

Unless otherwise noted, the content of this course material is licensed under a Creative Commons Attribution 3.0 License.
<http://creativecommons.org/licenses/by/3.0/>.

Copyright © 2009, Charles Severance.

You assume all responsibility for use and potential liability associated with any use of the material. Material contains copyrighted content, used in accordance with U.S. law. Copyright holders of content included in this material should contact open.michigan@umich.edu with any questions, corrections, or clarifications regarding the use of content. The Regents of the University of Michigan do not license the use of third party content posted to this site unless such a license is specifically granted in connection with particular content. Users of content are responsible for their compliance with applicable law. Mention of specific products in this material solely represents the opinion of the speaker and does not represent an endorsement by the University of Michigan. For more information about how to cite these materials visit <http://michigan.educommons.net/about/terms-of-use>.

Any medical information in this material is intended to inform and educate and is not a tool for self-diagnosis or a replacement for medical evaluation, advice, diagnosis or treatment by a healthcare professional. You should speak to your physician or make an appointment to be seen if you have questions or concerns about this information or your medical condition. Viewer discretion is advised: Material may contain medical images that may be disturbing to some viewers.

Chapter 4

Computing With Strings

Charles Severance

String Data Type

- A string is a sequence of characters
- A string literal uses quotes 'Hello' or "Hello"
- For strings, + means "concatenate"
- When a string contains numbers, it is still a string
- We can convert numbers in a string into a number using int()

```
>>> str1 = "Hello"  
>>> str2 = 'there'  
>>> bob = str1 + str2  
>>> print bobHellothere  
>>> str3 = '123'  
>>> str3 = str3 + 1
```

```
Traceback (most recent call last):  
File "<stdin>", line 1, in  
<module>TypeError: cannot  
concatenate 'str' and 'int' objects
```

```
>>> x = int(str3) + 1  
>>> print x  
124  
>>>
```

Input() is kind of useless

- When using `input("Prompt")` it is actually looking for an **expression** from input
- We use this just to prompt for numbers for simple programs
- We use `raw_input("Prompt")` for non-trivial programs

```
>>> x = input("Enter ")
```

```
Enter hello
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module> File  
"<string>", line 1, in <module>NameError:  
name 'hello' is not defined
```

```
>>> x = input("Enter ")
```

```
Enter 2 + 5
```

```
>>> print x
```

```
7
```

```
>>>
```

Real Programs

Use String Input

- We prefer to read data in using **strings** and then parse and convert the data as we need
- This gives us more control over error situations and/or bad user input
- Raw input numbers must be **converted** from strings

```
>>> name = raw_input("Enter:")  
Enter:Chuck
```

```
>>> print name  
Chuck
```

```
>>> apple = raw_input("Enter:")  
Enter:100
```

```
>>> x = apple - 10
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in  
<module>TypeError: unsupported operand  
type(s) for -: 'str' and 'int'
```

```
>>> x = int(apple) - 10
```

```
>>> print x
```

```
90
```

What Kind of Thing?

- We have a way to see what **kind** of data is in a variable
- We use a special function called `type()` to look at the *kind* of data is in a variable

```
>>> x = "Hello"
>>> print x
Hello
>>> print type(x)
<type 'str'>
>>> y = "Bob"
>>> print y
Bob
>>> print type(y)
<type 'str'>
>>> z = 45
>>> print z
45
>>> print type(z)
<type 'int'>
>>>
```

Looking Inside Strings

- We can get at every single character in a string using an index specified in square brackets
- The index value can be an expression that is computed
- The index value must be an integer

H	e	l	l	o		B	o	b
0	1	2	3	4	5	6	7	8

Figure 4.1: Indexing of the string "Hello Bob"

```
>>> greet = "Hello Bob"
>>> greet[0]
'H'
>>> print greet[0], greet[2], greet[4]
H l o
>>> x = 8
>>> print greet[x-2]
B
```

Slicing Strings

H	e	l	l	o		B	o	b
0	1	2	3	4	5	6	7	8

Figure 4.1: Indexing of the string "Hello Bob"

