STRINGS (CHAPTER 4)

Photo from RCReptiles.com

LEARNING OUTCOMES

- To understand the string data type and how strings are represented in the computer.
- To be familiar with the various operations that can be performed in strings through built-in functions and the string library.
- To understand basic file processing concepts and techniques for reading and writing text files in Python.
- To be able to understand and write programs that process textual information.

String

- String (noun)
- A cord usually made of fiber, used for fastening or tying.
- Something configured as a long, thin line.
- A set of objects threaded together: a string of beads.
- A series of similar or related acts, events, or items arranged or falling in or as if in a line.
- (Computer Science) A set of consecutive characters.

Strings and characters

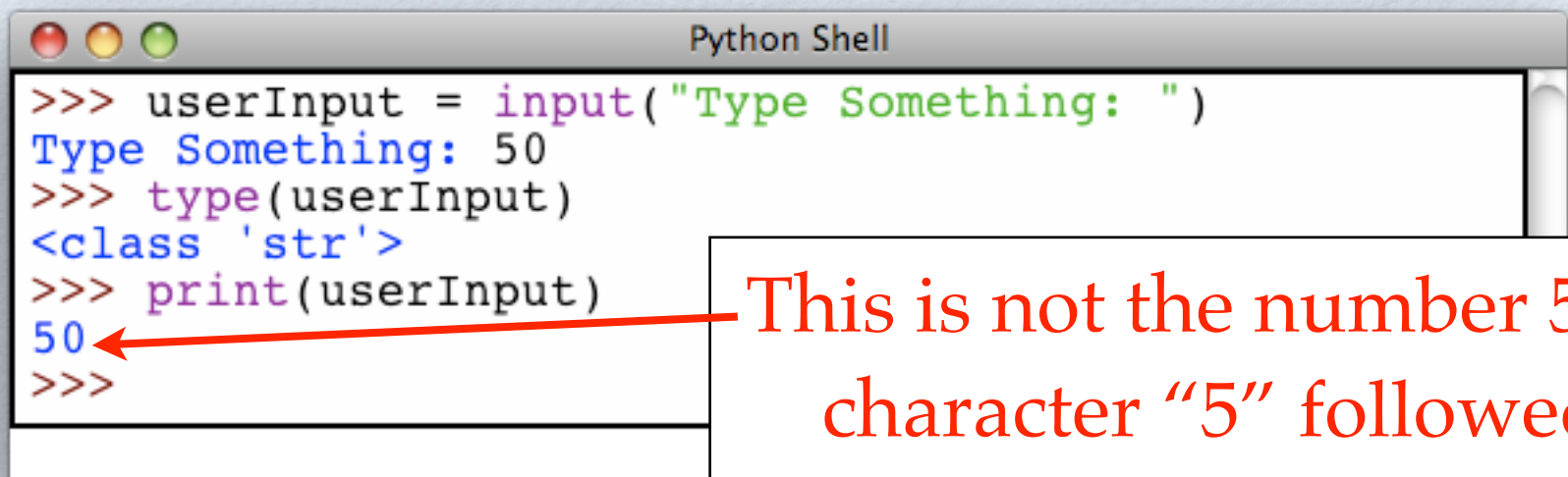
- A character is something that you can type in on the keyboard.
  - For example, the letter "`a`", or the digit "`1`"
- A string is a sequence of characters.
  - For example, the string "hello" is the sequence of characters "h", "e", "l", "l", "o"
- All input from the keyboard is done with character sequences.
- All textual output is done with character sequences.
- Therefore, strings are one of the most often-used data types.

Getting Input From the Keyboard

- You remember that the input() function is used to get user input from the keyboard.

