ISE 314X
Computer Programing for Engineers

Chapter 5
Sequences: Strings, Lists, and Files

Yong Wang
Assistant Professor
Systems Science & Industrial Engineering
Binghamton University

Objectives

- To understand the string and list data types
- To understand the basic idea of indexing
- To understand how to read and write text files

The String Data Type

- One most common use of computers is word processing
- Text is represented in programs by the *string* data type
- A string is a sequence of characters enclosed within quotation marks " " or ' '

The String Data Type

```
>>> str1 = "Hello,"
>>> str2 = 'world!'
>>> print(str1, str2)
Hello, world!
>>> type(str1)
<class 'str'>
```

The String Data Type

- Getting a string as input

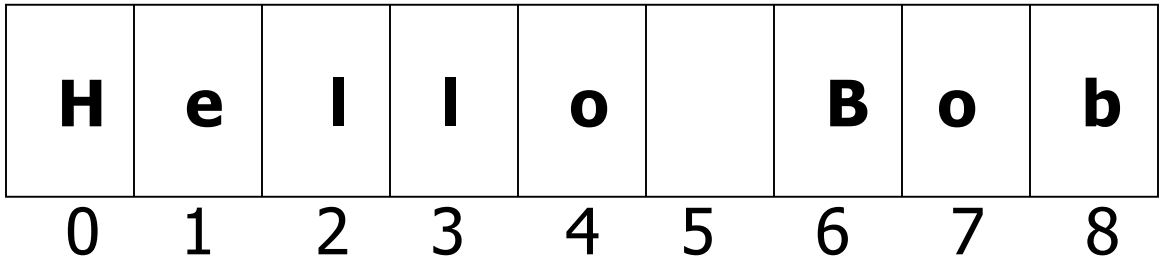
```
>>> name = input("Enter your name: ")
Please enter your name: C-3PO
>>> name
'C-3PO'
>>> print("Hello,", name)
Hello, C-3PO
```

- Note that `input` is not `eval`uated

The String Data Type

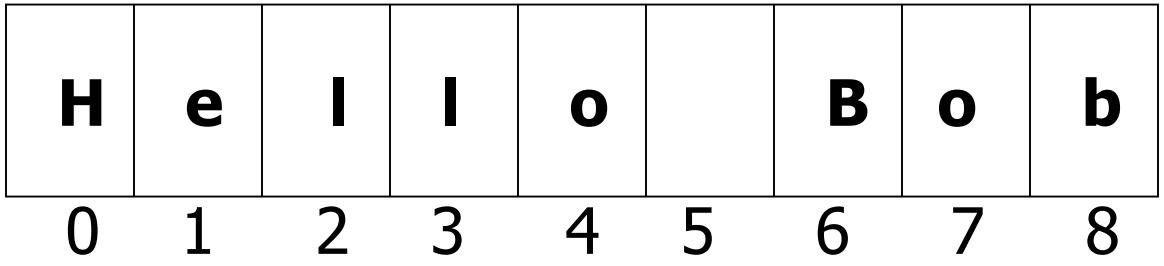
- We can access the individual characters in a string through *indexing*
- The positions in a string are numbered from the left, starting with 0

The String Data Type



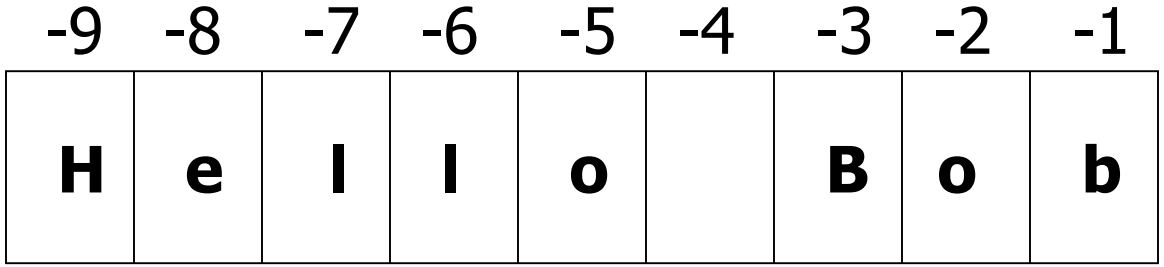

```
>>> greet = "Hello Bob"
>>> greet[0]
'H'
>>> print(greet[0], greet[2], greet[4])
H l o
>>> x = 8
>>> print(greet[x-2])
B
```

