Objectives

Chapter 5: Sequences: Strings, Lists, and Files

- To understand the **string data type** and how **strings** are represented in the computer.

- To be familiar with various operations that can be performed on strings through built-in functions and the string library.

- To understand the basic idea of **sequences** and **indexing** as they apply to Python **strings and lists**.

Objectives (cont.)

- To be able to apply string formatting to produce attractive, informative program output.
- 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.

The String Data Type

- The most common use of personal 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='spam'
>>> print(str1, str2)
Hello spam
>>> type(str1)
<class 'str'>
>>> type(str2)
<class 'str'>
```

The String Data Type

- Getting a string as input

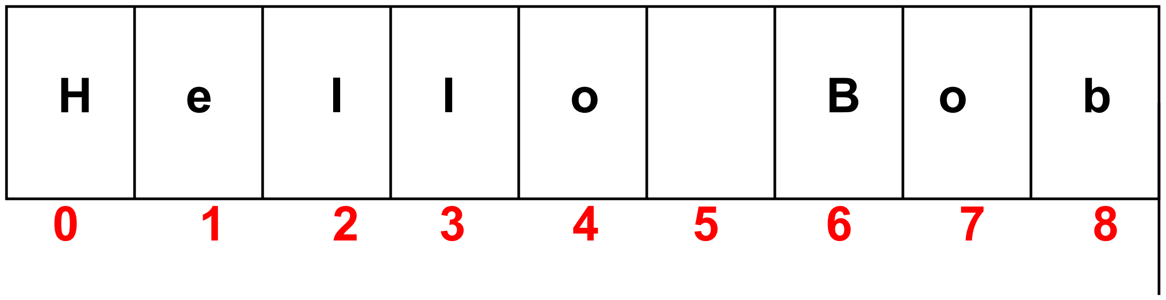
```
>>> firstName = input("Please enter your name: ")
Please enter your name: John
>>> print("Hello", firstName)
Hello John
```

- Notice that the input is not `eval`uated. We want to store the typed characters, not to evaluate them as a Python expression.

The String operations

