

# Strings

## Python Basics 6

Dr. Ryan Krauss

Grand Valley State University

# What is a string?

- ▶ not trying to insult anyone

# What is a string?

- ▶ not trying to insult anyone
- ▶ wanting to make sure we don't confuse anyone

# What is a string?

- ▶ not trying to insult anyone
- ▶ wanting to make sure we don't confuse anyone
- ▶ a string in Python is a variable made up of a list of characters

# What is a string?

- ▶ not trying to insult anyone
- ▶ wanting to make sure we don't confuse anyone
- ▶ a string in Python is a variable made up of a list of characters
- ▶ you create them by putting characters in single or double quotes:

# What is a string?

- ▶ not trying to insult anyone
- ▶ wanting to make sure we don't confuse anyone
- ▶ a string in Python is a variable made up of a list of characters
- ▶ you create them by putting characters in single or double quotes:
  - ▶ `var1 = "hello"`

# What is a string?

- ▶ not trying to insult anyone
- ▶ wanting to make sure we don't confuse anyone
- ▶ a string in Python is a variable made up of a list of characters
- ▶ you create them by putting characters in single or double quotes:
  - ▶ `var1 = "hello"`
  - ▶ `var2 = 'ryan'`

# Working with Strings

- ▶ concatenation



# Working with Strings

- ▶ concatenation
- ▶ search (`find`) and replace

# Working with Strings

- ▶ concatenation
- ▶ search (`find`) and replace
- ▶ slicing and indexing

# Working with Strings

- ▶ concatenation
- ▶ search (`find`) and replace
- ▶ slicing and indexing
- ▶ `split`

# Working with Strings

- ▶ concatenation
- ▶ search (`find`) and replace
- ▶ slicing and indexing
- ▶ `split`
- ▶ `join`

# Working with Strings

- ▶ concatenation
- ▶ search (`find`) and replace
- ▶ slicing and indexing
- ▶ `split`
- ▶ `join`
- ▶ escape characters

# Working with Strings

- ▶ concatenation
- ▶ search (`find`) and replace
- ▶ slicing and indexing
- ▶ `split`
- ▶ `join`
- ▶ escape characters
- ▶ formatting operators

# Working with Strings

- ▶ concatenation
- ▶ search (`find`) and replace
- ▶ slicing and indexing
- ▶ `split`
- ▶ `join`
- ▶ escape characters
- ▶ formatting operators
- ▶ `strip`

# Online Tutorial

This is a pretty good online tutorial:

```
http://www.tutorialspoint.com/python/  
python\_strings.htm
```



# Concatentation

- ▶ to concatenate strings means to "add" them together end to end:

# Concatenation

- ▶ to concatenate strings means to "add" them together end to end:
  - ▶ `'hello ' + 'world' = 'hello world'`

# Concatentation

- ▶ to concatenate strings means to "add" them together end to end:
  - ▶ `'hello ' + 'world' = 'hello world'`
- ▶ this can take several forms in terms of syntax:

# Concatenation

- ▶ to concatenate strings means to "add" them together end to end:
  - ▶ `'hello ' + 'world' = 'hello world'`
- ▶ this can take several forms in terms of syntax:
  - ▶ `c = 'hello ' + 'world'`

# Concatenation

- ▶ to concatenate strings means to "add" them together end to end:
  - ▶ `'hello ' + 'world' = 'hello world'`
- ▶ this can take several forms in terms of syntax:
  - ▶ `c = 'hello ' + 'world'`

# Concatentation

- ▶ to concatenate strings means to "add" them together end to end:
  - ▶ `'hello ' + 'world' = 'hello world'`
- ▶ this can take several forms in terms of syntax:
  - ▶ `c = 'hello ' + 'world'`

```
a = 'hello '
```

```
b = 'world'
```

```
c = a + b
```

