

Module 7

Strings, Sequences, Slicing

Emily Zhao

CSCI-UA.0002

Agenda

- Review Strings
- Practice Problems

What is a string again?

What is a string again?

- A data type
- Can be described as a “sequence of characters”
- Characters are arranged in a certain order

What are the two ways that I can iterate over a string?

What are the two ways that I can iterate over a string?

```
for c in "Emily":  
    print(c)
```

```
>> E  
>> M  
>> I  
>> L  
>> Y
```

```
name = "Emily"
```

```
for i in range(5):  
    print(name[i])
```

```
>> E      # name[0]  
>> M      # name[1]  
>> I      # name[2]  
>> L      # name[3]  
>> Y      # name[4]
```

String indexing


Forward direction indexing

| | | | | | |
|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 |
| P | y | t | h | o | n |
| -6 | -5 | -4 | -3 | -2 | -1 |

Backward direction indexing

Iterating over a String using Indexing

```
word = "Emily"
for i in range(0, 5):
    print (word[i])

word2 = "Supercalifragilisticexpialidocious"
for i in range(0, ):
    print(word[i])
```


Iterating over a String using Indexing

```
word = "Emily"  
for i in range(0, 5):  
    print (word[i])
```

```
word2 = "Supercalifragilisticexpialidocious"  
for i in range(0, len(word2)):  
    print(word[i])
```

Programming Challenge

Write a function that counts the #'s of vowels in a string
(A,E,I,O,U)

```
# name: countVowels  
# input: a string  
# processing: counts the number of vowels in a word  
# the number of vowels
```

```
def countVowels(word):  
    vowelCount = 0  
    for c in word:  
        if c == "a" or c == "e" or c == "i" or c == "o" or c == "u":  
            vowelCount += 1  
    return vowelCount  
  
print(countVowels("hello"))
```

String slicing

Slicing a String

- Sometimes you may find it necessary to extract a portion of a string from another string.
- You can use “slicing” notation in Python to extract a span of characters from a string into a new string. We call this new String a "substring". For example:

```
full_name = "Emily Zhao"  
first_name = full_name[0:5]  
  
print (first_name)  
  
>> Emily
```

Slicing a String

```
substring = bigstring[start:end:step]
```

- You must supply at least a start or an ending index value.
- Substrings contain all characters starting at the start value specified and continue up to (but do not include) the ending value.
- Omitting a starting or ending index value will cause Python to assume you want to start at the beginning of the string (if you omit a start value) or you want to continue slicing to the end of the string (if you omit the end value)
- **This should look a lot like the range function!**

String Slicing Notation

What will the following code print?

```
word = "Superman sings in the shower."
```

| | |
|--|---|
| <code>print (word[0:8])</code> | <code>> Superman</code> |
| <code>print (word[9:14])</code> | <code>> sings</code> |
| <code>print (word[:5])</code> | <code>> Super</code> |
| <code>print (word[9:])</code> | <code>> sings in the shower.</code> |
| <code>print (word[-7:])</code> | <code>> shower.</code> |
| <code>print (word[0:len(word):3])</code> | <code>> Seasgit or</code> |
| <code>print (word[30])</code> | <code>> IndexError: string index out of range</code> |

Programming Challenge: Pig Latin Translator

- Write a program that asks the user for a word
- Translate their word into Pig Latin.
A Pig Latin word can be generated using the following rules:
 - Remove the first letter of the word
 - Place the first letter of the word at the end of the word
 - Add the string "ay" to the end of the word




```
word = input("Choose a word to convert to pig latin: ")  
pig_word = word[1:len(word)] + word[0] + "ay"  
print(word, "is", pig_word, "in pig latin.")
```

**Strings cannot be changed
once they are created.**

→ True or false?

**Strings cannot be changed
once they are created.**

→ True!

Strings are “Immutable”

- Strings are an **immutable** data type. This means that they cannot be changed once they are created.
- This may seem counter intuitive, since we have been doing the following since the beginning of the semester:

```
word = "superman"
print ("word is", word)
word = "wonder woman"
print ("word is now", word)

>> word is superman
>> word is now wonder woman
```

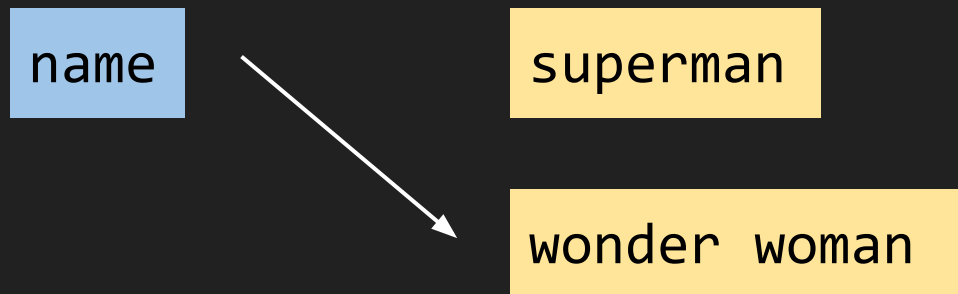
Strings are “Immutable”

- What actually happens “under the hood” is that Python creates a separate string in your computer’s memory and “points” to that string instead of the original one.

```
name = 'superman'
```



```
name = 'wonder woman'
```



Strings are “Immutable”

- This means that you cannot change the individual characters within a string using index notation. You will raise an exception if you attempt to do so.

```
word = "Superman"  
word[0] = "X"  
  
# exception!
```

How do you change a string then?

