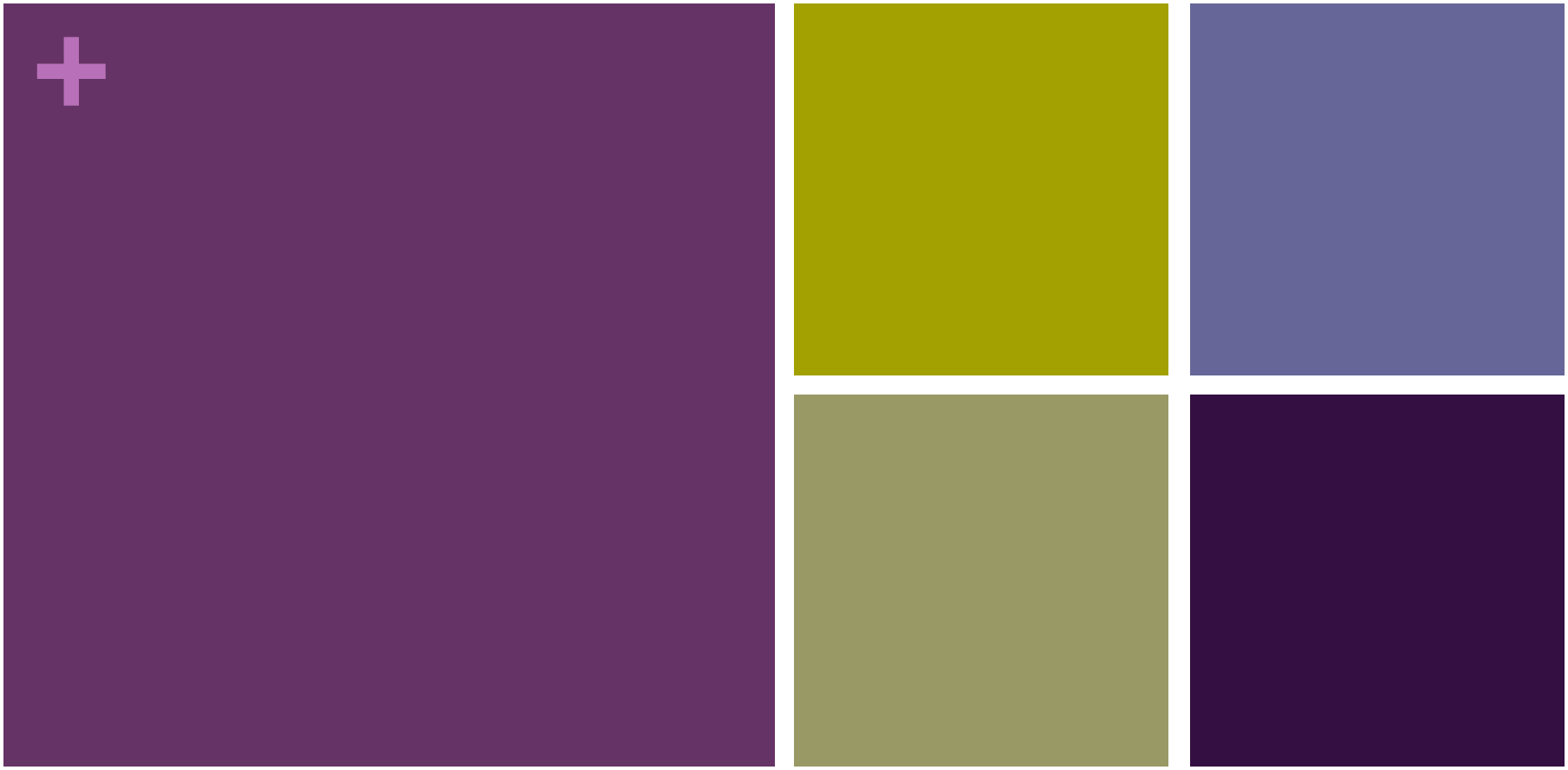


Introduction to String Manipulation

+

More about Strings …

What is a String?

