3.3 Comparing Strings

Python allows you to compare strings. This allows you to create decision structures that test the value of a string.

```
name1 = 'Mary'
name2 = 'Mark'
if name1 == name2:
     print('The names are the same.')
else:
     print('The names are NOT the same.')
```

Other String Comparisons

In addition to determining whether strings are equal or not equal, you can also determine whether one string is greater than or less than another string.

- The uppercase characters A through Z are represented by the numbers 65 through 90.
- The lowercase characters a through z are represented by the numbers 97 through 122.
- When the digits 0 through 9 are stored in memory as characters, they are represented by the numbers 48 through 57. (For example, the string 'abc123' would be stored in memory as the codes 97, 98, 99, 49, 50, and 51.)
- A blank space is represented by the number 32.

In addition to establishing a set of numeric codes to represent characters in memory, ASCII also establishes an order for characters. The character "A" comes before the character "B", which comes before the character "C", and so on.

When a program compares characters, it actually compares the codes for the characters.

The ASCII Character Set

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

Chapter 8 More About Strings

8.1 Basic String Operations

Concept:

Python provides several ways to access the individual characters in a string. Strings also have methods that allow you to perform operations on them.

Indexing

Each character in a string has an index that specifies its position in the string. Indexing starts at 0, so the index of the first character is 0, the index of the second character is 1, and so forth. The index of the last character in a string is 1 less than the number of characters in the string.

You can use an index to retrieve a copy of an individual character in a string.

Example:

```
my_string = 'Roses are red'
ch = my_string[6]
```

After this statement executes, ch will reference 'a'.

You can also use negative numbers as indexes, to identify character positions relative to the end of the string.

IndexError Exceptions

An IndexError exception will occur if you try to use an index that is out of range for a particular string.

The `len` Function

The `len` function can also be used to get the length of a string.

Example:

```
city = 'Boston'
size = len(city)
```

The second statement calls the `len` function, passing the city variable as an argument. The function returns the value 6, which is the length of the string 'Boston'. This value is assigned to the size variable.

Concatenation operation

This operation appends one string to the end of another string.

The + operator produces a string that is the combination of the two strings used as its operands.

Example:

```
>>> first_name = 'Emily' Enter
>>> last_name = 'Yeager' Enter
>>> full_name = first_name + ' ' + last_name Enter
>>> print(full_name) Enter
Emily Yeager
>>>
```

Strings Are Immutable

In Python, strings are immutable, which means that once they are created, they cannot be changed. Some operations, such as concatenation, give the impression that they modify strings, but in reality they do not.

Because strings are immutable, you cannot use an expression in the form *string* [*index*] on the left side of an assignment operator.

8.2 String Slicing

Concept:

You can use slicing expressions to select a range of characters from a string

To get a slice of a string, you write an expression in the following general format:

```
    string[start : end]
```

In the general format, *start* is the index of the first character in the slice, and *end* is the index marking the end of the slice. The expression will return a string containing a copy of the characters from *start* up to (but not including) *end*.