Revised: 7/27/2020

Text Processing

Michael C. Hackett

Assistant Professor, Computer Science

Community College of Philadelphia

Lecture Topics

- String Characters and Indexes
- String Testing Functions
- String Slicing
- Tokenization
- CSV Files

Colors/Fonts

Global Variable Names	Brown
Local Variable Names	Lt Blue
Literals	Blue
Keywords	Orange
Operators/Punctuation	Black
Functions	Purple
Parameters	Gold
Comments	Gray
Modules	Pink
Object/Class Names	Green

Source Code	**Consolas**
Output	Courier New

String Character Indexes

- Strings are a sequence type (like lists and tables) and are comprised of characters.
    - Characters can be letters, numbers, symbols and whitespace.

- Every character in a string has an index.

```
example = "Example String"
```

E	**x**	**a**	**m**	**p**	**l**	**e**		**S**	**t**	**r**	**i**	**n**	**g**
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**

String Characters

- Characters in a string can be accessed using subscript notation.

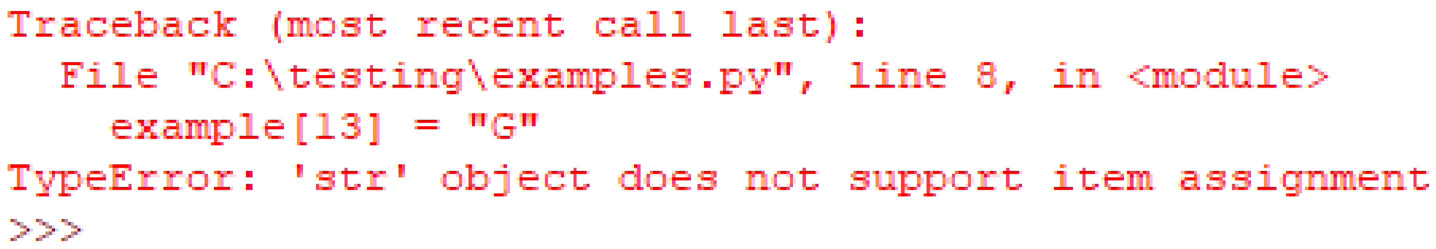
```
example = "Example String"
first_character = example[0]
print(first_character)
print(example[8])
```

```
E
S
```

String Character Indexes

- Strings, like tuples, are immutable.
  - Characters in a string cannot be changed.

```
example = "Example String"
example[13] = "G"
print(example)
```

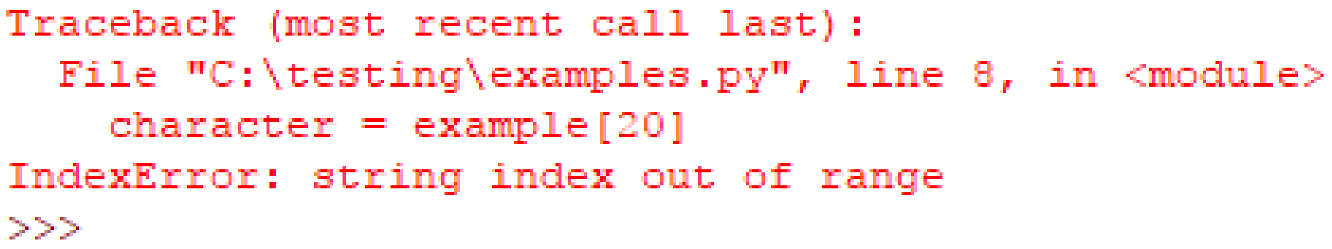


```
Traceback (most recent call last):
  File "C:\testing\examples.py", line 8, in <module>
    example[13] = "G"
TypeError: 'str' object does not support item assignment
>>>
```

String Character Indexes

- Attempting to access an index that does not exist will raise in an IndexError exception.

```
example = "Example String"
character = example[20]
print(character)
```



```
Traceback (most recent call last):
  File "C:\testing\examples.py", line 8, in <module>
    character = example[20]
IndexError: string index out of range
>>>
```

String Length

- A string’s length is the total number of characters it contains.
  - Use Python’s built-in len function to return the length of a string.

```python
example = "Example String"
length = len(example)
print(length)
```

```
14
```

