CSc 110, Autumn 2016

Lecture 11: Strings

Adapted from slides by Marty Stepp and Stuart Reges
**Strings**

- **string**: a type that stores a sequence of text characters.

  ```
  name = "text"
  name = expression
  ```

- Examples:
  ```
  name = "Daffy Duck"
  x = 3
  y = 5
  point = "(" + str(x) + ", " + str(y) + ")"
  ```
Indexes

• Characters of a string are numbered with 0-based *indexes*: 

\[
\text{name} = "\text{Ultimate}"
\]

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-8</td>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
</tr>
</tbody>
</table>

| character | U | l | t | i | m | a | t | e |

• First character's index : 0
• Last character's index : 1 less than the string's length
Accessing characters

• You can access a character with `string [index]`:
  ```python
  name = "Merlin"
  print(name[0])
  ```

  Output: M
Accessing substrings

• Syntax:
  
  part = \texttt{string[start : stop]}

• Example:

  \begin{verbatim}
  s = "Merlin"
  mid = [1:3]  # er
  \end{verbatim}

• If you want to start at the beginning you can leave off start

  \begin{verbatim}
  mid = [:3]   # Mer
  \end{verbatim}

• If you want to start at the end you can leave off the stop

  \begin{verbatim}
  mid = [1:]    # erlin
  \end{verbatim}
String methods

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>find(str)</code></td>
<td>index where the start of the given string appears in this string (-1 if not found)</td>
</tr>
<tr>
<td><code>substring(index1, index2)</code> or <code>substring(index1)</code></td>
<td>the characters in this string from <code>index1</code> (inclusive) to <code>index2</code> (exclusive); if <code>index2</code> is omitted, grabs till end of string</td>
</tr>
<tr>
<td><code>lower()</code></td>
<td>a new string with all lowercase letters</td>
</tr>
<tr>
<td><code>upper()</code></td>
<td>a new string with all uppercase letters</td>
</tr>
</tbody>
</table>

- These methods are called using the dot notation below:

```python
starz = "Biles & Manuel"
print(starz.lower())  # biles & manuel
```
String method examples

```
# index     012345678901
s1 = "Allison Obourn"
s2 = "Merlin The Cat"

print(s1.find("o"))  # 5
print(s2.lower())    # "merlin the cat"
```

• Given the following string:

```
# index     012345678901234567890123
book = "Building Python Programs"
```

• How would you extract the word "Python"?
Name border

• Prompt the user for full name

• Draw out the pattern to the left

• This should be resizable. Size 1 is shown and size 2 would have the first name twice followed by last name twice
Other String operations - length

• Syntax:

```
length = len(string)
```

• Example:

```
s = "Merlin"
count = len(s)  # 6
```
Looping through a string

• The `for` loop through a string using range:

```python
major = "CSc";
for letter in range(0, len(major)):
    print(major[letter:letter + 1])
```

• You can also use a `for` loop to print or examine each character without range.

```python
major = "CSc";
for letter in major:
    print(letter)
```

Output:
C
S
c
Strings question

• Write a program that reads two people's first names and suggests a name for their child

Example Output:
Parent 1 first name? Danielle
Parent 2 first name? John
Child Gender? f
Suggested baby name: JODANI

Parent 1 first name? Danielle
Parent 2 first name? John
Child Gender? Male
Suggested baby name: DANIJO
String tests

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>startswith()</code></td>
<td>whether one contains other's characters at start</td>
</tr>
<tr>
<td><code>endswith()</code></td>
<td>whether one contains other's characters at end</td>
</tr>
</tbody>
</table>

name = "Voldemort"

if(name.startswith("Vol")):  
    print("He who must not be named")

• The `in` keyword can be used to test if a string contains another string.

    example: "er" in name        # true
String question

• A Caesar cipher is a simple encryption where a message is encoded by shifting each letter by a given amount.
  • e.g. with a shift of 3, A → D, H → K, X → A, and Z → C

• Write a program that reads a message from the user and performs a Caesar cipher on its letters:

  Your secret message: Brad thinks Angelina is cute
  Your secret key: 3
  The encoded message: eudg wklqnv dqjholqd lv fxwh
Strings and ints

• All char values are assigned numbers internally by the computer, called ASCII values.

  • Examples:
    'A' is 65,  'B' is 66,  ' ' is 32
    'a' is 97,  'b' is 98,  '*' is 42

  • One character long Strings and ints can be converted to each other
    ord('a') is 97,  chr(103) is 'g'

  • This is useful because you can do the following:
    chr(ord('a' + 2)) is 'c'
# This program reads a message and a secret key from the user and
# encrypts the message using a Caesar cipher, shifting each letter.

def main():
    message = input("Your secret message: ")
    message = message.lower()
    key = int(input("Your secret key: "))
    encode(message, key)

# This method encodes the given text string using a Caesar
# cipher, shifting each letter by the given number of places.

def encode(text, shift):
    print("The encoded message: ")
    for letter in text:
        # shift only letters (leave other characters alone)
        if (letter >= 'a' and letter <= 'z'):
            letter = chr(ord(letter) + shift)
        # may need to wrap around
        if (letter > 'z'):
            letter = chr(ord(letter) - 26)
        elif (letter < 'a'):
            letter = chr(ord(letter) + 26)
        print(letter, end='')
    print()
format
Formatting text with `format`

```python
print("format string" . format(parameters))
```

• A format string can contain *placeholders* to insert parameters:
  • `{:d}`    integer
  • `{:f}`    real number
  • `{:s}`    string
    • these placeholders are used instead of + concatenation

• Example:
  ```python
  x = 3;
  y = -17;
  print("x is {:d} and y is {:d}!".format(x, y))
    # x is 3 and y is -17!
  ```
format width

- {:Wd} integer, W characters wide
- ...

```python
for i in range(1, 4):
    for j in range(1, 11):
        print("{:4d}".format(i * j), end='')
    print()  # to end the line
```

Output:

```
  1   2   3   4   5   6   7   8   9  10
  2   4   6   8  10  12  14  16  18  20
  3   6   9  12  15  18  21  24  27  30
```
format precision

• `{:Df}` real number, rounded to D digits after decimal
• `{:W.Df}` real number, W chars wide, D digits after decimal

gpa = 3.253764
print("your GPA is {:.1f}".format(gpa))
print("more precisely: {:.8.3f}".format(gpa))

Output:
your GPA is 3.3
more precisely: 3.254
Modify our Receipt program to better format its output.
  - Display results in the format below, with 2 digits after .

Example log of execution:

How many people ate? 4
Person #1: How much did your dinner cost? 20.00
Person #2: How much did your dinner cost? 15
Person #3: How much did your dinner cost? 25.0
Person #4: How much did your dinner cost? 10.00

Subtotal: $70.00
Tax: $5.60
Tip: $10.50
Total: $86.10
# Calculates total owed, assuming 8% tax and 15% tip

def results(subtotal):
    tax = subtotal * .08
    tip = subtotal * .15
    total = subtotal + tax + tip

    # print("Subtotal: $" + str(subtotal))
    # print("Tax: $" + str(tax))
    # print("Tip: $" + str(tip))
    # print("Total: $" + str(total))

    print("Subtotal: ${:.2f}".format(subtotal))
    print("Tax:     ${:.2f}".format(tax))
    print("Tip:    ${:.2f}".format(tip))
    print("Total:  ${:.2f}".format(total))