- A String is a data type in the Python programming language
- A String can be described as a "sequence of characters"
- Characters are arranged in a particular position in a String. For example:

```
word1 ="hello"
```

- In this String the character 'h' is the first character in the String, followed by 'e', etc. The following String contains the same characters but they are arranged in a different position:

```
word2 = "loleh"
```

Examining the Characters within a String

Accessing the individual characters in a String

- Some programming tasks require you to access the individual characters that are contained within a string.
- You can isolate individual characters within a string by using a for loop. The target variable in your loop will assume each character in your string one at a time. For example:

```
for c in “Craig”:
     print (c)

>> C
>> r
>> a
>> I
>> g
```

Using the individual characters in a string for calculations

- Now that you can iterate over a string it is pretty easy to conceptualize how you could analyze the contents of a string.
- Programming challenge: write a program to count the # of "S" characters in the string

```
"Sally Sells Sea Shells By the Sea Shore."
```

- Programming challenge: write a program to count the # of "S" and "s" characters in the string

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

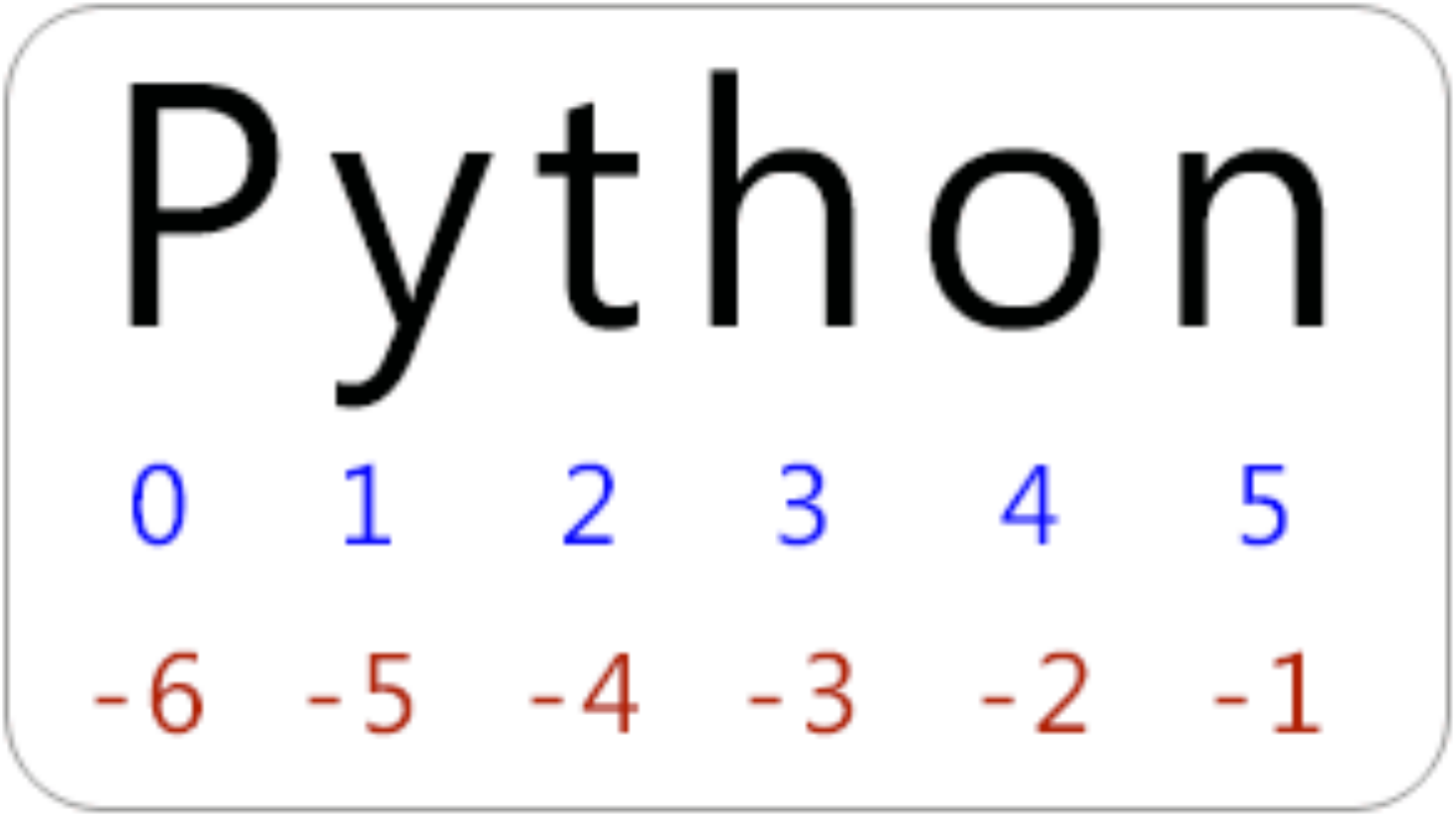
Using the individual characters in a string for calculations

```
word = 'Superman sings in the shower'

counter = 0

for c in word:

    if c == 's' or c ='S':

        counter += 1

print ("There are", counter, "'s' characters in
  the string", word)
```