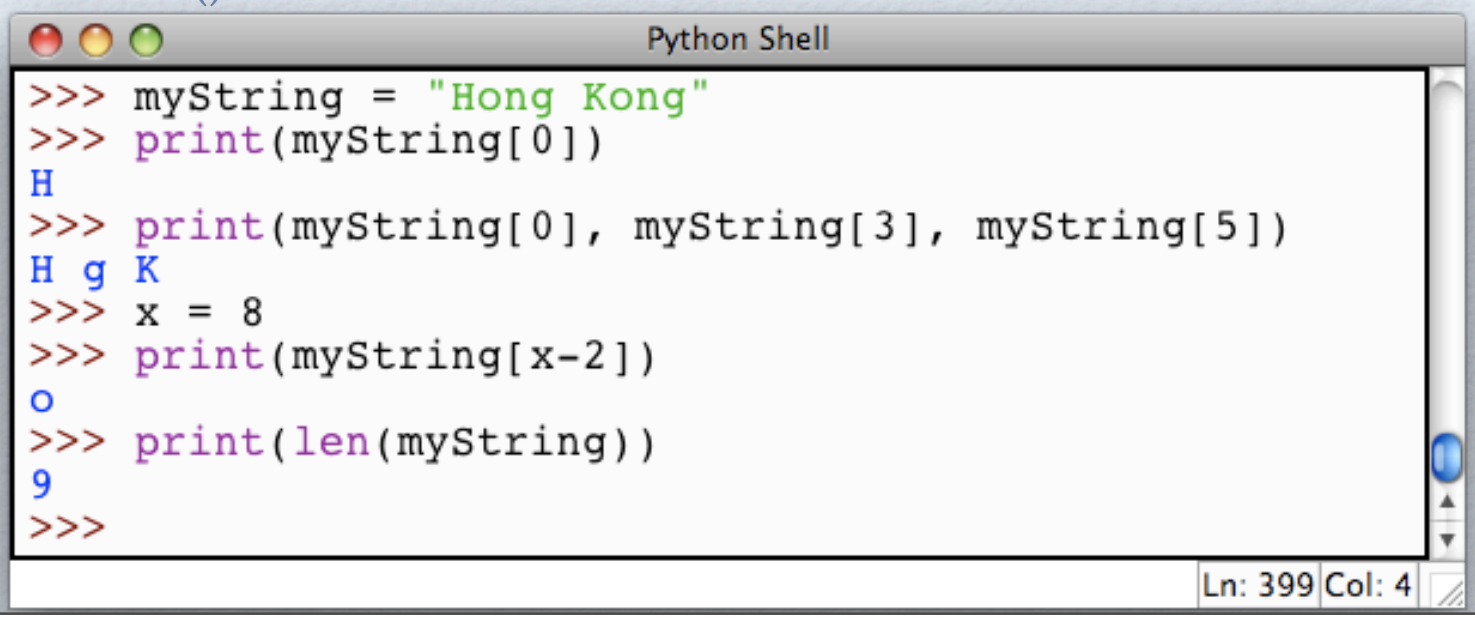
```
userInput = input("Type something: ")
```

- input() always treats the user input as a string



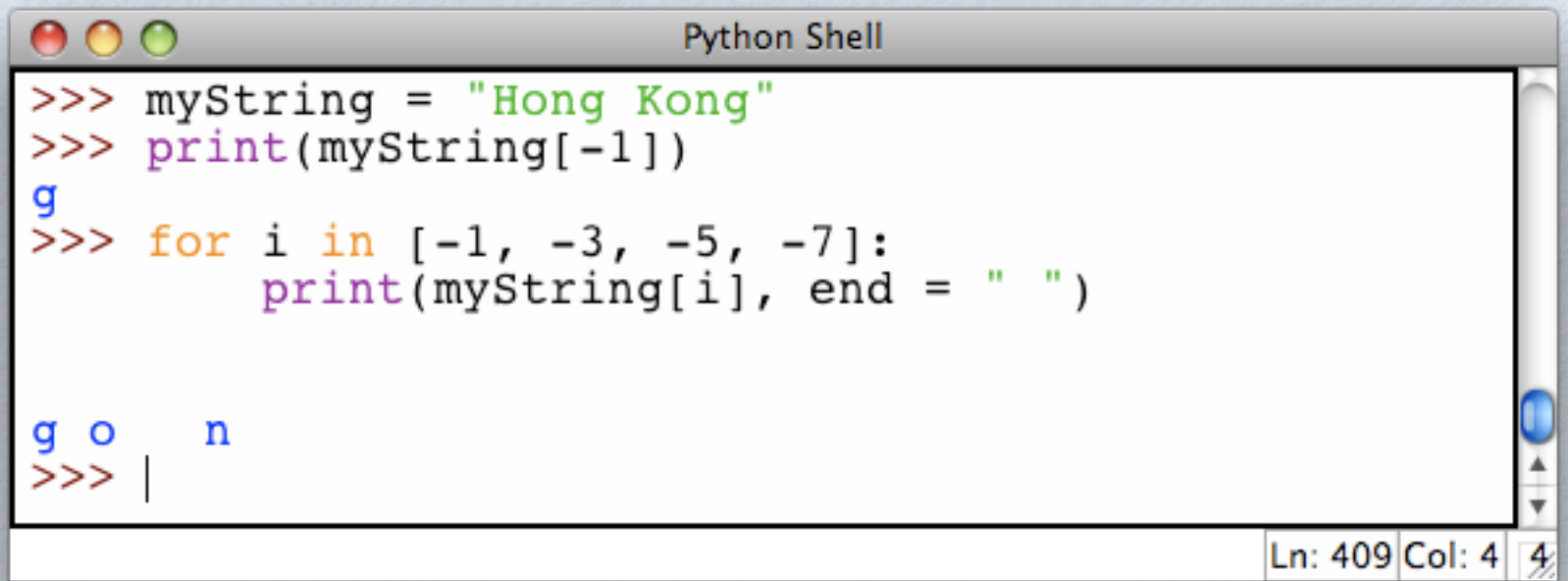
INDEXING STRINGS

- Python allows us to get individual characters out of a string.
- We write the name of the variable storing the string, then a pair of square brackets, with the **index** of the character we want between the brackets.
- The first character has index 0.
- The length of a string (number of characters) can be obtained with the function len()



