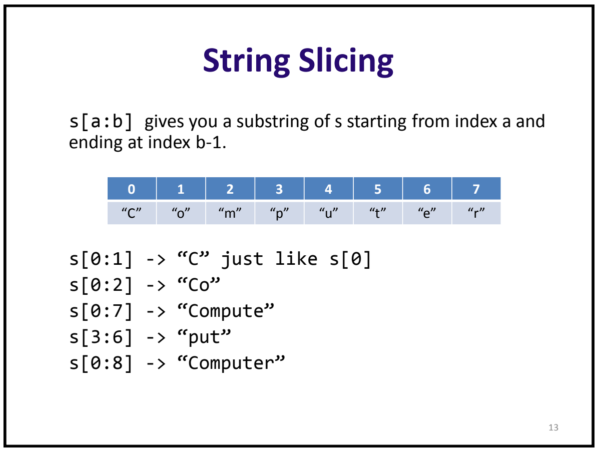
```
s[2] is the same as s[-6] both refer to "m"

To find last letter in string use:
s[-1]
```

String Indices

- Two ways to use square brackets
  - 1 number inside -> gives you 1 character of a string
    - s[0] gives you the first character in s
    - If s = "Computer", s[0] gives you 'C'
  - 2 numbers inside (separated by a colon) -> gives you a ***substring*** or string ***slice***

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



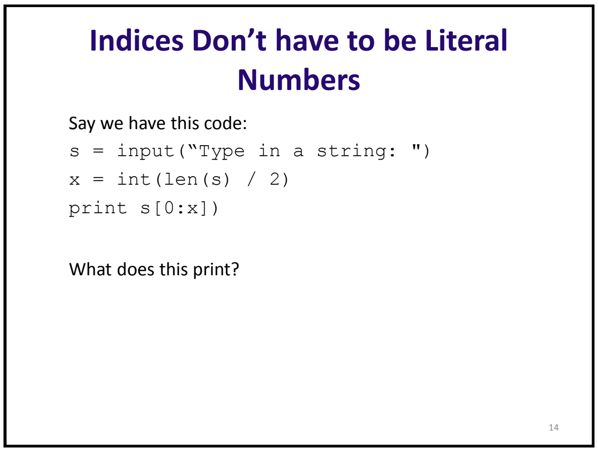
String Slicing

`s[a:b]` gives you a substring of s starting from index a and ending at index b-1.

0	1	2	3	4	5	6	7
“C”	“o”	“m”	“p”	“u”	“t”	“e”	“r”

```
s[0:1] -> “C” just like s[0]
s[0:2] -> “Co”
s[0:7] -> “Compute”
s[3:6] -> “put”
s[0:8] -> “Computer”
```

13



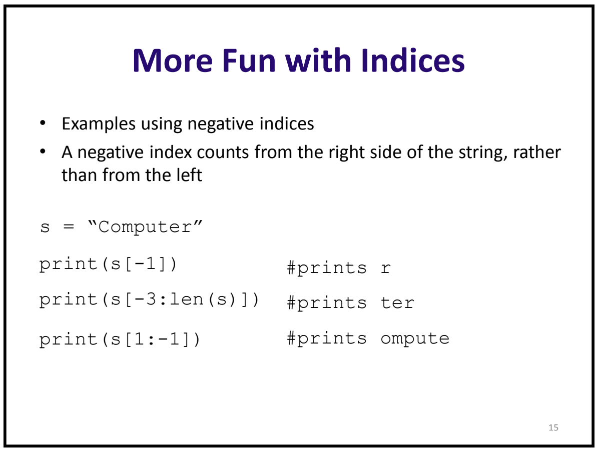
Indices Don’t have to be Literal Numbers

Say we have this code:

```
s = input(“Type in a string: ")
x = int(len(s) / 2)
print s[0:x])
```

What does this print?

14

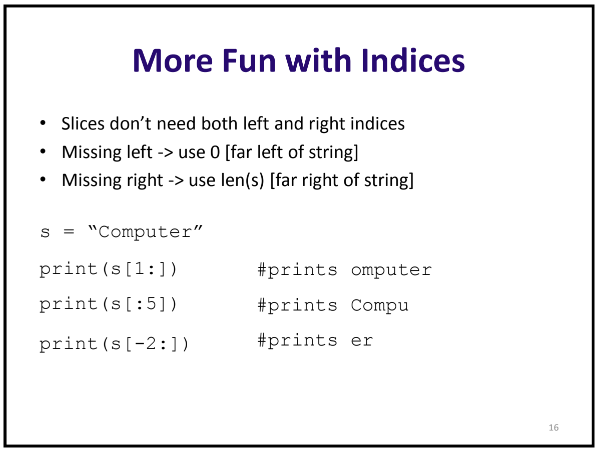


More Fun with Indices

- Examples using negative indices
- A negative index counts from the right side of the string, rather than from the left

```
s = “Computer”
print(s[-1])           #prints r
print(s[-3:len(s)])    #prints ter
print(s[1:-1])         #prints ompute
```

15

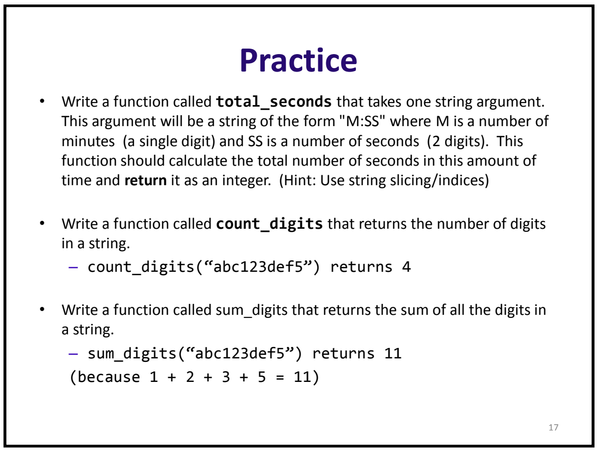


More Fun with Indices

- Slices don’t need both left and right indices
- Missing left -> use 0 [far left of string]
- Missing right -> use len(s) [far right of string]

```
s = “Computer”
print(s[1:])      #prints omputer
print(s[:5])      #prints Compu
print(s[-2:])     #prints er
```

16



Practice

- Write a function called **`total_seconds`** that takes one string argument. This argument will be a string of the form "M:SS" where M is a number of minutes (a single digit) and SS is a number of seconds (2 digits). This function should calculate the total number of seconds in this amount of time and **return** it as an integer. (Hint: Use string slicing/indices)

- Write a function called **`count_digits`** that returns the number of digits in a string.
  - `count_digits(“abc123def5”) returns 4`

- Write a function called sum_digits that returns the sum of all the digits in a string.
  - `sum_digits(“abc123def5”) returns 11`
  `(because 1 + 2 + 3 + 5 = 11)`

17