Strings
Quote Use

• Single Quotes

'These strings must fit on a single line of source'

• Double Quotes

"Also has to fit on a single line of source"

• Triple (single or double) Quotes

""" These quotes are very useful when you need to span multiple lines. They are also often used for long code comments """
String Concatenation

- $x = 'This will give you an error'$
  - ✗

- $x = ('This will not give you' + 'an error')$
  - ✓

- $x = 'But this will give you' + 'an error'$
  - ✗

- $x = ('Automatic concatenation lets' 'you do this')$
  - ✓
Slash \\

• We can use the \ to span multiple lines

\[
x = 'This will work correctly\ 
because we are using the slash'
\]

\[
y = 1 + 2 + 3 \ 
+ 4 + 5
\]

• Works with strings or expressions
  • No character can follow the \\

• Use parentheses for fewer errors

\[
x = ('This is clearer and 
 'safer to use.')
\]

\[
y = (1 + 2 + 3 + 
 4 + 5)
\]
Quotes inside strings

• To use apostrophes

"Let's use double quotes"

• To use double quotes in our strings

'They say, "use single quotes"

• Triple Quotes can take care of both cases

"""With 3 quotes it's "easy" to use apostrophes & quotes."""
or
"""With 3 quotes it's "easy" to use apostrophes & quotes."""
Character escaping

• Since some characters have special meanings, we have to escape them to use them in our strings

"We can \"escape\" characters like this"

'Or let\'s escape them like this'

'and this \ is how we get a backslash in our string'
Whitespace

- This is an empty string, not a character
  \[ x = \"\" \]

- This is a space
  \[ x = \ "\ " \]

- This is a tab (a single character)
  \[ x = \"\t\" \]

- This is a new line (in Unix/Mac OS X)
  \[ x = \"\n\"

- This is a new line (in Windows)
  \[ x = \"\r\n\"

- This is a new line (in old Mac <= 9)
  \[ x = \"\r\" \]
Strings are sequences

This is called slicing
Adding (+) and Repeating (*)

• We can add (concatenate) strings with +

```python
>>> x = 'This is part '
>>> y = 'of a string.'
>>> print(x + y)
This is part of a string
```

• We can also repeat them with *

```python
>>> x = 'Are we there yet?\n'
>>> print(x * 3)
Are we there yet?
Are we there yet?
Are we there yet?
```
Compare strings

• Test equality using ==

```python
secret_word = 'fish'
user_input = input('Please type the secret word:')
if user_input == secret_word:
    print("Correct. You know the secret word!")
else:
    print("No. That is not the secret word")
```

• What about <, >, <=, >=

```python
>>> 'a' < 'b'
True
>>> 'b' < 'a'
False
```
Characters are really numbers

• ASCII table

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Character</th>
<th>Decimal</th>
<th>Character</th>
<th>Decimal</th>
<th>Character</th>
<th>Decimal</th>
<th>Character</th>
<th>Decimal</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>nul</td>
<td>016</td>
<td>(dle)</td>
<td>032</td>
<td>sp</td>
<td>048</td>
<td>`</td>
<td>064</td>
<td>P</td>
</tr>
<tr>
<td>001</td>
<td>soh</td>
<td>017</td>
<td>(dc1)</td>
<td>033</td>
<td>!</td>
<td>049</td>
<td>1</td>
<td>065</td>
<td>A</td>
</tr>
<tr>
<td>002</td>
<td>stx</td>
<td>018</td>
<td>(dc2)</td>
<td>034</td>
<td>&quot;</td>
<td>050</td>
<td>2</td>
<td>066</td>
<td>B</td>
</tr>
<tr>
<td>003</td>
<td>etx</td>
<td>019</td>
<td>(dc3)</td>
<td>035</td>
<td>#</td>
<td>051</td>
<td>3</td>
<td>067</td>
<td>C</td>
</tr>
<tr>
<td>004</td>
<td>eot</td>
<td>020</td>
<td>(dc4)</td>
<td>036</td>
<td>$</td>
<td>052</td>
<td>4</td>
<td>068</td>
<td>D</td>
</tr>
<tr>
<td>005</td>
<td>enq</td>
<td>021</td>
<td>(nak)</td>
<td>037</td>
<td>%</td>
<td>053</td>
<td>5</td>
<td>069</td>
<td>E</td>
</tr>
<tr>
<td>006</td>
<td>ack</td>
<td>022</td>
<td>(syn)</td>
<td>038</td>
<td>&amp;</td>
<td>054</td>
<td>6</td>
<td>070</td>
<td>F</td>
</tr>
<tr>
<td>007</td>
<td>bel</td>
<td>023</td>
<td>(etb)</td>
<td>039</td>
<td>'</td>
<td>055</td>
<td>7</td>
<td>071</td>
<td>G</td>
</tr>
<tr>
<td>008</td>
<td>bs</td>
<td>024</td>
<td>(can)</td>
<td>040</td>
<td>(</td>
<td>056</td>
<td>8</td>
<td>072</td>
<td>H</td>
</tr>
<tr>
<td>009</td>
<td>tab</td>
<td>025</td>
<td>(em)</td>
<td>041</td>
<td>)</td>
<td>057</td>
<td>9</td>
<td>073</td>
<td>I</td>
</tr>
<tr>
<td>010</td>
<td>lf</td>
<td>026</td>
<td>(eof)</td>
<td>042</td>
<td>*</td>
<td>058</td>
<td>;</td>
<td>074</td>
<td>J</td>
</tr>
<tr>
<td>011</td>
<td>vt</td>
<td>027</td>
<td>(esc)</td>
<td>043</td>
<td>+</td>
<td>059</td>
<td>:</td>
<td>075</td>
<td>K</td>
</tr>
<tr>
<td>012</td>
<td>np</td>
<td>028</td>
<td>(fs)</td>
<td>044</td>
<td>,</td>
<td>060</td>
<td>&lt;</td>
<td>076</td>
<td>L</td>
</tr>
<tr>
<td>013</td>
<td>cr</td>
<td>029</td>
<td>(gs)</td>
<td>045</td>
<td>-</td>
<td>061</td>
<td>=</td>
<td>077</td>
<td>M</td>
</tr>
<tr>
<td>014</td>
<td>so</td>
<td>030</td>
<td>(rs)</td>
<td>046</td>
<td>.</td>
<td>062</td>
<td>&gt;</td>
<td>078</td>
<td>N</td>
</tr>
<tr>
<td>015</td>
<td>si</td>
<td>031</td>
<td>(us)</td>
<td>047</td>
<td>/</td>
<td>063</td>
<td>?</td>
<td>079</td>
<td>O</td>
</tr>
</tbody>
</table>
Character numerical values