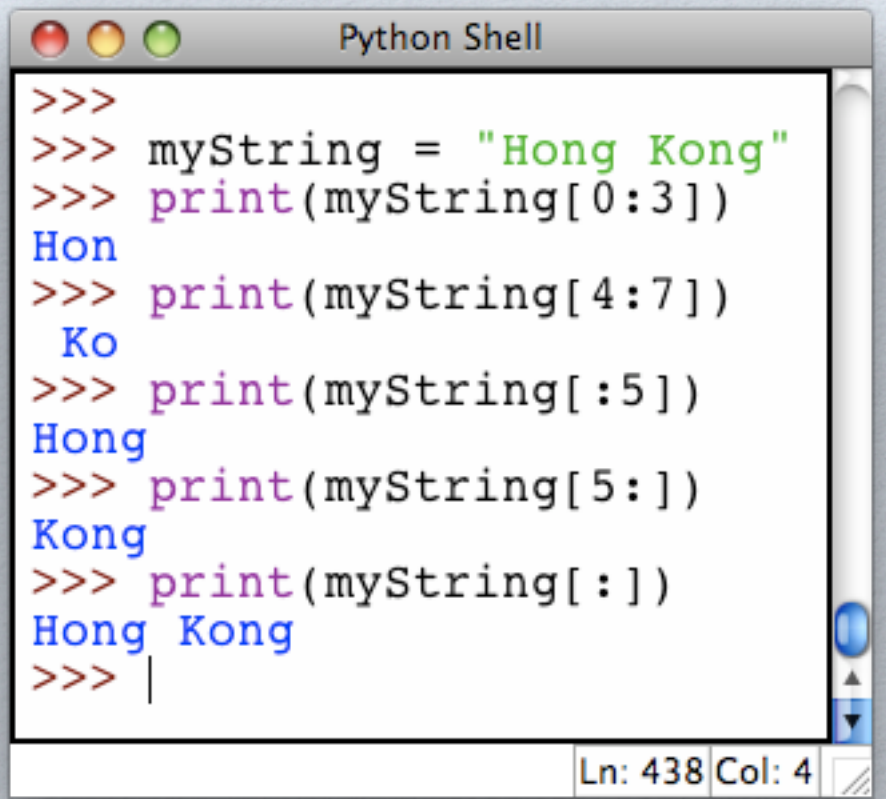
More Indexing Strings

- Unlike many other programming languages, Python also allows indexing from the right-hand-end of a string.
- We use **negative indices** to show that we are counting from the right.
- The rightmost character has a index of -1.



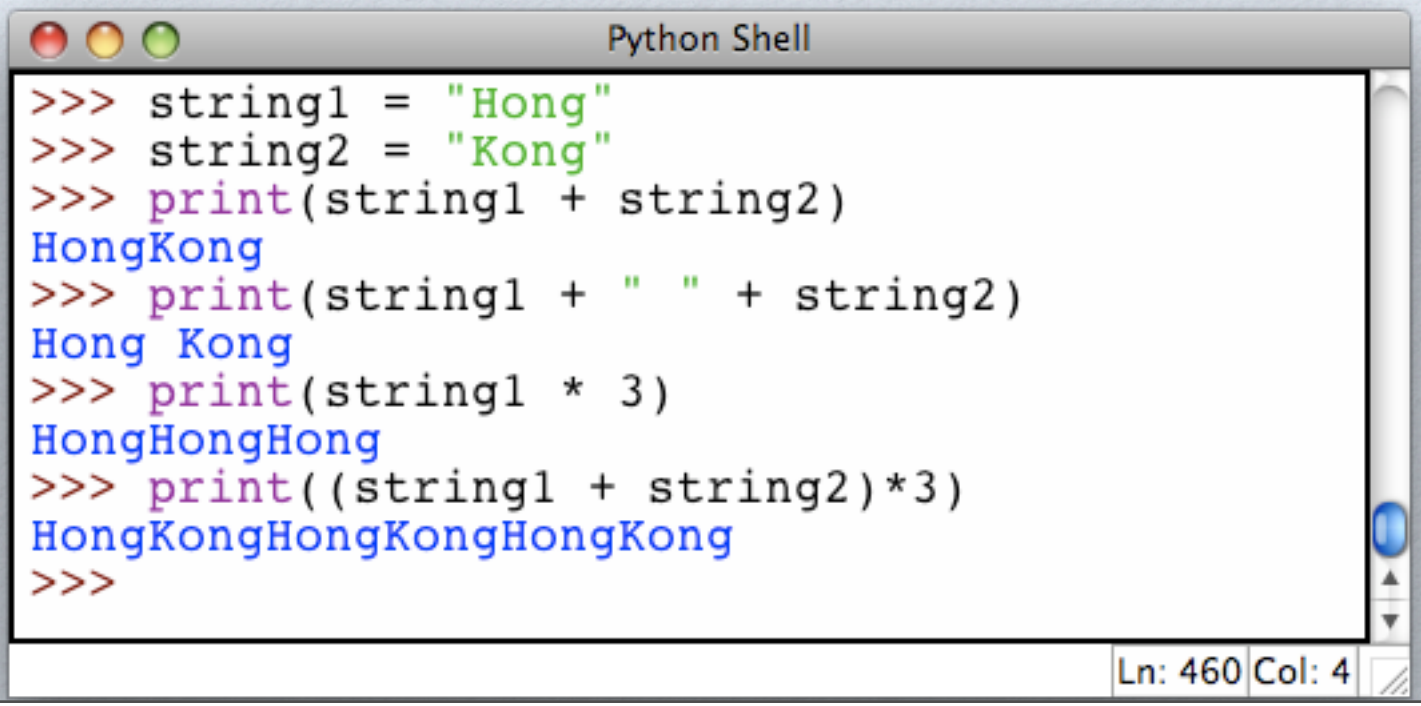
Getting Substrings

- Python allows us to **slice** a string.
- A slice of a string is a **contiguous (continuous) sequence of characters** from the string.
- It is also called a **substring**.