+

String Indexing

Looking at the string index



Indexing

- Another way to analyze a string is to use "indexing" notation
- Indexing is a way to reference individual elements within a string based on their position
- You can create an index by placing an integer value inside a pair of square brackets after your string variable.

```
word = "superman"
print (word[0])
>> s
```

- Indexes always start with zero. The first character in a string is considered the "0th" element in the string.
- The index of last character in a string is equal to the length of the string - 1

Indexing

```
# Superman

# 01234567

word = "Superman”

print (len(word))

print (word[0])

>> 8

>> S
```

Index Exceptions

- You will raise an exception if you attempt to access an index of a string that does not exist. For example:

```
word = "Superman”
print (word[10])

# error!  index 10 doesn’t exist on word
```

Iterating over a String using Indexing

- You can use a "for" loop to iterate over the characters in a String, like this:

```
word = "hello"
for c in word:
     print (c)
```

- However, if you want to iterate over the String using indexing you will need to generate a range of numbers that correlate to the indexes you wish to visit. For example:

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

Iterating over a String using Indexing

- Most times we won't know how long a String is when we begin to examine it.
- We can easily count the # of characters in a String using the len() function though. For example:

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

Programing Challenge

- Write a program to count the # of "S" characters in the following String.
- Use String indexing to examine each character.

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

Programing Challenge

- Given the String:

  `"Superman sings in the shower."`

- Write a program that examines the String in reverse order

Programing Challenge

- Given the String:

  `"Superman sings in the shower."`

- Write a program that examines every other character ("S", "p", "r", etc.)

Negative indexing

- You can use negative integers to identify character positions relative to the end of the string
- Example:

```
word = "Superman”
print (word[-1], word[-2], word[-3])

>> n a m
```

+

String Immutability

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:

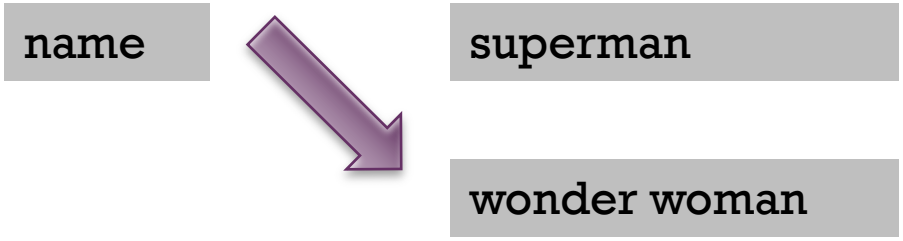
```
word = "superman"
print ("word is", word)
word = "wonder woman"
print ("word is now", word)

>> 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!
```

Changing a String

- In order to make changes to a String you will need to create a new String variable
- For example, consider this programming challenge:
  - Write a program that replaces all vowels in a String with the underscore character.
- At first glance you could assume that you could simply use String indexing to change the characters in question. For example:

```
word = "hello"
word[1] = "_"
word[4] = "_"
```

- But because Strings are immutable this won't work!