The String Data Type



- In a string of *n* characters, the last character is at position *n-1* since we start counting with 0

The String Data Type



- We can index from the right side using negative indexes

```
>>> greet[-1]
'b'
>>> greet[-3]
'B'
```

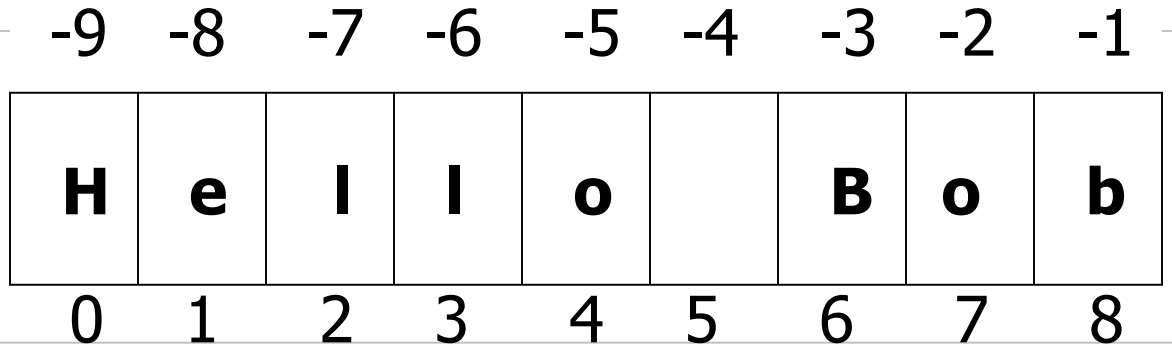
The String Data Type

- We can also access a continuous sequence of characters, called a *substring*, through a process called *slicing*

The String Data Type

- Slicing: <string>[<start>:<end>]
- start and end should both be ints
- The slice contains the substring beginning at position start and runs up to but doesn't include the position end

The String Data Type




```
>>> greet[5:100]
' Bob'
>>> greet[:5]
'Hello'
>>> greet[5:]
' Bob'
>>> greet[:]
'Hello Bob'
```

What about these?

```
>>> greet[-8:-5]
>>> greet[-3:]
>>> greet[:-6]
>>> greet[2:-6]
>>> greet[5:1]
>>> greet[2:2]
```

String Operations

- *Concatenation* “glues” two strings together (+)
- *Repetition* builds up a string by multiple concatenations of a string with itself (*)

String Operations

```
>>> "spam" + "eggs"
'spameggs'
>>> 3 * "spam"
'spamspamspam'
>>> (2 * "spam") + ("eggs" * 3)
'spamspameggseggseggs'
```

String Operations

- The function *len* returns the length of a string

```
>>> len("spam")
4
>>> for ch in "Spam":
...     print(ch, end=" ")  #try print(ch)

S p a m
```

String Operations

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

Simple String Processing

- Create a username using first initial and first seven characters of last name

```
# username.py
# Get user's first and last names
first = input("Enter your first name: ")
last = input("Enter your last name: ")

# concatenate first initial with
# 7 chars of last name
uname = first[0] + last[:7]
print("Your username is:",uname)
```

Simple String Processing

- Run the program in IDLE:

```
Enter your first name: taylor
Enter your last name: swift
Your username is: tswift
```

- Another try

```
Enter your first name: arnold
Enter your last name: schwarzenegger
Your username is: aschwarz
```

Exercise

- **Modify this program to use the main function**

```
# username.py
# Get user's first and last names
first = input("Enter your first name: ")
last = input("Enter your last name: ")

# concatenate first initial with
# 7 chars of last name
uname = first[0] + last[:7]
print("Your username is:",uname)
```

Strings, Lists, and Sequences

- *A list* is a sequence of data enclosed with brackets

```
>>> [1, 2, 5, -3]
[1, 2, 5, -3]
```

Strings, Lists, and Sequences

- These operations also apply to lists

```
>>> [1,2] + [1,4]
[1, 2, 1, 4]
>>> [1,2]*3
[1, 2, 1, 2, 1, 2]
>>> grade = ['A', 'BB', 'C']
>>> grade[0]
'A'
```

```
>>> grade[1:]
['BB', 'C']
>>> len(grade)
3
>>> type(grade)
<class 'list'>
```