Programming Challenge

Write a program that replaces all vowels in a String with the underscore character (_)

hello → **h_ll_**

Will this work?

```
word = "hello"

for i in range(len(word)):
    # if the letter is a vowel, reassign it to _
    if word[i] in ["a", "e", "i", "o", "u"]:
        word[i] = "_"

print(word)
```

How do you change a string then? → Gotta make a new one!

```
word = "hello"
new_word = ""

for i in range(len(word)):
    # if the letter is a vowel, add _ to new word
    if word[i] in ["a", "e", "i", "o", "u"]:
        new_word += "_"
    # if the letter is a consonant, just add the letter
    else:
        new_word += word[i]

print(new_word)
```

Testing Strings with in and not in

- The "in" operator is a Boolean operator that you can use to test to see if a substring exists inside of another string. Example:

```
word = "Grace Lily John Chris Tom"

if "Chris" in word:
    print ("found him!")
else:
    print ("can't find Chris")
```

- When you construct an expression with the “in” operator the result will evaluate to a Boolean

Programming Challenge: Balance Test

- Write a program to check if two strings are “balanced.”
- For example, strings s1 and s2 are balanced if all the characters in the s1 are present in s2. The character’s position doesn’t matter.

Case 1:

```
s1 = "Ya"
```

```
s2 = "PYnative"
```

```
> True
```

Case2:

```
s1 = "Ynf"
```

```
s2 = "PYnative"
```

```
> False
```

```
s1 = "Yn"
s2 = "PYnative"
flag = True

# loop through all the characters in s1
for c in s1:
    # test to see if the character is in s2
    if c in s2:
        continue
    else:
        flag = False

print(flag)
```

Programming Challenge: Palindrome Tester

- Write a program that asks the user for a word
- Determine whether the supplied word is a palindrome (a word that reads the same backwards and forwards)

RACECAR

```
word = input("Check if a word is a palindrome: ")
backwards_word = ""

for i in range(len(word)-1, -1, -1):
    backwards_word += word[i]

print(backwards_word)
if word == backwards_word:
    print(word, "is a palindrome!")
else:
    print(word, "is not a palindrome.")
```



```
word = input("Check if a word is a palindrome: ")  
  
if word == word[::-1]:  
    print(word, "is a palindrome!")  
else:  
    print(word, "is not a palindrome.")
```

String functions

Getting the largest and smallest character in a string

- You can use two built in Python functions to obtain the maximum and minimum characters in a string (based on their ASCII codes)

```
a = max("python")  
b = min("python")
```

```
print ("max:", a)  
print ("min:", b)
```

```
>> y  
>> h
```

String methods

```
stringvariable.method(arguments)
```

String methods

| | |
|-------------------------|---|
| <code>.isalnum()</code> | True if all characters are alphanumeric |
| <code>.isalpha()</code> | True if all characters are alphabetic |
| <code>.isdigit()</code> | True if all characters are digits |
| <code>.islower()</code> | True if all alpha characters are lower |
| <code>.isspace()</code> | True if all characters are “whitespace” |
| <code>.isupper()</code> | True if all alpha characters are upper |

String modifications

| | |
|----------------------------|--|
| <code>.lower()</code> | Returns a lowercase version of the string |
| <code>.upper()</code> | Returns an uppercase version of the string |
| <code>.rstrip()</code> | Removes whitespace at end of string |
| <code>.lstrip()</code> | Removes leading whitespace characters |
| <code>.capitalize()</code> | Returns a copy of the string with the first character capitalized |
| <code>.title()</code> | Returns a copy of the string with the first character of each word capitalized |
| <code>.swapcase()</code> | Returns a copy of the string where case is swapped among all alpha characters |

Programming Challenge: Headline Generator

- There's a popular subreddit that likes to write the titles to their posts like such:
- Write a function that takes a string and returns a title that randomly alternates the casing of the text.



```
import random
title = "only survivor forced to leave family breaks neck during escape"

def headlineGen(title):
    new_title = ""

    for i in range(len(title)):
        num = random.randint(0,1)
        if num == 0:
            new_title += title[i].lower()
        else:
            new_title += title[i].upper()

    return new_title

print(headlineGen(title))
```