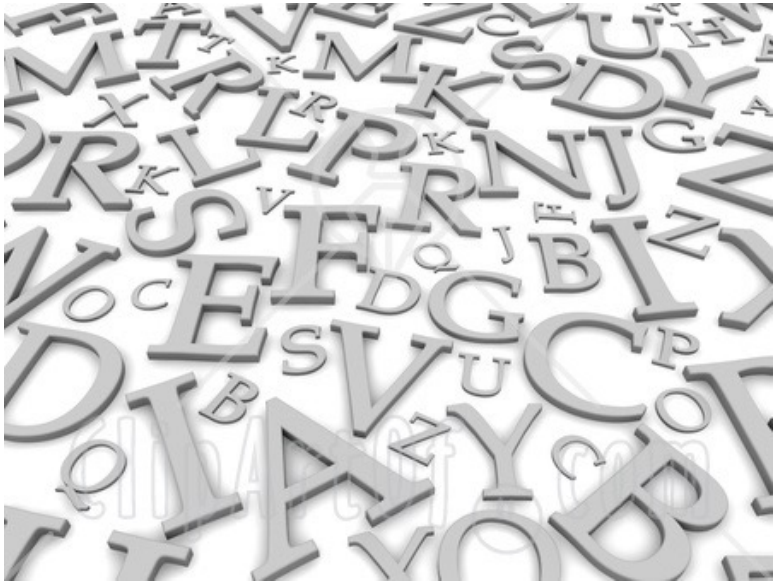
Changing a String

- Instead, we can use a new String variable to hold our changes and then use a loop to examine each character in sequence to "build" the new String. For example:

```
word    = "hello"
newword = ""

for c in word:
    if c == 'e' or c == 'o':
        newword += "_"
    else:
        newword += c
```

Character Counter

- Write a function that accepts three arguments – a string, a character to search for and a replacement character
- Have your function search the string for all occurrences of that character and perform the replacement
- Return the new String