# Concatentation

- ▶ to concatenate strings means to "add" them together end to end:
  - ▶ `'hello ' + 'world' = 'hello world'`
- ▶ this can take several forms in terms of syntax:
  - ▶ `c = 'hello ' + 'world'`

```
a = 'hello '  
b = 'world'  
c = a + b
```

or

```
c = 'hello '  
c += 'world'
```

# Search (find)

- ▶ `find` is a method of any string instance



# Search (find)

- ▶ `find` is a method of any string instance
  - ▶ this is object-oriented talk

# Search (find)

- ▶ `find` is a method of any string instance
  - ▶ this is object-oriented talk

## Search (find)

- ▶ `find` is a method of any string instance
  - ▶ this is object-oriented talk

```
In [1]: c = 'hello world'
```

```
In [2]: c.find('w')
```

```
Out[2]: 6
```

```
In [3]: c.find('wo')
```

```
Out[3]: 6
```

```
In [4]: c.find('wr')
```

```
Out[4]: -1
```

# Replace

```
In [1]: c = 'hello world'
```

```
In [2]: c.replace('world', 'ryan')
```

```
Out[2]: 'hello ryan'
```

```
In [3]: c
```

```
Out[3]: 'hello world'
```

- ▶ note that `replace` does not modify `c`

## Replace (cont.)

- ▶ use a new variable to capture the new string

```
In [4]: d = c.replace('world', 'ryan')
```

```
In [5]: d
```

```
Out [5]: 'hello ryan'
```

```
In [6]: c
```

```
Out [6]: 'hello world'
```

# Slicing and Indexing

- ▶ in many ways, strings behave like lists of characters:

```
In [1]: c = 'hello world'
```

```
In [2]: c.find(' ')
```

```
Out[2]: 5
```

```
In [3]: c[5]
```

```
Out[3]: ' '
```

```
In [4]: c[0:5]
```

```
Out[4]: 'hello'
```

```
In [5]: c[6:]
```

```
Out[5]: 'world'
```

# Split

- ▶ break a string at a specific character

# Split

- ▶ break a string at a specific character
- ▶ two flavors:



# Split

- ▶ break a string at a specific character
- ▶ two flavors:
  - ▶ split into a list

# Split

- ▶ break a string at a specific character
- ▶ two flavors:
  - ▶ split into a list
  - ▶ split a maximum number of times

# Split

- ▶ break a string at a specific character
- ▶ two flavors:
  - ▶ split into a list
  - ▶ split a maximum number of times

# Split

- ▶ break a string at a specific character
- ▶ two flavors:
  - ▶ split into a list
  - ▶ split a maximum number of times

```
In [1]: d = 'this is a longer string'
```

```
In [2]: d.split(' ')
```

```
Out[2]: ['this', 'is', 'a', 'longer',  
         'string']
```

```
In [3]: d.split(' ',1)
```

```
Out[3]: ['this', 'is a longer string']
```

## Split (cont.)

- ▶ capture the output as two strings:

```
In [4]: part1, part2 = d.split(' ',1)
```

```
In [5]: part1  
Out[5]: 'this'
```

```
In [6]: part2  
Out[6]: 'is a longer string'
```

# Join

- ▶ the opposite of split

# Join

- ▶ the opposite of split
- ▶ take a list and concatenate it into a string, using the joining character in between each element

# Join

- ▶ the opposite of split
- ▶ take a list and concatenate it into a string, using the joining character in between each element



# Join

- ▶ the opposite of split
- ▶ take a list and concatenate it into a string, using the joining character in between each element

```
In [1]: mylist = ['this', 'is', 'my', 'list']
```

```
In [2]: ' '.join(mylist)
```

```
Out[2]: 'this is my list'
```

```
In [3]: '-'.join(mylist)
```

```
Out[3]: 'this-is-my-list'
```

# Joining with newlines

- ▶ If you have a list of text that you want to be lines in a text file, join them with newline characters:

```
In [1]: list2 = ['line 1', 'line 2', \
                'line 3']
```

```
In [2]: str2 = '\n'.join(list2)
```

```
In [3]: print(str2)
```

```
line 1
```

```
line 2
```

```
line 3
```

# Escape Characters

- ▶ sometimes we need to insert characters in strings that are tricky to represent such as tabs or newlines

# Escape Characters

- ▶ sometimes we need to insert characters in strings that are tricky to represent such as tabs or newlines
  - ▶ or sometimes we read these characters in from text files

# Escape Characters

- ▶ sometimes we need to insert characters in strings that are tricky to represent such as tabs or newlines
  - ▶ or sometimes we read these characters in from text files
- ▶ how do we represent these characters in Python code?