Iterating Over String Characters

- A for loop can iterate over a string’s characters.
  - To loop through all or part of a string’s characters:

```
example = "Example"
for index in range(0, len(example)) :
  print(example[index])
```

  - To loop through all of a string’s characters:

```
example = "Example"
for character in example :
  print(character)
```

```
E
x
a
m
p
l
e
```

Testing if a String is Alphabetic

- The string’s isalpha function determines if a string contains only letters.

```
example1 = "Example"
example2 = "Example123"

if example1.isalpha() :
  print("example1 contains only letters")

if example2.isalpha() :
  print("example2 contains only letters")
```

```
example1 contains only letters
```

Testing if a String is Numeric

- The string’s isdigit function determines if a string contains only numbers.

```
example1 = "Example"
example2 = "Example123"

if example1.isdigit() :
  print("example1 contains only numbers")

if example2.isdigit() :
  print("example2 contains only numbers")
```

Testing if a String is Alphanumeric

- The string’s isalnum function determines if a string contains only letters and/or numbers.

```
example1 = "Example"
example2 = "Example123"

if example1.isalnum() :
  print("example1 contains only letters and/or numbers")

if example2.isalnum() :
  print("example2 contains only letters and/or numbers")
```

```
example1 contains only letters and/or numbers
example2 contains only letters and/or numbers
```

Testing if a String Contains Only Whitespace

- The string’s isspace function determines if a string contains only spaces.

```
example1 = "Example"
example2 = "   "

if example1.isspace() :
  print("example1 contains only spaces.")

if example2.isspace() :
  print("example2 contains only spaces.")
```

```
example2 contains only spaces.
```

Testing if a String is Uppercase

- The string’s isupper function determines if a string contains only uppercase letters.

```
example1 = "Example123"
example2 = "EXAMPLE123"

if example1.isupper() :
  print("example1 contains only uppercase letters.")

if example2.isupper() :
  print("example2 contains only uppercase letters.")
```

```
example2 contains only uppercase letters.
```

Testing if a String is Lowercase

- The string’s islower function determines if a string contains only lowercase letters.

```python
example1 = "Example123"
example2 = "example123"

if example1.islower() :
  print("example1 contains only lowercase letters.")

if example2.islower() :
  print("example2 contains only lowercase letters.")
```

```
example2 contains only lowercase letters.
```

Testing if a *Character* is Alphabetic

```
example = "ABC 123 ! xyz"

if example[2].isalpha() :
  print("Character at index 2 is a letter.")

if example[5].isalpha() :
  print("Character at index 5 is a letter.")

if example[7].isalpha() :
  print("Character at index 7 is a letter.")

if example[8].isalpha() :
  print("Character at index 8 is a letter.")

if example[10].isalpha() :
  print("Character at index 10 is a letter.")
```

Testing if a *Character* is Numeric

```
example = "ABC 123 ! xyz"

if example[2].isdigit() :
  print("Character at index 2 is a number.")

if example[5].isdigit() :
  print("Character at index 5 is a number.")

if example[7].isdigit() :
  print("Character at index 7 is a number.")

if example[8].isdigit() :
  print("Character at index 8 is a number.")

if example[10].isdigit() :
  print("Character at index 10 is a number.")
```

Testing if a *Character* is Alphanumeric

```
example = "ABC 123 ! xyz"

if example[2].isalnum() :
  print("Character at index 2 is a letter/number.")

if example[5].isalnum() :
  print("Character at index 5 is a letter/number.")

if example[7].isalnum() :
  print("Character at index 7 is a letter/number.")

if example[8].isalnum() :
  print("Character at index 8 is a letter/number.")

if example[10].isalnum() :
  print("Character at index 10 is a letter/number.")
```

Testing if a *Character* is Whitespace

```
example = "ABC 123 ! xyz"

if example[2].isspace() :
  print("Character at index 2 is a space.")

if example[5].isspace() :
  print("Character at index 5 is a space.")

if example[7].isspace() :
  print("Character at index 7 is a space.")

if example[8].isspace() :
  print("Character at index 8 is a space.")

if example[10].isspace() :
  print("Character at index 10 is a space.")
```