PUTTING STRINGS TOGETHER

- Python also lets us add strings together and multiply a string by a number.
- Adding strings together **concatenates** the string.
- Multiplying a string with a number **repeats** the string.

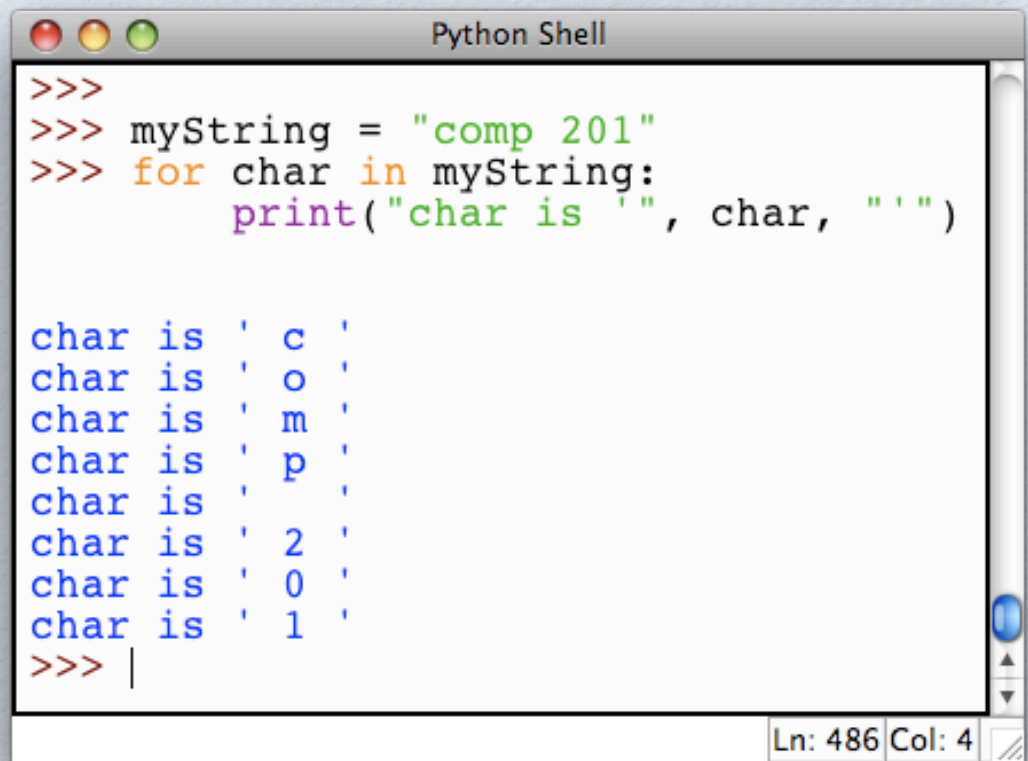


A LIST OF CHARACTERS

- In mathematics, a **list** is an **ordered collection** of items.
- A string in Python is a collection of characters.
- The collection has an order (obviously, since we would all agree that “hello” does not equal to “lehlo”)
- Therefore, a Python string is a list of characters.
- The sequences that we used in our `for` loops are also lists.

Iterating over a string

- Since a string is a list of characters, we can also use a for loop to iterate over a string.
- When we used a list of numbers, the for loop went over each number in turn.
- With a string, the for loop will go over each character in the string in turn.



String Operations Recap

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

EXAMPLE

- Suppose that we would like to print out the abbreviation of a month that corresponds to a certain month number.
- For example,
  - The number 1 would print out "Jan"
  - The number 2 would print out "Feb"
- Also suppose that we have the following string:

```
months = "JanFebMarAprMayJunJulAugSepOctNovDec"
```

- How would we solve this problem?

WRITING THE ALGORITHM

- Our problem looks quite complicated, but it is well within what we have learned so far.
- Algorithm:

  1. Construct the string containing the month abbreviations.
  2. Get the number of the required month from the user.
  3. Compute starting position of the month from the month number.
  4. Get the corresponding 3-letter substring.
  5. Print the result.

CHECK THE STRING

- For each month number, write down the character positions of the string that corresponds to it.