- We can also look at any continuous section of a string using a colon
- The second number is one beyond the end of the slice - “up to but not including”
- If a number is omitted it is assumed to be the beginning or end

```
>>> greet = "Hello Bob"
>>> greet[0:3]
'Hel'
>>> greet[5:9]
' Bob'
>>> greet[:5]
'Hello'
>>> greet[5:]
' Bob'
>>> greet[:]
'Hello Bob'
```


String indexes from the right

- **Negative index numbers** in a string start from the right (or end) of the string and work backwards

-9 -8 -7 -6 -5 -4 -3 -2 -1

H	e	l	l	o		B	o	b
0	1	2	3	4	5	6	7	8

Figure 4.1: Indexing of the string "Hello Bob"

```
>>> greet = "Hello Bob"
>>> greet[-1]
'b'
>>> greet[-3]
'B'
```

A Character too Far

- You will get a **python error** if you attempt to index beyond the end of a string.
- So be careful when constructing index values and slices

```
>>> zot = "abc"
```

```
>>> print zot[5]
```

```
Traceback (most recent call last):  
File "<stdin>", line 1, in  
<module>IndexError: string index  
out of range
```

```
>>>
```

String Operators

- We do a lot of work with strings and Python has a lot of support for strings
- With respect to strings, Python is a “smooth operator”

Operator	Meaning
+	Concatenation
*	Repetition
<string>[]	Indexing
<string>[:]	Slicing
len(<string>)	Length
for <var> in <string>	Iteration through characters

Table 4.1: Python string operations.

How Long is a String?

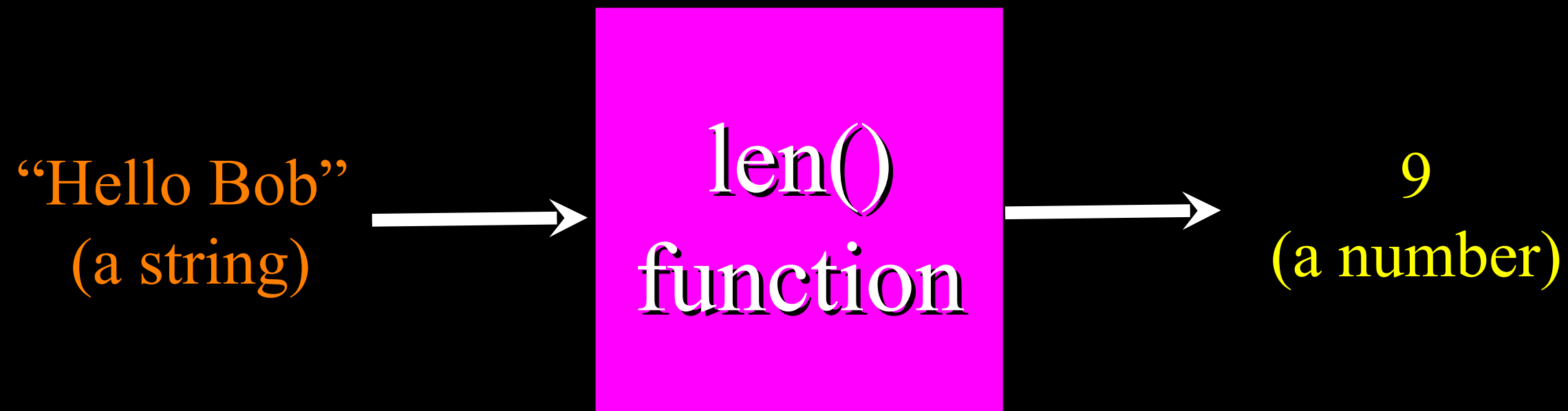
- The `len()` function takes a string as a parameter and returns the number of characters in the string
- Actually `len()` tells us the number of elements of any set or sequence

```
>>> greet = "Hello Bob"  
>>> print len(greet)  
9  
>>> x = [ 1, 2, "fred", 99]  
>>> print len(x)  
4  
>>>
```

Len Function

```
>>> greet = "Hello Bob"  
>>> x = len(greet)  
>>> print x  
9
```

A function is **some stored code** that we use. A function takes some **input** and produces an **output**.