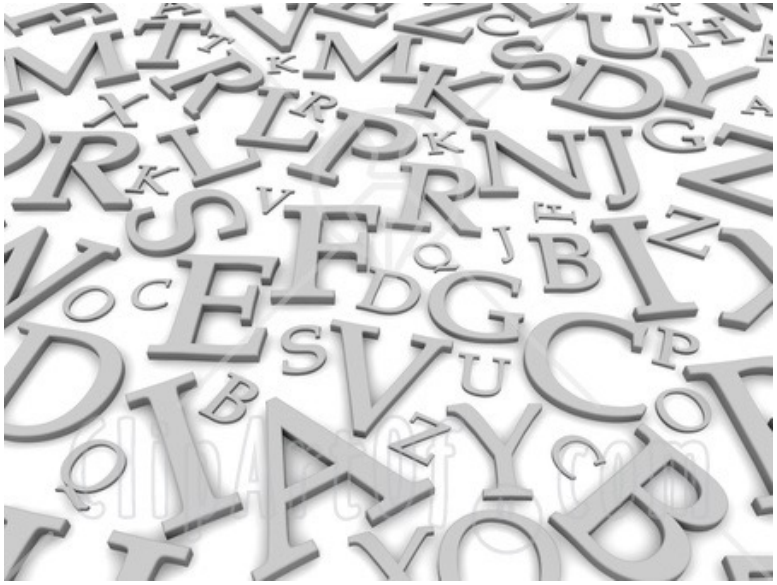


+

Some Practice Problems

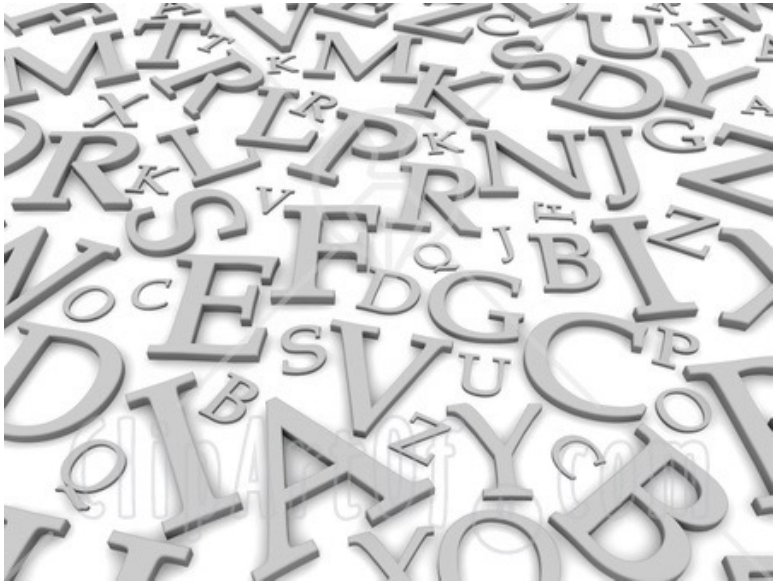
Character Counter

- Write a function that accepts two arguments – a string and an single character
- Have your function search the string for all occurrences of that character
- Return the number of times the character was found in the supplied string



Vowel Counter

- Write a new function that counts the #'s of vowels in a string (A,E,I,O,U) and returns the number.



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)

+

The ASCII Table & Strings

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:

```
value = ord("A")

>> 65
```

- The ord() function accepts one argument – a single character- and returns an integer that represents the ASCII value of that character

Programming Challenge

- Write a program that asks the user for a string
- Print the string, character by character, along with its associated ASCII value
- Example:

```
give me a name: craig

>> c = 99
>> r = 114
>> a = 97
>> i = 105
>> g = 103
```

Creating strings from their ASCII codes

- The ord() function turns a single character into its equivalent ASCII integer code
- You can also reverse the process and turn an integer into its equivalent letter value using the chr() function
- Example:

```
x = chr(65)
print (x)

>> A
```

Programming Challenge

- Write a program that generates a random password for the user
- Passwords should follow these rules:
  - Be at least 10 characters long
  - Contain a mixture of uppercase letters, lowercase letters and numbers

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

+

String Functions

String Functions

- We already know a few functions that can be used in conjunction with Strings
- For example, the len() function can be used to count the # of characters in a String
- As you can imagine, there are other functions that you can use when working with Strings to make things easier for you as a programmer!

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) – Example:

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

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

>> y
>> h
```

Programming Challenge

- Write a program to find the letter that is closest to the end of the alphabet in the following Strings.

```
zelda
peach
apple
```

- Then find the letter that is closest to the beginning of the alphabet.

Programming Challenge

- Write a program to find the letter that is closest to the end of the alphabet in the following Strings. ***Don't use the max function!***

```
zelda
peach
apple
```

- Then find the letter that is closest to the beginning of the alphabet. ***Don't use the min function!***

Slicing a String

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 =“John Smith”
first_name = full_name[0:4]

print (first_name)

>> John
```

String Slicing Notation

- When you ask Python to slice a string you need to use bracket notation to specify the index range of the characters you wish to extract.
- The syntax for this notation is as follows:

```
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."


print (word[0:8])

print (word[9:14])

print (word[:5])

print (word[9:])

print (word[-7:])

print (word[0:len(word):3])
```

Programming Challenge: Username Generation

- You just accepted a position at NYU's ITS department and your first task is to write a program that generates student Net IDs for all incoming freshmen
- Net IDs are generated as follows:
  - The first two characters of a student's first name
  - The first two characters of a student's last name
  - The last three characters of a student's N#
- Write a program that asks the user for these three pieces of information (first name, last name and N#) and generate their Net ID.
- Note that if a student's first name or last name is less than 2 characters then you should use the entire first name or last name.