Month	Starting Position	Ending Position	Month	Starting Position	Ending Position
1			7		
2			8		
3			9		
4			10		
5			11		
6			12		

Slicing the String

- Assume that the number of the month is stored in the variable `monthNumber`.
- Given the starting positions that we figured out, write down an equation that would give you the appropriate starting position for the corresponding month.

```
pos =
```

WRITING THE PROGRAM

```
# MonthAbbreviation.py

def main():
      # string containing all the abbreviations
      months = "JanFebMarAprMayJunJulAugSepOctNovDec"

      # Get month number from user
      userInput =
      monthNumber =

      # calculate starting position of substring
      pos =

      # Get the appropriate slice from the big string
      monthAbbrev = months[                    ]

      print("The abbreviation you want is", monthAbbrev)

main()
```

Python Data Types

- So far, we have seen four types of data that a computer can process:
  - Integers: whole numbers. e.g. `5`, `11`, `-1`, `0`
  - Floats: numbers with decimals, e.g. `3.5`, `11.0`, `-5.923`, `3.1416`
  - Characters: stuff that we can type in on the keyboard
  - Strings: sequences of characters, e.g. `"hello"`, `"12345"`

STORING DATA

- Recall: Variables are **storage places** in a computer's memory for data.
- Recall: To store data inside a variable, we **assign** the variable the value of the data.
- Recall: To get the data stored inside a variable, we **retrieve** the data from the variable.
- For students with prior programming background: unlike some other programming languages, Python allows a variable to store any kind of data.

Data types and conversions

- To find out what type of data a particular variable is storing, we can use the `type()` function.

```
>>> type(5)
<class 'int'>
>>> type ("05")
<class 'str'>
```

- To convert one kind of data to another:
  - `int()`: turns a value into a whole number. Can be used to convert a string into a number, or can be used to round down a decimal number.
  - `float()`: turns a value into a decimal number. Can be used to convert a string into a number.
  - `str()`: turns a number into a string.

Representing Data

- Recall: At the bottommost level (i.e. the electronics), everything is a bunch of wires, that can only be switched on and off.
- Therefore, at the lowest level, everything in a computer is represented as a bunch of 1s and 0s.
- As a result, all numbers are represented in the computer by their binary equivalents.
- e.g. 2 is actually represented in the computer as 10
- 55 is represented in the computer as 110111
- (You'll learn more about this in COMP 212)

Character Representation

- Binary works very well for numbers, but what about for characters?
- Turns out that there's an international standard called the **ASCII Code** that maps each alphanumeric character to a number.
- For example, the character A is stored as the number 65 (or more precisely, as its binary equivalent 1000001.
- All characters have their ASCII code. The character + is represented as the number 43, and the character a as 97.
- So what number corresponds to what character?

THE ASCII TABLE

The 7-bit ASCII Table										
	0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1	NL	VT	NP	CR	SO	SI	DLE	DC1	DC2	DC3
2	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3	RS	US	SP	!	“	#	$	%	&	‘
4	(	)	*	+	,	-	.	/	**0**	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	**A**	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	**a**	b	c
10	d	e	f	g	h	I	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	DEL		

Character Ranges

- The ASCII code for each alphanumeric character is fairly easy to remember if you remember the range:

Lowercase English	‘a’	‘b’	‘c’	…	‘z’
	97	98	99	…	112
Uppercase English	‘A’	‘B’	‘C’	…	‘Z’
	65	66	67	…	90
Digits	‘0’	‘1’	‘2’	…	‘9’
	48	49	50	…	57
Other symbols	‘&’	‘=‘	‘+’	…	
	38	42	43	…	

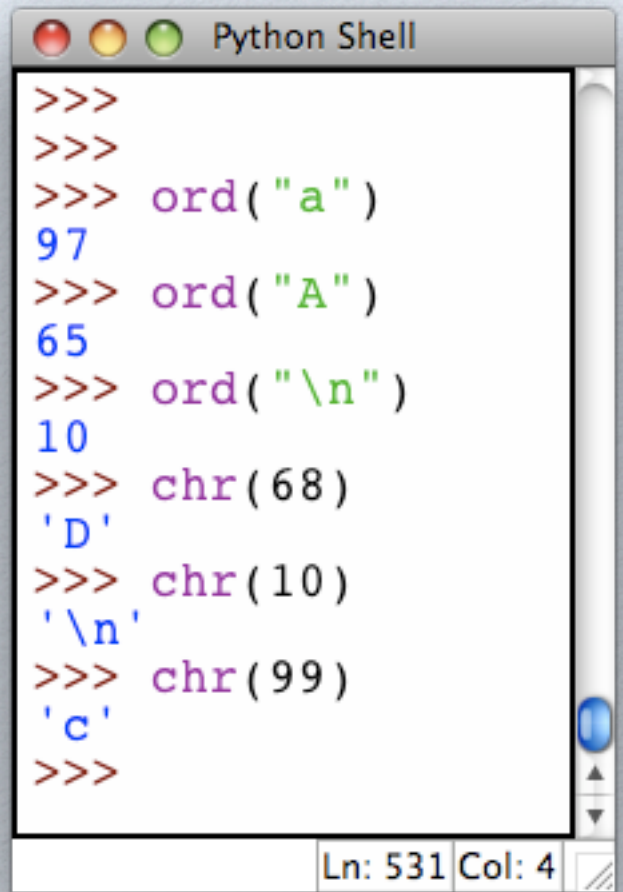
SPECIAL CHARACTERS

- Some of the ASCII table characters cannot be typed that easily.