Strings, Lists, and Sequences

- Strings are always sequences of characters, but *lists* can be sequences of arbitrary values

```
>>> nestedList = [1, "Spam", 3.4, [7, 'inner']]
>>> nestedList
[1, 'Spam', 3.4, [7, 'inner']]
>>> len(nestedList)
4
```

Strings, Lists, and Sequences

- Convert an int month into the three-letter abbreviation

```
>>> months = ["Jan", "Feb", "Mar", "Apr",
... "May", "Jun", "Jul", "Aug", "Sep", "Oct",
... "Nov", "Dec"]
>>> months[3]
'Apr'
```

- To get month *n* out of the sequence, do this

```
monthAbbr = months[n-1]
```

Strings, Lists, and Sequences

```
# month_conv.py
def main():
    months = ["Jan", "Feb", "Mar", "Apr",
              "May", "Jun",  "Jul", "Aug",
              "Sep", "Oct", "Nov", "Dec"]
    n = eval(input("Enter a month number between 1 and 12: "))

    print("The month abbreviation is", months[n-1] + ".")

main()
```

- **Can we change the indentation in the list?**

Strings and Secret Codes

- In the computer, a character is stored as a number
- ASCII (American Standard Code for Information Interchange) table stores 128 characters
- Unicode can store 100,000+ characters

Strings and Secret Codes

- The *ord* function converts a single character to the numeric (ordinal) code
- The *chr* function converts a numeric code to the corresponding character

```
>>> ord("A")
65
>>> ord("a")
97
>>> chr(97)
'a'
>>> chr(65)
'A'
```

Strings and Secret Codes

- The encoding algorithm pseudocode

```
get the text message
for each character in the message:
    print the ordinal number of the character
```

Strings and Secret Codes

```
# text2num.py
def main():
    print("This program converts texts to numbers.")
    message = input("Enter the text: ")
    print("Here are the Unicode codes:")
    for ch in message:
        print(ord(ch),  end=" ")

main()
```

Strings and Secret Codes

- Run the program:

```
This program converts texts to numbers.
Enter the text: The Matrix
Here are the Unicode codes:
84 104 101 32 77 97 116 114 105 120
```

Strings and Secret Codes

- Now we need a program to decode the message
- The pseudocode for a decoder:

```
get the sequence of encoded numbers
message = ""
for each number in the input:
    convert the number to a character
    add the character to the end of message
print the message
```

Strings and Secret Codes

- The method `split` will split a string into substrings based on spaces

```
>>> "Hello string methods!".split()
['Hello', 'string', 'methods!']

>>> str1 = "Hello string methods!"
>>> str1.split()
['Hello', 'string', 'methods!']
```

Strings and Secret Codes

- The *separator* can be other characters

```
>>> "32,24,25,57".split()
['32,24,25,57']
>>> "32,24,25,57".split(",")
['32', '24', '25', '57']
```

Strings and Secret Codes

- Use *eval* to convert a string into a number

```
>>> numStr = "32"
>>> eval(numStr)
32
>>> x = eval(input("Enter a number: "))
Enter a number: 3.14
>>> x
3.14
```

Strings and Secret Codes

```
# num2text.py
def main():
    print("This prog converts encoded numbers into texts.")
    inString = input("Enter the encoded message: ")
    message = ""
    for numStr in inString.split():
        # convert the (sub)string to a number
        codeNum = eval(numStr)
        # append character to message
        message = message + chr(codeNum)
    print("The decoded message is:", message)

main()
```

Strings and Secret Codes

- Run the program:

```
This program converts numbers into texts.
Enter the encoded message: 84 104 101 32 77 97
  116 114 105 120
The decoded message is: The Matrix
```