Testing if a *Character* is an Uppercase Letter

```
example = "ABC 123 ! xyz"

if example[2].isupper() :
  print("Character at index 2 is an uppercase letter.")

if example[5].isupper() :
  print("Character at index 5 is an uppercase letter.")

if example[7].isupper() :
  print("Character at index 7 is an uppercase letter.")

if example[8].isupper() :
  print("Character at index 8 is an uppercase letter.")

if example[10].isupper() :
  print("Character at index 10 is an uppercase letter.")
```

Testing if a *Character* is a Lowercase Letter

```
example = "ABC 123 ! xyz"

if example[2].islower() :
  print("Character at index 2 is a lowercase letter.")

if example[5].islower() :
  print("Character at index 5 is a lowercase letter.")

if example[7].islower() :
  print("Character at index 7 is a lowercase letter.")

if example[8].islower() :
  print("Character at index 8 is a lowercase letter.")

if example[10].islower() :
  print("Character at index 10 is a lowercase letter.")
```

Replacing parts of a String

- The string’s replace function replaces part of a string with new data.
- Two arguments (both strings)- first is the string to find, second is what to replace it with. *CASE SENSITIVE*

```python
orig_string = "Today is Monday."
new_string = orig_string.replace("Monday", "Tuesday")
print(new_string)
```

```
Today is Tuesday.
```

Note the value of orig_string **does not change**.
The replace method returns a new string with the every sequence of the first argument replaced with the second argument.

Replacing parts of a String

- The string’s replace function replaces all matches.

```
orig_string = "Today is Monday."
new_string = orig_string.replace("day", "night")
print(new_string)
```

```
Tonight is Monnight.
```

String Slicing

- String slicing selects a range of characters from a string.
  - String slices are commonly referred to as *substrings*.
- The general syntax for slicing a string is:

```
string[startIndex:endIndex]
```

- This will return a string containing all characters between those indexes.
  - The start index is inclusive.
  - The end index is exclusive.

String Slicing

```python
college = "Community College of Philadelphia"
slice = college[10:17]
print(slice)
```

```
College
```

String Slicing

- Specifying only start index will return a slice beginning with the start index’s character through the end of the string.

```python
college = "Community College of Philadelphia"
slice = college[10:]
print(slice)
```

```
College of Philadelphia
```

String Slicing

- Specifying only an ending index will return a slice beginning with the start of the string up to, but not including, the ending index.

```
college = "Community College of Philadelphia"
slice = college[:17]
print(slice)
```

```
Community College
```

String Slicing

- String slicing is safe from IndexError exceptions.
- If the starting index is greater than the ending index, an empty list will be returned.

```
college = "Community College of Philadelphia"
slice = college[13:7]
print(slice)
```

```
[ ]
```

String Slicing

- If the ending index is beyond the length of the string, Python will use the length as the ending index.

```
college = "Community College of Philadelphia"
slice = college[21:100]
print(slice)
```

```
Philadelphia
```

String Slicing

- If the starting index is negative, Python will use 0 as the starting index.

```
college = "Community College of Philadelphia"
slice = college[-5:9]
print(slice)
```

```
Community
```

- This is not the case if there is no ending index or the ending index is negative.

String Slicing

- When only a negative starting index is specified, the slice will begin relative to the end of the string.

```
college = "Community College of Philadelphia"
slice = college[-12:]
print(slice)
```

```
Philadelphia
```

String Slicing

- When both starting and ending indexes are negative, the slice will begin and end relative to the end of the string.

```python
college = "Community College of Philadelphia"
slice = college[-23:-30]
print(slice)
```

```
College
```

- If a negative starting index is greater (closer to zero) than the negative ending index, an empty string will be returned.

Tokenizing Strings

- **Tokenization** is the process of splitting up a string into smaller units.

- Strings are tokenized using a ***delimiter*** (Usually spaces or commas but can be any number of characters.)
  - For example, the string “Community College of Philadelphia” could be tokenized using whitespace as the delimiter which would break it up into 4 separate strings or ***tokens***: “Community” “College” “of” and “Philadelphia”.

Tokenizing Strings

- Strings have a split function that can tokenize a string into a list of strings.

```
string_to_tokenize = "Alabama Alaska Arkansas Arizona"

tokens = string_to_tokenize.split()
```

- By default, the split function uses whitespace as the delimiter.

