String Processing

CS 1111
Introduction to Programming

Spring 2019

[*The Coder's Apprentice*, §10]

Collections

Ordered, Dup allow

- List
- Range
- String
- Tuple

- *collection[index]*
- Access an element's value via index
- Index is int, starting at 0

Unordered, No Dup

- Dict

- *collection[key]*
- Access an element's value via key
- Key is primitive type, unique

Review

Strings

- Sequence of characters (letters, numbers, punctuation marks, spaces, …)
- String literals = sequence of characters enclosed by quotations
  - print("Hello World!")
- Quotations inside quotations
  - "Python's fun!"
- Must match quotations
  - Single with single, double with double

Length of Strings

Review

- Length of a string = the number of characters in a string

- len("Hello World!")
- len("")

length = 12

length of empty string = 0

String Concatenation ("+")

Review

- Attach string to another string

```
firstName = "Thomas"
lastName = "Jefferson"
name = firstName + " " + lastname
print("Name is ", name)                    # Name is Thomas Jefferson
```

String Repetition ("*")

Review

- Produce a string that is composed of repeated characters

```
dashes = "-" * 10          # dashes = "----------"
```

Note: results in string

String Conversion

Review

- Convert numbers to strings

```
average_grades = 85
result = "Average Test1 grade is " + str(average_grades)
print(result)
```

What happens if average_grades is not casted

- Convert strings to numbers

```
prod_id = int("149")
price = float("85.45")
print(prod_id)
print(price)
```

Review

Special Characters

- Escape character

  print("Python is an \"interpreted\" language")

- New line character

  print("Jake\nJohn\nJane\n")

String Equality

string1 equals to string2 if and only if

- They are of the same length, and
- They contain the exact same sequence of characters, and case sensitive

name1 = "Thomas Jefferson"

name2 = "Thomas Jefferson"

name3 = "Thomas jefferson"

name4 = "Thomas Jenkinson"

only name1 = name2

Slicing

Refer to a range of values

s[start : stop]

- Slice from start index (inclusive) to stop index (exclusive)

s[start :]

- Slice from start index (inclusive) to the end of the collection

s[: stop]

- Slice from the beginning of the collection to the stop index (exclusive)

s[start : stop : step]

- Slice from start index (inclusive) to stop index (exclusive), skip step-1 then grab

in Operator

substring in string

- Returns True if substring exists in string
- False, otherwise

```
name = "Thomas Jefferson"
print("Jeff" in name)          # True
```

String Testing Methods: isdigit( ), isspace( )

isdigit( )

- Returns True if the string *s* consists of only digits and contains at least one character; and False otherwise

```
name = "Thomas Jefferson"
print(name.isdigit())        # False, contains non-digits
```

isspace( )

- Returns True if the string *s* consists of only white space characters (blank, newline, tab) and contains at least one character; and False otherwise

```
name = "Thomas Jefferson"
print(name.isspace())        # False, contains non-white space
```

String Testing Methods: isalnum( ), isalpha( )

isalnum( )

- Returns True if the string *s* consists of only letters or digits and contains at least one character; and False otherwise

```
name = "Thomas Jefferson"
print(name.isalnum())          # False, contains space
```

isalpha( )

- Returns True if the string *s* consists of only letters and contains at least one character; and False otherwise

```
name = "Thomas Jefferson"
print(name.isalpha())          # False, contains space
```

String Testing Methods: islower( ), isupper( )

islower( )

- Returns True if the characters in the string are lowercase and the string contains at least one character; and False otherwise

```
name = "Thomas Jefferson"
print(name.islower())          # False, contains uppercase
```

isupper( )

- Returns True if the characters in the string are uppercase and the string contains at least one character; and False otherwise

```
name = "Thomas Jefferson"
print(name.isupper())          # False, contains lowercase
```

String Modification Methods: lower( ), upper( )

lower( )

- Convert a string to lowercase

```
name = "Thomas Jefferson"
print(name.lower())          # thomas jefferson
```

upper( )

- Convert a string to uppercase

```
name = "Thomas Jefferson"
print(name.upper())          #THOMAS JEFFERSON
```

String Modification Methods: strip( ), rstrip( ), and lstrip( )

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

```
name = "  Thomas Jefferson  "
print(name.strip())
```

```
# Thomas Jefferson
(no spaces)
```

rstrip(char)

- Returns a copy of the string with all instances of *char* that appear at the end of the string removed

lstrip(char)

- Returns a copy of the string with all instances of *char* that appear at the beginning of the string removed

Default: remove whitespace characters

- spaces, newlines (\n), and tabs (\t)

Other String Methods: join( )

join(seq_list)

- Returns a copy of the string, which is the concatenation of the string with intervening occurrences of *seq_list.*
- *seq_list* must be a list

```
name = "Thomas Jefferson"
seq_list = ["$", "--", "@"]
print(name.join(seq_list))
                    #    $Thomas Jefferson--Thomas Jefferson@
```

Other String Methods: split( )

split(delimiter)

- Returns a list of all the words in the string, using *delimiter* as the separator
- Default separator: space

```
name = "Thomas Jefferson"
print(name.split())          #   ['Thomas', 'Jefferson']
```

Other String Methods: count( )

count(substring)

- Returns the number of non-overlapping occurrences of *substring* in the string *s*

```
name = "Thomas Jefferson"
print(name.count("f"))          # 2
print(name.count("ff"))         # 1
```

Search and Replace Methods: startswith( )

startswith(substring)

- Returns True if the string *s* begins with *substring* and False otherwise

```
name = "Thomas Jefferson"
print(name.startswith("Th"))          # True
```

Search and Replace Methods: endswith( )

endswith(substring)

- Returns True if the string *s* ends with *substring* and False otherwise

```
name = "Thomas Jefferson"
print(name.endswith("son"))          # True
```

Search and Replace Methods: find( ) and rfind( )

find(substring)

- Returns the lowest index in the string *s* where *substring* begins, or -1 if substring is not found

```
name = "Thomas Jefferson"
print(name.find("f"))          # 9
print(name.find("cs1111"))     # -1
```

rfind(substring)

- Returns the highest index in the string *s* where *substring* begins, or -1 if substring is not found

```
name = "Thomas Jefferson"
print(name.rfind("f"))         # 10
```

Search and Replace Methods: index( )

index(substring, beg=0 end=len(string))

- Returns the lowest index in the string *s* where *substring* begins, or raises an exception (ValueError: substring not found) if substring is not found

```
str1 = "Thomas Jefferson"
str2 = "e"
print(str1.index(str2))          # 8
print(str1.index(str2, 9))       # 11
print(str1.index(str2, 9, 12))   # 11
print(str1.index(str2, 9, 11))   # error
    # from index 9, up to index 11 (not include index 11)
```

Search and Replace Methods: replace( )

replace(old, new)

- Returns a copy of the string with all instances of *old* replaced by *new*

```
name = "Thomas Jefferson"
print(name.replace("Thomas","George"))
                                    # George Jefferson
```

Summary

- Must know (based on exam2 topic list, as of 03/04/2019)
  - `substring in string`
  - `string1 + string2`
  - `string.strip()`
  - `string.split()`
  - `string.split(delimiter)`
  - `string.find(substring)`