Name of character	In Python	ASCII value
backslash	'\\'	92
backspace	'\b'	8
horizontal tab	'\t'	9
null	'\0'	0
newline	'\n'	10
…	…	…

Character Conversion

- Inside Python, we can use the `ord()` (ordinal) function to get the ASCII code of a character.
- The `chr()` function does the reverse (turns an ASCII number back to its corresponding character.)



EXERCISE

- Write a function, `encode()`, that will ask the user for a string, and then encode it by representing each character by its ASCII code equivalent.

```
>>> encode("Hello!")
72 101 108 108 111 33
```

More String Operations

- Python provides a number of other useful operations for strings:

Function	Meaning
`s.capitalize()`	Makes a copy of `s` with first character capitalized
`s.upper()`	Makes a copy of `s` with all characters capitalized
`s.lower()`	Makes a copy of `s` with all characters lower cased
`s.split()`	Split `s` into a list of substrings
`s.count(sub)`	Counts number of occurrences of `sub` in `s`

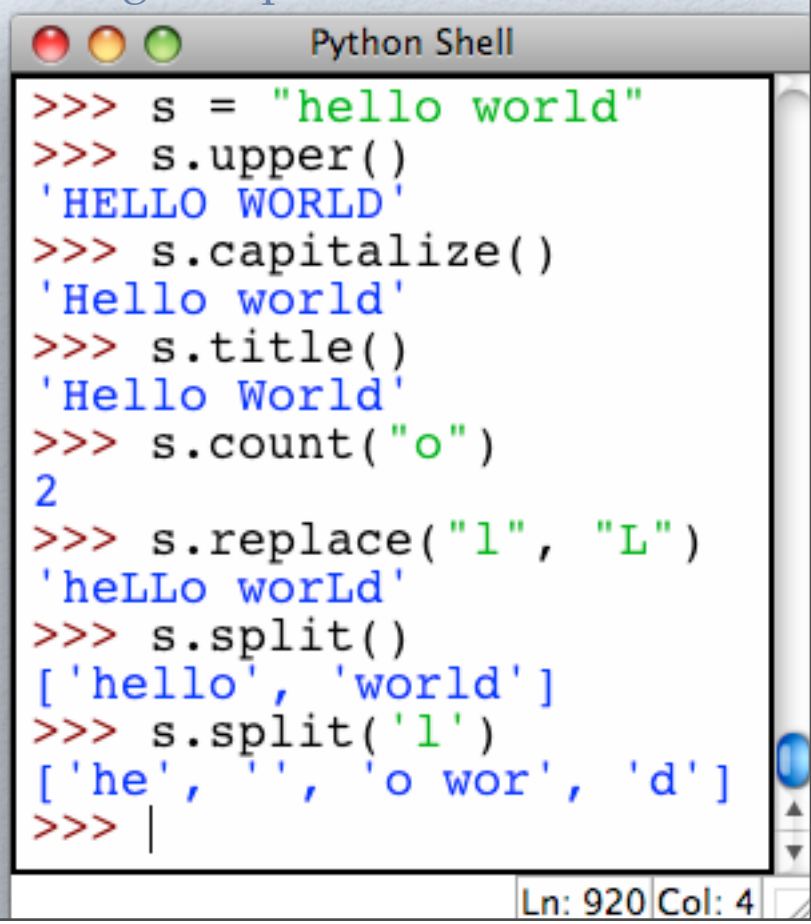
- Note: these functions do not change `s`. They make a copy of `s` and apply the changes to that instead.
- More functions here: http://www.python.org/doc/2.5.2/lib/string-methods.html

String operations examples

- Note the syntax of the operations:

```
string.function(parameters)
```

- It's like we're giving a command to the string and asking it to perform the function.
  - e.g. s.upper() -- we're giving a command: "string s, upper()!"
  - compared with eval(s) -- where we're asking Python: to turn the string s into a number.
- The string is an **object** and the function is a **method** that it has to perform.
- This is the concept of **object-based programming**, which we will see later in the semester.