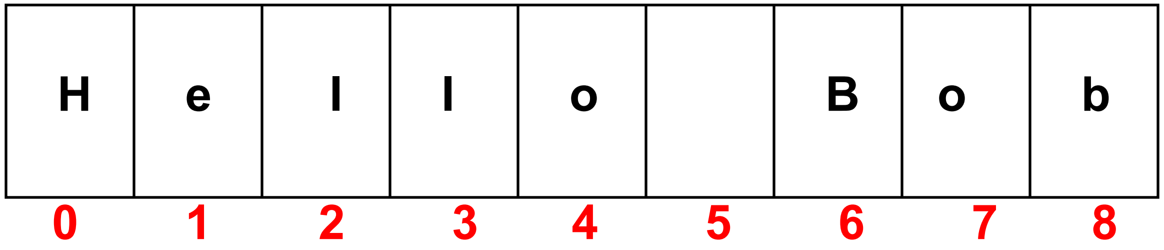
- We can access the individual characters in a string through ***indexing***.
- The positions in a string are numbered from the left, starting with **0**.
- The general form is **<string>[<expr>]**, where the value of **expr** determines which character is selected from the string.

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.

- We can index from the right side using **negative indexes**.

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

The String Data Type

- **Indexing** returns a string containing a single character from a larger string.
- We can also access a contiguous 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**

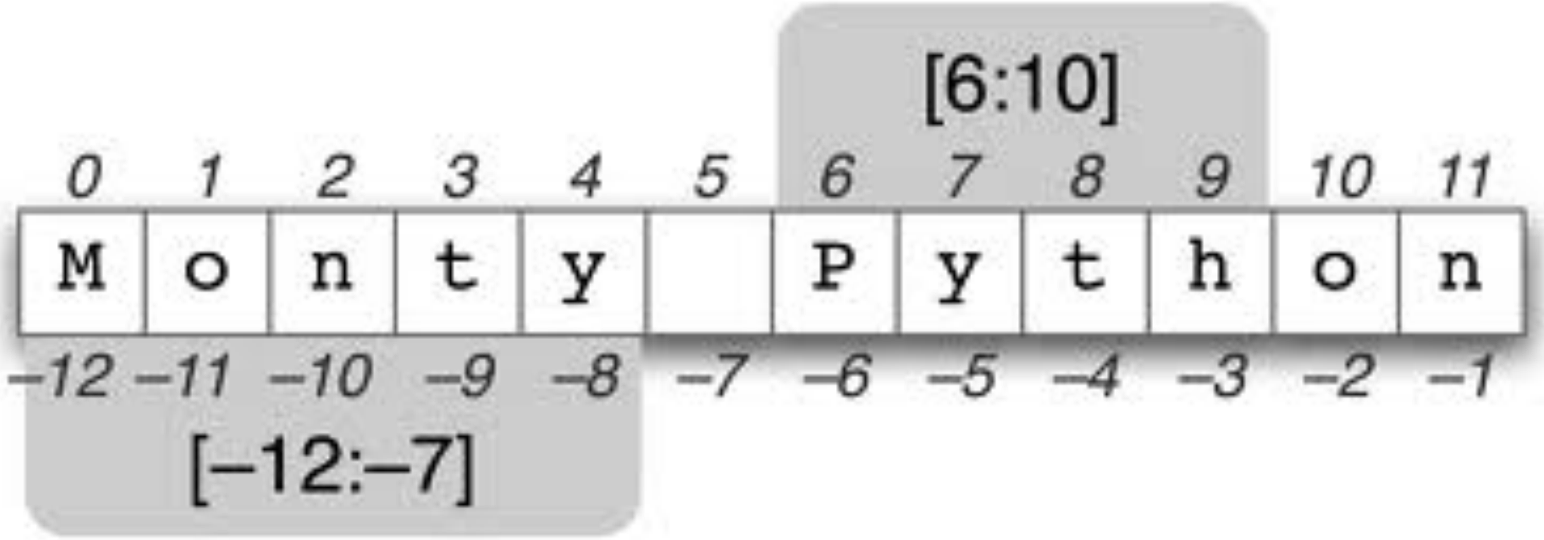
- The slice contains the substring beginning at position **start** and runs up to **but doesn’t include** the position **end**.

The String Data Type

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

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

The String Data Type



The String Data Type

- Can we put two strings together into a longer string?
  - Two ways

- ***Concatenation*** “glues” two strings together (+)