Tokenizing Strings

```
string_to_tokenize = "Alabama Alaska Arkansas Arizona"
tokens = string_to_tokenize.split()

print("Total tokens=", len(tokens))

for token in tokens :
  #Print Each Token
  print("Token:", token)

print("Done")
```

```
Total tokens= 4
Token: Alabama
Token: Alaska
Token: Arkansas
Token: Arizona
Done
```

Tokenizing Strings

```python
string_to_tokenize = "Philadelphia, PA"
tokens = string_to_tokenize.split()

print("Total tokens=", len(tokens))

for token in tokens :
  #Print Each Token
  print("Token:", token)

print("Done")
```

```
Total tokens= 2
Token: Philadelphia,
Token: PA
Done
```

Tokenizing Strings

- To specify a custom delimiter, pass it as a string argument to the split function.

```
string_to_tokenize = "Alabama,Alaska,Arkansas,Arizona"

tokens = string_to_tokenize.split(",")
```

- The custom delimiter can be any number of characters long.

Tokenizing Strings

```
string_to_tokenize = "Alabama,Alaska,Arkansas,Arizona"
tokens = string_to_tokenize.split(",")

print("Total tokens=", len(tokens))

for token in tokens :
  #Print Each Token
  print("Token:", token)

print("Done")
```

```
Total tokens= 4
Token: Alabama
Token: Alaska
Token: Arkansas
Token: Arizona
Done
```

Tokenizing Strings

```
string_to_tokenize = "Philadelphia, PA"
tokens = string_to_tokenize.split(",")

print("Total tokens=", len(tokens))

for token in tokens :
  #Print Each Token
  print("Token:", token)

print("Done")
```

```
Total tokens= 2
Token: Philadelphia
Token:  PA
Done
```

Trimming Strings

- Occasionally, strings may have extra whitespace at the start or end of its character sequence.

```
string_to_tokenize = "   Alabama,Alaska,Arkansas,Arizona   "
tokens = string_to_tokenize.split(",")

for token in tokens :
  #Print Each Token
  print("Token: ", token, ".", sep="")

print("Done")
```

```
Token:    Alabama.
Token: Alaska.
Token: Arkansas.
Token: Arizona   .
Done
```

Trimming Strings

- The string’s lstrip (left strip) function removes leading whitespace.

```
string_to_tokenize = "   Alabama,Alaska,Arkansas,Arizona   "
string_to_tokenize = string_to_tokenize.lstrip()
tokens = string_to_tokenize.split(",")

for token in tokens :
  #Print Each Token
  print("Token: ", token, ".", sep="")

print("Done")
```

```
Token: Alabama.
Token: Alaska.
Token: Arkansas.
Token: Arizona   .
Done
```

Trimming Strings

- The string’s rstrip (right strip) function removes trailing whitespace.

```
string_to_tokenize = "   Alabama,Alaska,Arkansas,Arizona   "
string_to_tokenize = string_to_tokenize.lstrip()
string_to_tokenize = string_to_tokenize.rstrip()
tokens = string_to_tokenize.split(",")

for token in tokens :
  #Print Each Token
  print("Token: ", token, ".", sep="")

print("Done")
```

```
Token: Alabama.
Token: Alaska.
Token: Arkansas.
Token: Arizona.
Done
```

Trimming Strings

```
string_to_tokenize = "   Alabama,Alaska,Arkansas,Arizona   "
string_to_tokenize = string_to_tokenize.lstrip().rstrip()
tokens = string_to_tokenize.split(",")

for token in tokens :
  #Print Each Token
  print("Token: ", token, ".", sep="")

print("Done")
```

```
Token: Alabama.
Token: Alaska.
Token: Arkansas.
Token: Arizona.
Done
```

Trimming Strings

- Sometimes, extra whitespace may be captured as part of a token.
- Trimming the token removes that extra leading/trailing whitespace.