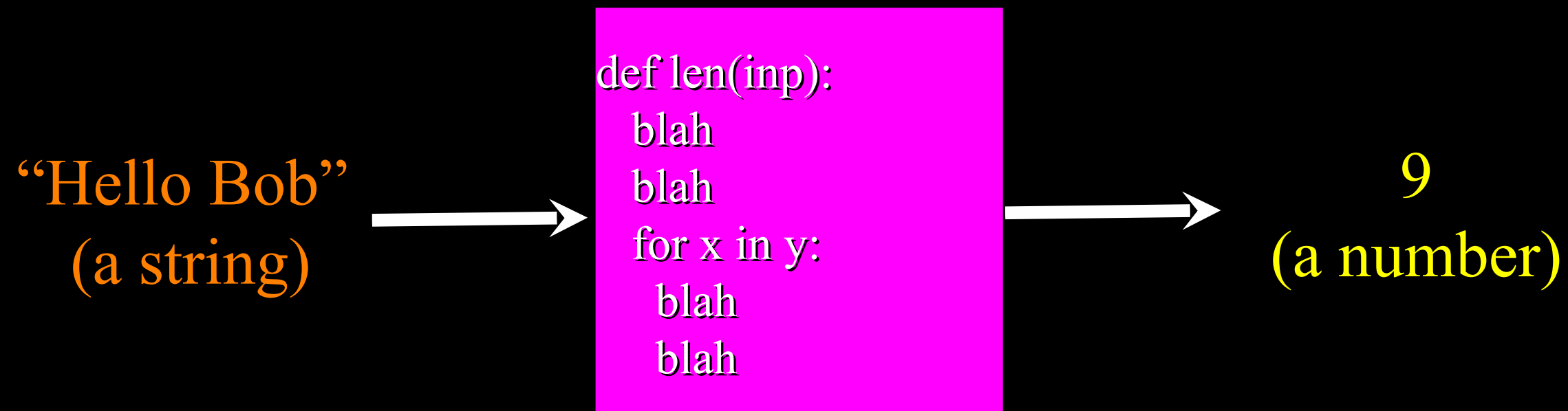


Guido wrote this code

Len Function

```
>>> greet = "Hello Bob"  
>>> x = len(greet)  
>>> print x  
9
```

A function is **some stored code** that we use. A function takes some **input** and produces an **output**.



Multiplying Strings?

- While it is seldom useful, the **asterisk** operator applies to strings

```
>>> zig = "Hi"  
>>> zag = zig * 3  
>>> print zag  
HiHiHi  
>>> x = " "*80
```

Looping Through a String

- A string is a **sequence** (ordered set) of characters
- The **for loop** iterates through a **sequence**, with the **iteration variable** taking successive values from the **sequence** each time the loop body is run

```
>>> zap = "Fred"  
>>> for xyz in zap:  
...     print xyz  
...  
F  
r  
e  
d  
>>>
```


String Library

String Library

- Python has a number of string operations which are in the **string library**
- We use these library operations quite often when we are pulling apart input data
- To use these, we import the **string library**

```
import string
```

```
zap =string.lower(greet)
```

What is a Library?

- Some super developers in the Python world write the libraries for us to use
- import `string`
- Somewhere there is a file `string.py` with a bunch of `def` statements