```
>>> "spam" + "eggs"
'spameggs'
>>> "Spam" + "And" + "Eggs"
'SpamAndEggs'
```

The String Data Type

- ***Repetition*** builds up a string by multiple concatenations of a string with itself (*)

```
>>> 3 * "spam"
'spamspamspam'
>>> "spam" * 5
'spamspamspamspamspam'
>>> (3 * "spam") + ("eggs" * 5)
'spamspamspameggseggseggseggseggs‘
```

- The function ***len*** will return the length of a string.

```
>>> len("spam")
4
```

- Can be used in the loop structure

```
>>> for ch in "Spam!":
        print (ch, end=" ")
>>> S p a m !
```

The String Data Type

String variable is kinds of object !

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

Simple String Processing

How to create a username on a computer system by using your first and last name ?

Simple String Processing

- **Usernames on a computer system**
  - Initial letter of first name , first seven letters of last name

```
# get user's first and last names
first = input("Please enter your first name (all lowercase): ")
last = input("Please enter your last name (all lowercase): ")

# concatenate first initial with 7 chars of last name
uname = first[0] + last[:7]
```

Simple String Processing

```
>>>
Please enter your first name (all lowercase): john
Please enter your last name (all lowercase): doe
uname =  jdoe

>>>
Please enter your first name (all lowercase): donna
Please enter your last name (all lowercase): rostenkowski
uname =  drostenk
```

- Read the input as a single string, then **split** it apart into substrings, each of which represents one word?
- Strings are objects and have **useful methods** associated with them
- One of these methods is ***split***. This will split a string into substrings based on spaces.

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

Strings and Secret Codes

- **Split** can be used on characters other than space, by supplying the character as a parameter.

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

Other String Methods

- There are a number of other **string methods**. Try them all!
  - **s.capitalize()** – Copy of s with only the first character capitalized
  - **s.title()** – Copy of s; first character of each word capitalized
  - **s.center(width)** – Center s in a field of given width

Other String Operations

- **s.count(sub)** – Count the number of occurrences of sub in s
- **s.find(sub)** – Find the first position where sub occurs in s
- **s.join(list)** – Concatenate list of strings into one large string using s as separator.
- **s.ljust(width)** – Like center, but s is left-justified

Other String Operations

- **s.lower()** – Copy of s in all lowercase letters
- **s.lstrip()** – Copy of s with leading whitespace removed
- **s.replace(oldsub, newsub)** – Replace occurrences of oldsub in s with newsub
- **s.rfind(sub)** – Like find, but returns the right-most position
- **s.rjust(width)** – Like center, but s is right-justified

Other String Operations

- s.rstrip() – Copy of s with trailing whitespace removed
- s.split() – Split s into a list of substrings
- s.upper() – Copy of s; all characters converted to uppercase

- Python lists are also a kind of sequence.

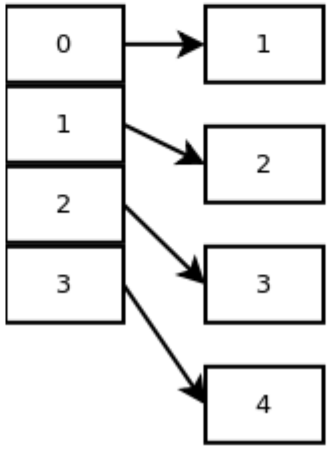
Strings, Lists, and Sequences

- It turns out that **strings** are really a special kind of ***sequence***, so these operations **also apply to sequences**!

```
>>> [1,2] + [3,4]
[1, 2, 3, 4]
>>> [1,2]*3
[1, 2, 1, 2, 1, 2]
>>> grades = ['A', 'B', 'C', 'D', 'F']
>>> grades[0]
'A'
>>> grades[2:4]
['C', 'D']
>>> len(grades)
5
```

Strings, Lists, and Sequences

- Strings are always sequences of characters, but ***lists*** can be sequences of arbitrary values.
- myNumber = [1, 2, 3, 4]
- myString=[“spam”, “U”]



- **Lists** can have **numbers, strings, or both**!

  myList = [1, "Spam ", 4, "U"]

Simple String Processing

Write program to solve problem:

- How to convert an integer(1-12) into the three letter abbreviation for that month?

Strings, Lists, and Sequences

- We create the lookup table for months to a list:

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

- To get the months out of the sequence: 0<=n<12, do this:

  monthAbbrev = months[n-1]

Strings, Lists, and Sequences

```
# month2.py
#  A program to print the month name, given it's number.
#  This version uses a list as a lookup table.

def main():

    # months is a list used as a lookup table
    months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun",
              "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"]

    n = eval(input("Enter a month number (1-12): "))

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

main()
```

- The code that creates the list is split over two lines. Python knows that the **expression isn’t complete until the closing] is encountered**.

Strings, Lists, and Sequences

```
# month2.py
#  A program to print the month name, given it's number.
#  This version uses a list as a lookup table.

def main():

    # months is a list used as a lookup table
    months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun",
              "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"]

    n = eval(input("Enter a month number (1-12): "))

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

main()
```

- Since the list is indexed starting from 0, the *n-1* calculation is straight-forward enough to put in the print statement without needing a separate step.

Strings, Lists, and Sequences

- This version of the program is easy to extend to print out the whole month name rather than an abbreviation!

- months = ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"]

Strings, Lists, and Sequences

- Lists are ***mutable***, meaning they can be changed. **Strings can not be changed.**

```
>>> myList = [34, 26, 15, 10]
>>> myList[2]
15
>>> myList[2] = 0
>>> myList
[34, 26, 0, 10]
>>> myString = "Hello World"
>>> myString[2]
'l'
>>> myString[2] = "p"

Traceback (most recent call last):
  File "<pyshell#16>", line 1, in -toplevel-
    myString[2] = "p"
TypeError: object doesn't support item assignment
```

Lists Have Methods, Too

- **Append** method can be used to add an item at the end of a list
- squares = []
- For x in range (1,5):
  - squares.append(x*x)