In [91]: ord('a')
Out[91]: 97

In [92]: ord('A')
Out[92]: 65

In [93]: chr(90)
Out[93]: 'Z'

In [94]: ord('Z')
Out[94]: 90
Print the ABCs

• Using numbers...

```python
x = ord('a')
print("We present the alphabet: ")
for ind in range(26):
    print(chr(x+ind), end=" ")
```

We present the alphabet:
```
a b c d e f g h i j k l m n o p q r s t u v w x y z
```
String Comparisons

- Characters are compared by their numerical value
- shorter strings are smaller
- If first characters are equal, compare the next one

```python
>>> ord('a')
97
>>> ord('c')
99
>>> 'a' < 'c'
True
>>> 'ab' < 'a'
False
>>> 'ab' < 'ac'
True
>>> 'az' < 'ba'
True
```
String Comparisons

- These are characters, not numbers

```python
>>> '1' < '2'
True
>>> '11' < '12'
True
>>> '34' < '99'
True
>>> '9' < '10'
False

>>> 'A' < 'a'
True
>>> ord('A')
65
>>> ord('a')
97
>>> 'B' < 'a'
True
```

- Capital letters are smaller (refer to ascii table)
Testing membership

```python
alphabet = 'abcdefghijklmnopqrstuvwxyz'
vowels = 'aeiou'

for letter in alphabet:
    print('Letter ', letter, end=" ")
    if letter in vowels:
        print('is a vowel. ')
    else:
        print('is a consonant. ')
```

Letter a is a vowel.
Letter b is a consonant.
Letter c is a consonant.
Letter d is a consonant.
Letter e is a vowel.
...
import string

In [75]: import string

In [76]: print(string.ascii_lowercase)
abcdefghijklmnopqrstuvwxyz

In [77]: print(string.ascii_letters)
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ

In [78]: print(string.digits)
0123456789

In [79]: print(string.punctuation)
!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~

In [80]: ' ' in string.whitespace
Out[80]: True

Contains many useful functions
String is an Object

- Objects contain
  - Data
    - `x = 'Hello'`  # data is sequence of characters
  - Actions (methods)
    - things object can do (often on self)
Upper/Lower Case methods

- These methods are available to all string objects

- Strings are IMMUTABLE
  - this means that characters in the sequence cannot change
  - methods return a new string
  - original data is unchanged

```python
>>> x = "This is a string"
>>> x.upper()
'THIS IS A STRING'
>>> x.lower()
'this is a string'
>>> x.swapcase()
'tHIS IS A STRING'
>>> x.title()
'This Is A String'
>>> x = 'fish paste'
>>> x.capitalize()
'Fish paste'

>> print(x)
fish paste
```
What kind of character

>>> '78'.isalnum()
True
>>> '78fish'.isalnum()
True
>>> '78 fish'.isalnum()
False
>>> '78.fish'.isalnum()
False
>>> 'fish'.isalpha()
True
>>> '78'.isalpha()
False
>>> '78'.isdigit()
True
>>> '78.5'.isdigit()
False

• These methods are available to all string objects

• Tests that return boolean types:
  • isalnum() - does the string contain only letters and digits
  • isalpha() - does the string contain only letters
  • isdigit() - does the string contain only digits
Formatting strings with strip(...)
**Formatting strings**

```
In [92]: x = "Introduction"

In [93]: '|'+x.center(40)+'|
Out[93]: '|              Introduction              |

In [94]: '|'+x.center(40,'=')+'|
Out[94]: '|==============Introduction==============|

In [95]: '|'+x.ljust(40,'=')+'|
Out[95]: '|Introduction============================|

In [96]: '|'+x.rjust(40,'=')+'|
Out[96]: '|============================Introduction|'
```
import string

user_input = input("Enter a 10-digit phone number: ")
user_input = user_input.strip().lower()

error = False
number = ""

for char in user_input:
    if char not in string.digits + ' -() 	':
        error = True; break
    elif char.isdigit():
        number += char

if not error and len(number)==10:
    number = '(+number[:3]+')+number[3:6]+'-'+number[6:]
    print("The phone number is", number)
else:
    error = True

if error:
    print("Not a properly formatted 10-digit phone number.")
Example (Palindrome tester)

```python
# Palindrome tester
import string

originalString = input('Input a string: ')
modifiedStr = originalString.lower()

badChars = string.whitespace + string.punctuation

for char in modifiedStr:
    if char in badChars:  # remove bad characters
        modifiedStr = modifiedStr.replace(char,'')

print('The original string is: ' + originalString)
print('the modified string is: ' + modifiedStr)
print('the reversal is:        ' + modifiedStr[::-1])

if modifiedStr == modifiedStr[::-1]: # it is a palindrome
    print('String is a palindrome')
else:
    print('String is NOT a palindrome')
```