`string`

`def`

`def`

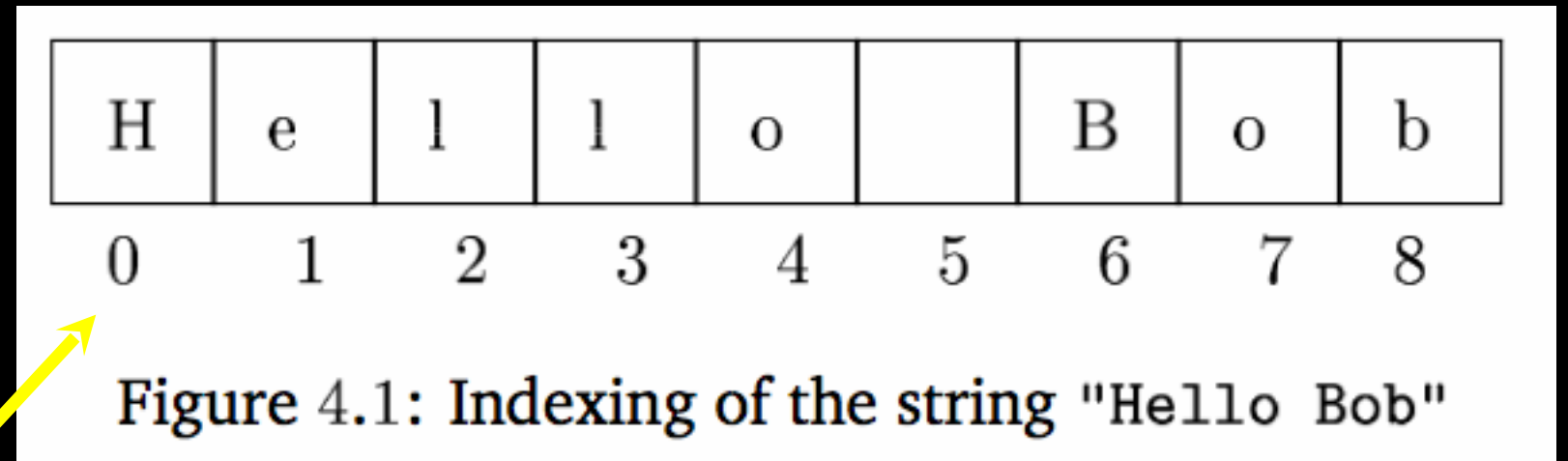
`def`

Function	Meaning
<code>capitalize(s)</code>	Copy of <code>s</code> with only the first character capitalized
<code>capwords(s)</code>	Copy of <code>s</code> ; first character of each word capitalized
<code>center(s, width)</code>	Center <code>s</code> in a field of given <code>width</code>
<code>count(s, sub)</code>	Count the number of occurrences of <code>sub</code> in <code>s</code>
<code>find(s, sub)</code>	Find the first position where <code>sub</code> occurs in <code>s</code>
<code>join(list)</code>	Concatenate <code>list</code> of strings into one large string
<code>ljust(s, width)</code>	Like <code>center</code> , but <code>s</code> is left-justified
<code>lower(s)</code>	Copy of <code>s</code> in all lowercase characters
<code>lstrip(s)</code>	Copy of <code>s</code> with leading whitespace removed
<code>replace(s, oldsub, newsub)</code>	Replace occurrences of <code>oldsub</code> in <code>s</code> with <code>newsub</code>
<code>rfind(s, sub)</code>	Like <code>find</code> , but returns the rightmost position
<code>rjust(s, width)</code>	Like <code>center</code> , but <code>s</code> is right-justified
<code>rstrip(s)</code>	Copy of <code>s</code> with trailing whitespace removed
<code>split(s)</code>	Split <code>s</code> into a list of substrings (see text)
<code>upper(s)</code>	Copy of <code>s</code> ; all characters converted to uppercase

Table 4.2: Some components of the Python string library

Searching a String

- We use the `find()` function to search for a substring within another string
- `find()` finds the first occurrence of the substring
- If the substring is not found, `find()` returns `-1`
- Remember that string position starts at zero



```
>>> import string
>>> greet = "Hello Bob"
>>> pos = string.find(greet,"o")
>>> print pos
4
>>> aa = string.find(greet,"z")
>>> print aa
-1
```

Making everything UPPER CASE

- You can make a copy of a string in **lower case** or **upper case**
- Often when we are searching for a string using `find()` - we first convert the string to lower case so we can find a string regardless of case

```
>>> import string
>>> greet = "Hello Bob"
>>> nnn = string.upper(greet)
>>> print nnn
HELLO BOB
>>> lll = string.lower(greet)
>>> print lll
hello bob
>>>
```