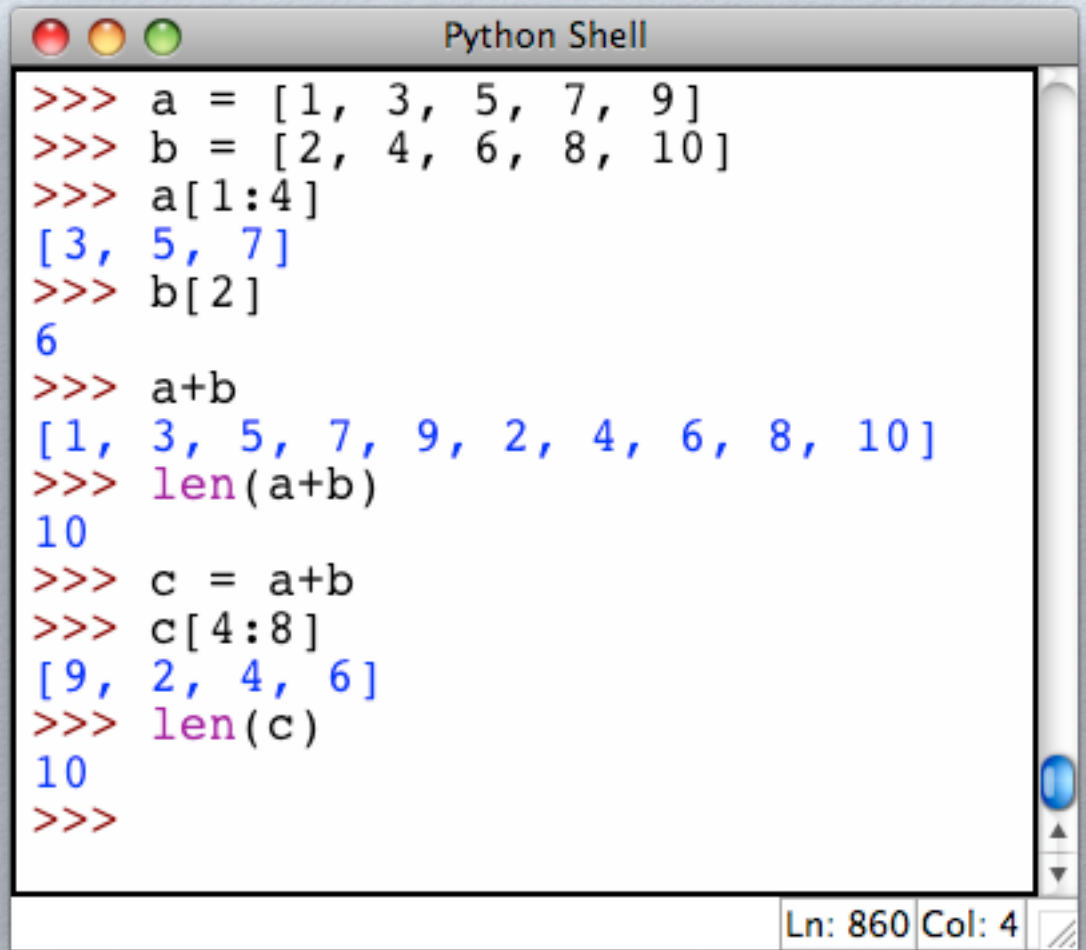
Lists, Sequences and Strings

- Recall: A Python string is just a list of characters.
- The sequences that we used in our `for` loops are also lists.
- Therefore, lists in Python can handle both strings and numbers.
- Unlike some other programming languages, Python allows us to mix it up and create lists that contain more than one kind of data.

```
myList = [1, "Hong", 5, "Kong"]
```

STORING LISTS

- Just like variables can store strings, variables can also store lists.
- And we can also index, concatenate, repeat and slice lists, just like they were strings.
- The concatenation of two lists and the repetition of a list is also a list.



EXERCISE

- Now that we know more about lists, let's rewrite our month abbreviation program.
- Problem: we would like to print out the abbreviation of a month that corresponds to a certain month number.
- This time, we'll use a list to store the month abbreviations:

```
months = ["Jan", "Feb", "Mar", "Apr", "May",
"Jun", "Jul", "Sep", "Oct", "Nov", "Dec"]
```