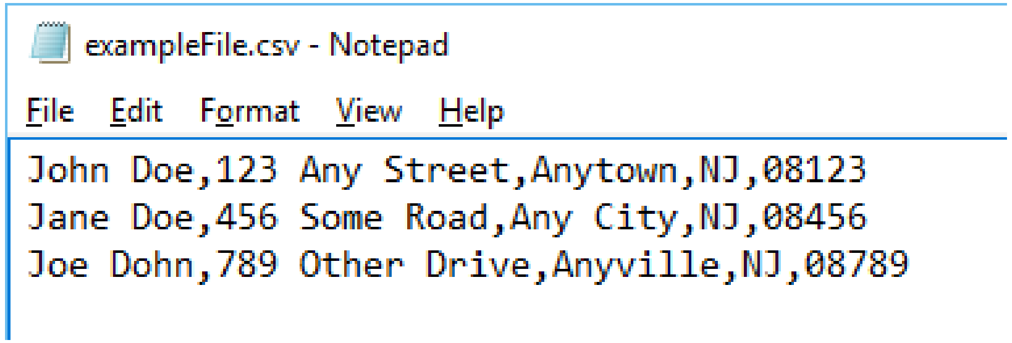
```
string_to_tokenize = "Philadelphia, PA"
tokens = string_to_tokenize.split(",")

for token in tokens :
  #Print Each Token
  print("Token:", token.lstrip().rstrip(), ".", sep="")

print("Done")
```

```
Total tokens= 2
Token: Philadelphia
Token: PA
Done
```

CSV files

- ***Comma separated values*** (or CSV) is a widely recognized text file format where each line of the file contains values that are separated by commas.

- Many database and spreadsheet programs use CSV format to export and import data.
  - Filename ends with .csv

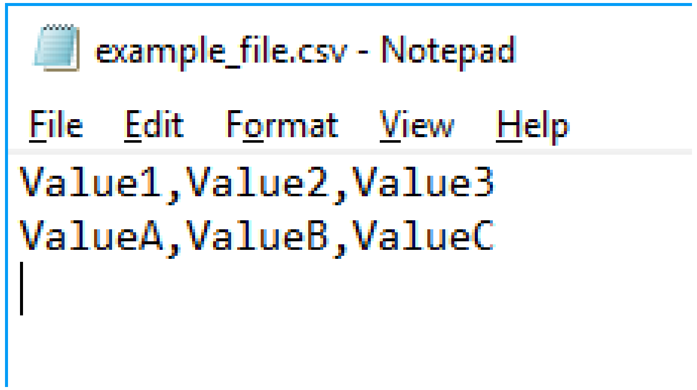


```
John Doe,123 Any Street,Anytown,NJ,08123
Jane Doe,456 Some Road,Any City,NJ,08456
Joe Dohn,789 Other Drive,Anyville,NJ,08789
```

Writing CSV files

- There is no special object for writing a CSV file.
  - Write the comma separated values as you would normally write to a file.

```
my_csv_file = open("example_file.csv", "w")
v1 = "ValueA"
v2 = "ValueB"
v3 = "ValueC"

my_csv_file.write("Value1,Value2,Value3\n")
my_csv_file.write(v1 + "," + v2 + "," + v1 + "\n")

my_csv_file.close()
```

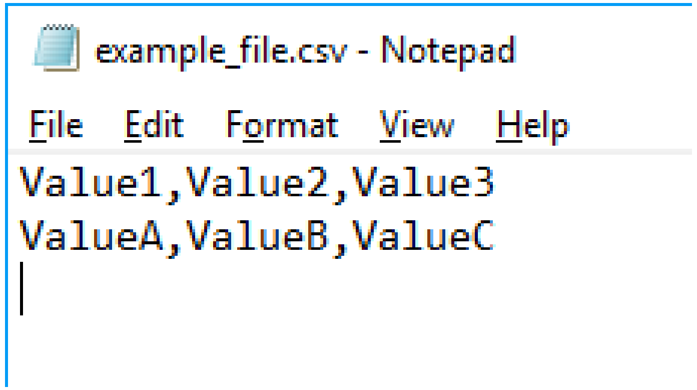


```
Value1,Value2,Value3
ValueA,ValueB,ValueC
```

Reading CSV files

- There is no special object for reading a CSV file.
  - Read the file as you would read any text file.
  - For each line in the file, split the line using a comma as the delimiter.

```
my_csv_file = open("example_file.csv", "r")

for line in my_csv_file :
  tokens = line.rstrip("\n").split(",")
  #Use the tokens list to access
  #the individual values of that line

my_csv_file.close()
```



```
Value1,Value2,Value3
ValueA,ValueB,ValueC
```