+

String Operators

String Operators

- We already know that the "+" and "*" operators can be used in conjunction with a String
- The "+" operator can be used to "concatenate" two Strings together
- The "*" operator can be used to repeat a String a certain number of times (specified as an integer)

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 = "Jackson James 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

Testing Strings with in and not in

- You can also test to see if a string is not in another string by using the "not" keyword in your expression. Example:

```
word = "Jackson James John Chris Tom"

if "Johnny" not in word:
    print ("No Johnny!")
else:
    print ("Johnny is here!")
```

Programming Challenge: Simple Username and Password Validator

- Write a program that asks a user for a username and a password
- Ensure that the user doesn't use their username as part of their password. For example:

```
username:  craig
password:  craig1234 # invalid!

username:  craig
password:  abc123    # ok!
```

+

String Methods

String Methods

- A "method" is a function that belongs to an "object", and performs some operation on that object. Example:
- We really haven't explored objects much in class as of yet, but strings can be used as a good introduction to objects and methods.
- The general syntax for calling a method on an object looks a lot like a function call, with the notable exception that the method call generally does not need to pass an argument to the function specifying which piece of data is being operated on. Syntax:

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

String Testing Methods

- String testing methods allow you to determine if certain patterns existing within a given string variable. For example:

```
word = '1234'
if word.isdigit() == True:
    print ("All chars in word are digits!")
else:
    print ("Not all chars in word are digits!")
```

- In the above example we are calling the "isdigit" method on the string variable "word". This method returns True if all characters contained in this string are numeric digits (0-9) and False if not.

String Testing Methods

Method	Output
`isalnum()`	True if all characters are alphanumeric
`isalpha()`	True if all characters are alphabetic
`isdigit()`	True if all characters are digits
`islower()`	True is all alpha characters are lower
`isspace()`	True if all characters are “whitespace”
`isupper()`	True if all alpha characters are upper

Programming Challenge: Character Analysis

- Write a program that counts the # of spaces, digits, vowels and consonants in a string that the user inputs. Example:

```
>> Enter a phrase:  Sally Sells 1000 sea
shells.

>> Spaces:  4
>> Digits:  4
>> Vowels:  5
>> Consonants:  12
```

Modification Methods

- Remember that strings are immutable and cannot be directly changed.
- With that said, they do contain a number of "modification" methods that return new copies of the string that reflect a desired change. For example:

```
word = "Craig"
newword = word.lower()
print (word)

>> craig
```

String Modification Methods

Method	Output
`lower()`	Returns a lowercase version of the string
`upper()`	Returns an uppercase version of the string
`rstrip()`	Removes whitespace at end of string
`lstrip()`	Removes leading whitespace characters
`capitalize()`	Returns a copy of the string with the first character capitalized
`title()`	Returns a copy of the string with the first character of each word capitalized
`swapcase()`	Returns a copy of the string where case is swapped among all alpha characters

Programming Challenge

- Write a program that accepts a phrase from the user
- Strip out any leading or trailing "white space" from the string
- If the string has an even number of characters, make it all lowercase
- If the string has an odd number of characters, make it all uppercase

Searching and 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.

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

Programming Challenge: He Who Shall Not Be Named

- Write a program that asks a user to type in a message.
- If they use the word "voldemort" anywhere in their message you should warn them to be more careful!

Replacing substrings

- You can have Python replace all occurrences of a substring by using the replace() method. Example:

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

print (word)
print (newword)

>> Voldemort had one goal in life - to kill Harry Potter.
>> He who shall not be named had one goal in life - to kill Harry
Potter.
```

Programming Challenge: Redacted!

- Write a program that redacts all instances of "Python" from the following string:

  "Python is a programming language that is fairly easy to learn. Python is great for beginning and advanced programmers."

- To produce the following:

  "_____ is a programming language that is fairly easy to learn. _____ is great for beginning and advanced programmers."

Programming Challenge

- Write a function that counts the number of letters in a string. You should count both uppercase and lowercase letters. Example:

  ```
  x = count_letters("Python is fun")
  print (x)

  >> 11
  ```