```
>>> squares
[1, 4, 9, 16]
```

String Representation and Message Encoding

- Inside the computer, strings are represented as sequences of 1's and 0's, just like numbers.
- A string is stored as a sequence of binary numbers, one number per character.
- It doesn't matter what value is assigned as long as it's done consistently.
- Question:
  - How to match number with character?

String Representation and Message Encoding

- **ASCII system** (American Standard Code for Information Interchange)
  - uses the numbers 0 through127 to represent the characters

- For example: A-Z are represented by the values 65-90, and the lowercase versions have codes 97-120

ASCII value	Character	Control character	ASCII value	Character	ASCII value	Character	ASCII value	Character
000	(null)	NUL	032	(space)	064	@	096	
001	☺	SOH	033	!	065	A	097	a
002	☻	STX	034	"	066	B	098	b
003	♥	ETX	035	#	067	C	099	c
004	♦	EOT	036	$	068	D	100	d
005	♣	ENQ	037	%	069	E	101	e
006	♠	ACK	038	&	070	F	102	f
007	(beep)	BEL	039	'	071	G	103	g
008	◘	BS	040	(	072	H	104	h
009	(tab)	HT	041	)	073	I	105	i
010	(line feed)	LF	042	*	074	J	106	j
011	(home)	VT	043	+	075	K	107	k
012	(form feed)	FF	044	,	076	L	108	l
013	(carriage return)	CR	045	-	077	M	109	m
014	♫	SO	046	.	078	N	110	n
015	☼	SI	047	/	079	O	111	o
016	►	DLE	048	0	080	P	112	p
017	◄	DC1	049	1	081	Q	113	q
018	↕	DC2	050	2	082	R	114	r
019	‼	DC3	051	3	083	S	115	s
020	π	DC4	052	4	084	T	116	t
021	§	NAK	053	5	085	U	117	u
022	▬	SYN	054	6	086	V	118	v
023	↨	ETB	055	7	087	W	119	w
024	↑	CAN	056	8	088	X	120	x
025	↓	EM	057	9	089	Y	121	y
026	→	SUB	058	:	090	Z	122	z
027	←	ESC	059	;	091	[	123	{
028	(cursor right)	FS	060	<	092	\	124	¦
029	(cursor left)	GS	061	=	093	]	125	}
030	(cursor up)	RS	062	>	094	^	126	~
031	(cursor down)	US	063	?	095	_	127	⌂

127 values are not enough!

- **Unicode: developed by the International Standards Organization to remedy this situation**
- Python supports **Unicode** (100,000+ characters)
- **Unicode uses the same codes as ASCII for the 127 characters originally defined there.**

String Representation and Message Encoding

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

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

String Representation and Message Encoding

- Using **ord** and **char** we can convert a string into and out of numeric form.

- **The encoding algorithm is simple**:

```
get the message to encode
for each character in the message:
   print the letter number of the character
```

- A for loop iterates over a sequence of objects, so the for loop looks like:
  for ch in <string>

String Representation and Message Encoding

```
# text2numbers.py
#     A program to convert a textual message into a sequence of
#         numbers, utlilizing the underlying Unicode encoding.

def main():
    print("This program converts a textual message into a sequence")
    print ("of numbers representing the Unicode encoding of the message.\n")

    # Get the message to encode
    message = input("Please enter the message to encode: ")

    print("\nHere are the Unicode codes:")

    # Loop through the message and print out the Unicode values
    for ch in message:
        print(ord(ch),  end=" ")

    print()

main()
```

Decode

- We now have a program to convert messages into a type of **"code"**, but it would be nice to have a program that could **decode the message**!

- **The outline for a decoder**:

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

Decode

```
# numbers2text.py
#     A program to convert a sequence of Unicode numbers into
#         a string of text.

def main():
    print ("This program converts a sequence of Unicode numbers into")
    print ("the string of text that it represents.\n")

    # Get the message to encode
    inString = input("Please enter the Unicode-encoded message: ")

    # Loop through each substring and build Unicde 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("\nThe decoded message is:", message)

main()
```