Search and Replace

- The `replace()` function is like a “search and replace” operation in a word processor
- It replaces **all occurrences** of the **search string** with the **replacement string**

```
>>> import string
>>> greet = "Hello Bob"
>>> nstr = string.replace(greet, "Bob", "Jane")
>>> print nstr
Hello Jane
>>> greet = "Hello Bob"
>>> nstr = string.replace(greet, "o", "X")
>>> print nstrHellX BXB
>>>
```

Stripping Whitespace

- Sometimes we want to take a string and remove whitespace at the beginning and/or end
- `lstrip()` and `rstrip()` to the left and right only
- `strip()` Removes both begin and ending whitespace

```
>>> import string
>>> greet = " Hello Bob "
>>> string.lstrip(greet)
'Hello Bob '
>>> string.rstrip(greet)
' Hello Bob'
>>> string.strip(greet)
'Hello Bob'
>>>
```


Breaking Strings into Parts

- We are often presented with input that we need to break into pieces
- We use the `split()` function to break a string into a **sequence of strings**

```
>>> import string
>>> abc = "With three words"
>>> stuff = string.split(abc)
>>> print stuff
['With', 'three', 'words']
>>>
```

```
>>> import string
>>> abc = "With three words"
>>> stuff = string.split(abc)
>>> print stuff
['With', 'three', 'words']
>>> print len(stuff)
3
>>> print stuff[1]
three
```

```
>>> print stuff
['With', 'three', 'words']
>>> for w in stuff:
...     print w
...
With
three
words
>>>
```

Split breaks a string into parts produces a list of strings. We think of these as words. We can access a particular word or loop through all the words.

```
>>> import string
>>> line = "first,second,third"
>>> thing = string.split(line)
>>> print thing
['first,second,third']
>>> print len(thing)
1
>>> thing = string.split(line, ",")
>>> print thing
['first', 'second', 'third']
>>>
```

```
>>> line = "A lot      of spaces"
>>> etc = line.split()
>>> print etc
['A', 'lot', 'of', 'spaces']
>>>
```

You can specify what **delimiter** character to use in the splitting.

Also when you do not specify a delimiter, multiple spaces is thought of as “one” delimiter.

You can also just add **.split()** to the end of a string variable.

File Processing

File Processing

- A text file can be thought of as a sequence of lines

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

Return-Path: <postmaster@collab.sakaiproject.org>

Date: Sat, 5 Jan 2008 09:12:18 -0500 To: source@collab.sakaiproject.org From:
stephen.marquard@uct.ac.za Subject: [sakai] svn commit: r39772 -
content/branches/Details: [http://source.sakaiproject.org/viewsvn/?
view=rev&rev=39772](http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772)

Opening a File

- Before we can read the contents of the file we must tell Python which file we are going to work with and what we will be doing with the file
- This is done with the `open()` function
- `open()` returns a “file handle” - a variable used to perform operations on the file
- Kind of like “File -> Open” in a Word Processor

Using open()

- `handle = open(filename, mode)` `fhand = open("mbox.txt", "r")`
- returns a handle use to manipulate the file
- filename is a string
- mode is “r” if we are planning reading the file and “w” if we are going to write to the file.

File Handle as a Sequence

- A **file handle** open for read can be treated as a **sequence** of strings where each line in the file is a string in the sequence
- We can use the **for statement** to iterate through a **sequence**
- Remember - a **sequence** is an ordered set

```
xfile = open("mbox.txt", "r")
```

```
for cheese in xfile:  
    print cheese
```


Counting Lines in a File

- Open a file read-only
- Use a for loop to read each line
- Count the lines and print out the number of lines

```
pizza = open("mbox.txt", "r")  
  
howmany = 0  
for slice in pizza:  
    howmany = howmany + 1  
  
print howmany
```

Summary

- String Data Type
- `input()` and `raw_input()`
- Indexing strings
- Slicing strings
- String operators
- String `len()` function
- Looping through a string
- String Library
- Searching strings
- Changing Case
- Removing Whitespace
- Splitting a string into parts
- File Processing
- Opening a File
- Looping through a file