# Escape Characters

- ▶ sometimes we need to insert characters in strings that are tricky to represent such as tabs or newlines
  - ▶ or sometimes we read these characters in from text files
- ▶ how do we represent these characters in Python code?
  - ▶ answer: escape characters:

# Escape Characters

- ▶ sometimes we need to insert characters in strings that are tricky to represent such as tabs or newlines
  - ▶ or sometimes we read these characters in from text files
- ▶ how do we represent these characters in Python code?
  - ▶ answer: escape characters:
    - ▶ the backslash "`\`" escapes special characters

# Escape Characters

- ▶ sometimes we need to insert characters in strings that are tricky to represent such as tabs or newlines
  - ▶ or sometimes we read these characters in from text files
- ▶ how do we represent these characters in Python code?
  - ▶ answer: escape characters:
    - ▶ the backslash "`\`" escapes special characters
    - ▶ tab is "`\t`"



# Escape Characters

- ▶ sometimes we need to insert characters in strings that are tricky to represent such as tabs or newlines
  - ▶ or sometimes we read these characters in from text files
- ▶ how do we represent these characters in Python code?
  - ▶ answer: escape characters:
    - ▶ the backslash "`\`" escapes special characters
    - ▶ tab is "`\t`"
    - ▶ newline is "`\n`"

# Escape Characters

- ▶ sometimes we need to insert characters in strings that are tricky to represent such as tabs or newlines
  - ▶ or sometimes we read these characters in from text files
- ▶ how do we represent these characters in Python code?
  - ▶ answer: escape characters:
    - ▶ the backslash "`\`" escapes special characters
    - ▶ tab is "`\t`"
    - ▶ newline is "`\n`"
    - ▶ there are many others

# String substitutions

- ▶ somewhat tricky but very powerful

# String substitutions

- ▶ somewhat tricky but very powerful
- ▶ substitute values into a string pattern in a consistent way

# String substitutions

- ▶ somewhat tricky but very powerful
- ▶ substitute values into a string pattern in a consistent way
- ▶ examples:

# String substitutions

- ▶ somewhat tricky but very powerful
- ▶ substitute values into a string pattern in a consistent way
- ▶ examples:
  - ▶ looping through data files and wanting to save plots as "fig\_1.png", "fig\_2.png", ...

# String substitutions

- ▶ somewhat tricky but very powerful
- ▶ substitute values into a string pattern in a consistent way
- ▶ examples:
  - ▶ looping through data files and wanting to save plots as "fig\_1.png", "fig\_2.png", ...
  - ▶ specifying precise formatting of number to string conversions:

# String substitutions

- ▶ somewhat tricky but very powerful
- ▶ substitute values into a string pattern in a consistent way
- ▶ examples:
  - ▶ looping through data files and wanting to save plots as "fig\_1.png", "fig\_2.png", ...
  - ▶ specifying precise formatting of number to string conversions:
    - ▶ `'%0.4f' % pi = '3.1416'`



## Substitution Example

```
for i in range(1,4):  
    filename = 'fig_%i.png' % i  
    ...  
    savefig(filename, dpi=300)
```

# Multiple Substitutions

```
a = 17
```

```
b = 3.1234567
```

```
c = 'hello'
```

```
fmt = 'test_%i_%0.3f_%s.jpg'
```

```
mystr = fmt % (a,b,c)
```

- ▶ what is `mystr`?

# Strip

- ▶ remove whitespace from beginning and end of string

# Strip

- ▶ remove whitespace from beginning and end of string
  - ▶ but not from the middle