Exercise

- **Modify this program so it can take care of the following inputs:**

84; 104; 101; 32; 77; 97; 116; 114; 105; 120

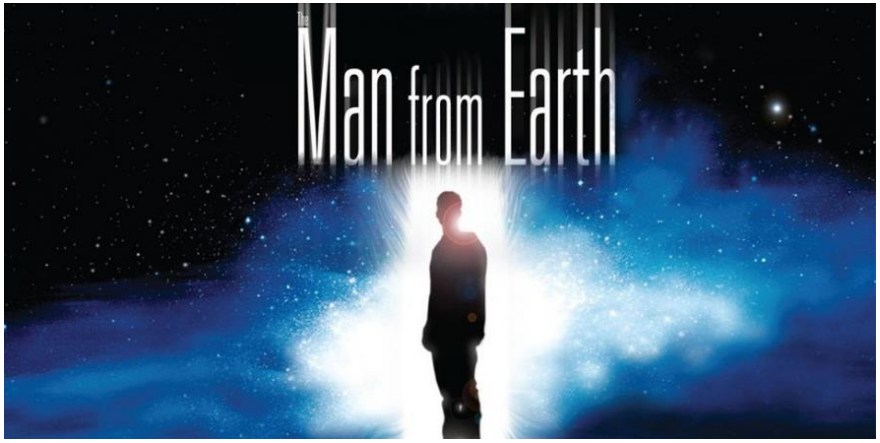
```
# num2text.py
def main():
    print("This prog converts encoded numbers into texts.")
    inString = input("Enter the encoded message: ")
    message = ""
    for numStr in inString.split():
        # convert the (sub)string to a number
        codeNum = eval(numStr)
        # append character to message
        message = message + chr(codeNum)
    print("The decoded message is:", message)

main()
```

Other String Methods

```
>>> s = "hello, I come here for an argument."
>>> s.capitalize()
'Hello, i come here for an argument.'
>>> s.title()
'Hello, I Come Here For An Argument.'
>>> s.lower()
'hello, i come here for an argument.'
>>> s.upper()
'HELLO, I COME HERE FOR AN ARGUMENT.'
>>> s.replace("I", "you")
'hello, you come here for an argument.'
```

Other String Methods

```
>>> s.center(54)
'    hello, I come here for an argument    '
>>> s.center(10)
'hello, I come here for an argument'
>>> s.count('e')
5
>>> s.find(',')
5
```




```
>>> " ".join(["Man", "from", "Earth"])
'Man from Earth'
>>> "++".join(["Man", "from", "Earth"])
'Man++from++Earth'
```

Other String Methods

```
>>> '42'.isdigit()
True
>>> 'four'.isdigit()
False
>>> '   remove_the_spaces   '.strip()
'remove_the_spaces'
>>> 'Binghamton'.startswith('Bing')
True
>>> 'Binghamton'.startswith('bing')
False
```

Other String Operations

- s.ljust(width) – Like center, but s is left-justified
- s.lstrip() – Copy of s with leading white space removed
- s.rfind(sub) – Like find, but returns the rightmost position
- s.rjust(width) – Like center, but s is right-justified
- s.rstrip() – Copy of s with trailing white space removed

Input/Output as String Manipulation

- Let's say we want to enter a date in the format “05/22/2015” and output “May 22, 2015”

Input/Output as String Manipulation

- Pseudocode
  - Input the date in mm/dd/yyyy format (dateStr)
  - Split dateStr into month, day, and year strings
  - Convert the month string into a month number
  - Use the month number to lookup the month name
  - Create a new date string in the form “Month Day, Year”
  - Output the new date string

Input/Output as String Manipulation

```
# dateconvert.py
# Converts a date in form "mm/dd/yyyy" to "month day, year"

def main():
    # get the date
    dateStr = input("Enter a date (mm/dd/yyyy): ")
    # split into components
    monthStr, dayStr, yearStr = dateStr.split("/")

    # convert monthStr to the month name
    months = ["January", "February", "March", "April",
              "May", "June", "July", "August",
              "September", "October", "November", "December"]
    monthStr = months[int(monthStr)-1]

    # output result in month day, year format
    print("The converted date is:", monthStr, dayStr+",", yearStr)

main()
```

Input/Output as String Manipulation

- Run the program:

```
Enter a date (mm/dd/yyyy): 01/23/2010
The converted date is: January 23, 2010
```

Input/Output as String Manipulation

- Why do we use `int` and not `eval`?

```
>>> eval("05")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 1
    05
     ^
SyntaxError: invalid token
```

Input/Output as String Manipulation

- We can use the *str* function to convert a number into a string

```
>>> str(500)
'500'
>>> value = 3.14
>>> str(value)
'3.14'
```

Input/Output as String Manipulation

```
>>> value = 3.14
>>> print("The value is", value+".")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +:
  'float' and 'str'
```

Input/Output as String Manipulation

- Two ways to remedy this:

```
>>> print("The value is", value, ".")
The value is 3.14 .
>>> print("The value is", str(value)+".")
The value is 3.14.
```

Input/Output as String Manipulation

- Type conversion operations:

Function	Meaning
float(<expr>)	Convert expr to a floating point value
int(<expr>)	Convert expr to an integer value
str(<expr>)	Return a string representation of expr
eval(<string>)	Evaluate string as an expression