Finding substrings

- You can find whether a string exists inside another string by using the `find()` method. Example:

```
word = "Like finding a needle in a haystack!"  
location = word.find("needle")  
print (location)
```

- The `find()` method will return the index of the first occurrence of a substring within a string.
- If the `find()` method cannot find the desired substring it will return -1

Searching + Replacing

- Programs often need to perform search and replace functions on data, much like the “find and replace” functionality that exists in your word processor.
- You can have Python replace all occurrences of a substring by using the `replace()` method.

```
text = "Voldemort had one goal in life - to kill Harry Potter."  
newText = text.replace("Voldemort", "He who shall not be named")  
  
print (newText)
```

Getting the ASCII value of a character

- Remember that Python (and all programming languages) use the standard ASCII encoding system to organize individual characters
- You can use the `ord()` function to look up the ASCII value of a character by doing the following:
- The `ord()` function accepts one argument – a single character- and returns an integer that represents the ASCII value of that character

```
value = ord("A")  
>> 65
```

Getting the ASCII value of a character

- You can also reverse the process and turn an integer into its equivalent letter value using the `chr()` function

```
value = chr(65)
```

```
>> A
```

| | | | | | | | | | | | | | | | |
|----|------------|----|------------|----|-----------|----|---|----|---|----|---|-----|---|-----|------------|
| 0 | <u>NUL</u> | 16 | <u>DLE</u> | 32 | <u>SP</u> | 48 | 0 | 64 | @ | 80 | P | 96 | ` | 112 | p |
| 1 | <u>SOH</u> | 17 | <u>DC1</u> | 33 | ! | 49 | 1 | 65 | A | 81 | Q | 97 | a | 113 | q |
| 2 | <u>STX</u> | 18 | <u>DC2</u> | 34 | " | 50 | 2 | 66 | B | 82 | R | 98 | b | 114 | r |
| 3 | <u>ETX</u> | 19 | <u>DC3</u> | 35 | # | 51 | 3 | 67 | C | 83 | S | 99 | c | 115 | s |
| 4 | <u>EOT</u> | 20 | <u>DC4</u> | 36 | \$ | 52 | 4 | 68 | D | 84 | T | 100 | d | 116 | t |
| 5 | <u>ENQ</u> | 21 | <u>NAK</u> | 37 | % | 53 | 5 | 69 | E | 85 | U | 101 | e | 117 | u |
| 6 | <u>ACK</u> | 22 | <u>SYN</u> | 38 | & | 54 | 6 | 70 | F | 86 | V | 102 | f | 118 | v |
| 7 | <u>BEL</u> | 23 | <u>ETB</u> | 39 | ' | 55 | 7 | 71 | G | 87 | W | 103 | g | 119 | w |
| 8 | <u>BS</u> | 24 | <u>CAN</u> | 40 | (| 56 | 8 | 72 | H | 88 | X | 104 | h | 120 | x |
| 9 | <u>HT</u> | 25 | <u>EM</u> | 41 |) | 57 | 9 | 73 | I | 89 | Y | 105 | i | 121 | y |
| 10 | <u>LF</u> | 26 | <u>SUB</u> | 42 | * | 58 | : | 74 | J | 90 | Z | 106 | j | 122 | z |
| 11 | <u>VT</u> | 27 | <u>ESC</u> | 43 | + | 59 | ; | 75 | K | 91 | [| 107 | k | 123 | { |
| 12 | <u>FF</u> | 28 | <u>FS</u> | 44 | , | 60 | < | 76 | L | 92 | \ | 108 | l | 124 | |
| 13 | <u>CR</u> | 29 | <u>GS</u> | 45 | - | 61 | = | 77 | M | 93 |] | 109 | m | 125 | } |
| 14 | <u>SO</u> | 30 | <u>RS</u> | 46 | . | 62 | > | 78 | N | 94 | ^ | 110 | n | 126 | ~ |
| 15 | <u>SI</u> | 31 | <u>US</u> | 47 | / | 63 | ? | 79 | O | 95 | _ | 111 | o | 127 | <u>DEL</u> |

Programming Challenge:

Calculate the sum and average of the digits present in a string.
Also return how many special characters there are.

```
str1 = "PYnative29@###$!#8496"
```

```
> Sum: 38
```

```
> Average: 6.3333333333333333
```

```
> Special character count: 6
```

```
str1 = "PYnative29@###$!#8496"  
special = 0  
nums = 0  
total = 0
```

```
for c in str1:  
    if c.isalnum() == False:  
        special += 1  
    if c.isdigit():  
        nums += 1  
        total += int(c)
```

```
print("Sum:", total)  
print("Average:", total/nums)  
print("Special characters:", special)
```

Homework

- Self-Paced Learning Module #8
- Quiz #8