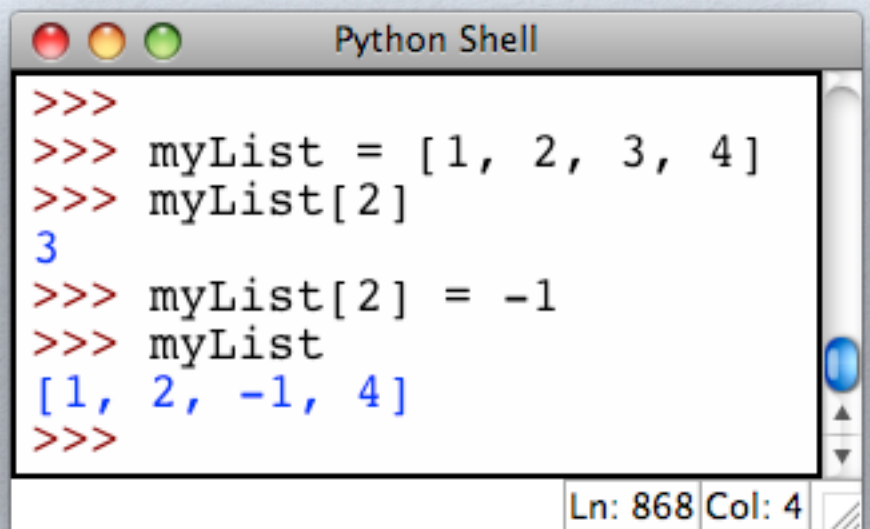
Our New Algorithm

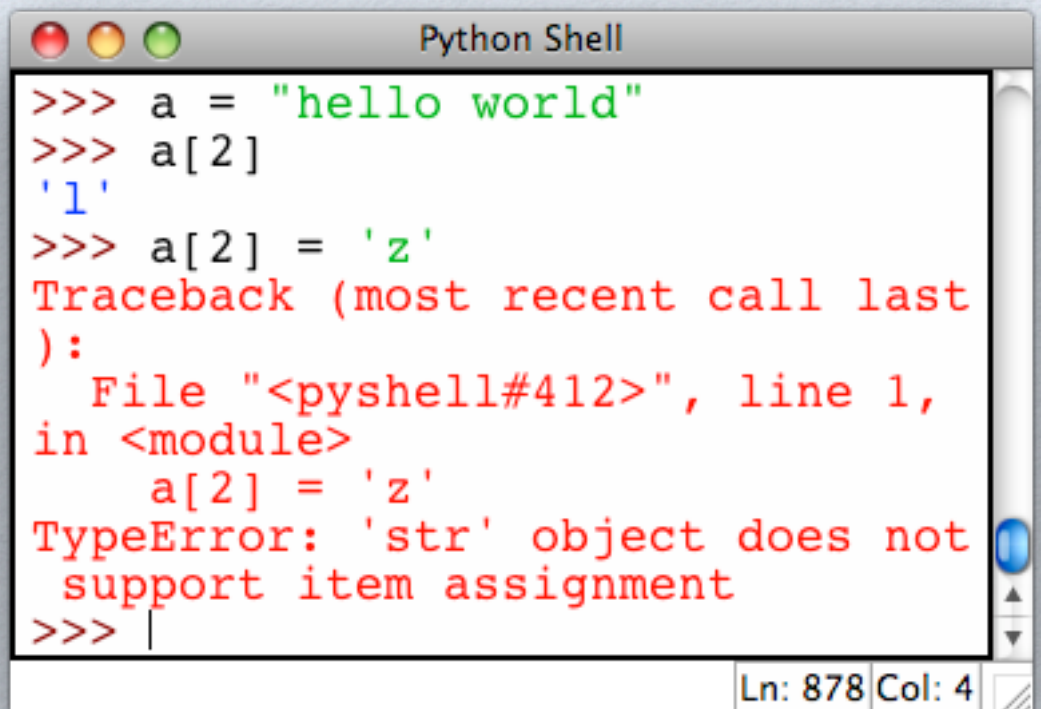
- Using the list makes our problem a lot simpler.
- All that we need to do is to get the month number from the user, and then index into the `months` list.
- Can you write the algorithm?

My Algorithm:

Strings and Lists

- Now we know strings and lists, and we know the similarities between them.
- However, there are some differences:
  - The `capitalize()`, `split()`, `lower()`, etc functions are for strings only.
  - Items in a list can be changed, characters in a string cannot.





EXERCISE

- Earlier, we wrote a function called encode(), which would take a string and encode each character into its ASCII code.
- Write a function called decode(), that will take the encoded string and give you back the original.

```
>>> decode("72 101 108 108 111 33")
Hello!
```

Some hints

- We will read in the list of ASCII codes as a string of space-separated numbers.
- We can `split()` the string to get a list of substrings, each containing one number.
- We can then `eval()` each substring to get the number, which should be the ASCII code.
- Can you write the algorithm?

My Algorithm

File input and output

- So far, all of our input and output has been done via the keyboard and the monitor.
- However, a lot of applications need to deal with data that is stored into files:
  - e.g. word processing, spreadsheets, drawing programs, webpages, etc.
- For COMP 201, we will concern ourselves with text files (files that humans can read), rather than binary files (such as photos)

READING IN A FILE

- Before we read in a file, we need to **open** it.

```
filename = "myFile.txt"
infile = open(filename, "r")
```

- The variable infile is called a **filehandle**. It represents the file inside the Python program.
- The string "r" after the name of the file tells Python that we are going to **read** the file.
- If the file does not exist, Python will give us an error.

READING IN A FILE

- Once the file is opened, Python offers a loop-like structure for reading in a text file one line at a time:

```
for line in infile:
    <variable line now contains one line>
```

- Once the line is read in, we can manipulate it just like it was an ordinary string.

```
    words = line.split() # whatever we need to do
```

- When all the lines have been read in, Python will exit the loop automatically.

- We can then **close** the file.

```
infile.close()
```

Writing to a File

- If we want to save something to a file, we say that we need to **write** to the file.
- Before we write to the file, we also need to **open** it.

```
filename = "myOutputFile.txt"
outfile = open(filename, "w")
```

- The "w" after the filename indicates to Python that we want to open the file and write to it.
- If the file does not yet exist, Python will create an empty file for us.
- If the file already exists, Python will erase everything in it and give us an empty file.

Writing to a file

- When the file has been opened, we can write to it.
- Writing to a file is similar to printing to the screen.

```
print("hello, world", file=outfile)
```

- The file= at the end of the statement indicates to Python that we want to print to a file instead of to the screen.
- When we are done writing, then we need to close the file.

```
outfile.close()
```

EXERCISE

- Suppose that you have a text file containing the data of students.
- The format of the file is:

```
name studentID GPA age
```

  with a space in between each field.

- We want to read in all the data and calculate the average GPA of all the students.

YOUR ALGORITHM

- You should have all the knowledge that you need to complete this program.
- Hints: Each line is read in from the file as one single string.
- The format of the file is fixed (i.e. the fields are always separated by spaces, the 3rd field is always the GPA)

My Algorithm:

You now know...

- What strings are, and how they are stored in the computer.
- How to work with strings through basic functions and the string library.
- How to read from text files and output to text files in Python.
- How to process textual information in Python