Topic String

Mark Lutz, Learning Python 5th, O'REILLY

String

- An ordered collection of characters used to store and represent text-based information
- strings can be used to represent just about anything that can be encoded as text: symbols and words (e.g., your name), contents of text files loaded into memory, Internet addresses, Python programs, and so on.
- They can also be used to hold the absolute binary values of bytes, and multibyte Unicode text used in internationalized programs.
- Python has no distinct type for individual characters; instead, you just use one-character strings.
- Python strings are categorized as immutable sequences, meaning that the characters they contain have a left-to-right positional order and that they cannot be changed in-place.

Common string literals and operations

Operation	Interpretation	Operation	Interpretation
S = ''	empty string	S1 + S2	concatenate
S = "spam's"	double quotes, same as single	S * 3	repeat
S = 's\np\ta\x00m'	escape sequences	S[i]	Index
S = """.."""	triple-quoted block strings	S[i:j]	slice
S = r'\temp\spam'	raw strings	len(S)	length
"a %s parrot" % kind	string formatting expression	S.find('pa')	search
"a {0} parrot".format(kind)	string formatting method	S.rstrip()	remove whitespace
S.replace('pa', 'xx')	replacement	S.split(',')	split on delimiter
'spam'.join(strlist)	delimiter join	S.lower()	case conversion
S.encode('latin-1')	unicode encoding	B.decode('utf8')	unicode decoding
re.match('sp(.*)am', line)	Pattern matching: library module		

Single-Quoted and Double-Quoted

- Around Python strings, single- and double-quote characters are interchangeable.

```
>>> 'shrubbery', "shrubbery"
('shrubbery', 'shrubbery')
```

- The reason for supporting both is that it allows you to embed a quote character of the other variety inside a string without escaping it with a backslash.

```
>>> 'knight"s', "knight's"
('knight"s', "knight's")
```

- Python automatically concatenates adjacent string literals in any expression, although it is almost as simple to add a + operator between them to invoke concatenation explicitly.

```
>>> title = "Meaning " 'of' " Life"
>>> title
'Meaning of Life'
```

Escape Sequences

- Backslashes are used to introduce special character codings known as escape sequences.

```
>>> s = 'a₩nb₩tc'
a
b	c
>>> len(s)  # The two characters ₩n stand for a single character
5
```

- In fact, 3.X defines str strings formally as sequences of Unicode code points, not bytes, to make this clear.

Escape	Meaning	Escape	Meaning
\n	Newline (linefeed)	\"	Double quote
\r	Carriage return	\b	Backspace
\t	Horizontal tab	\\	Backslash
\'	Single quote	\newline	Ignored

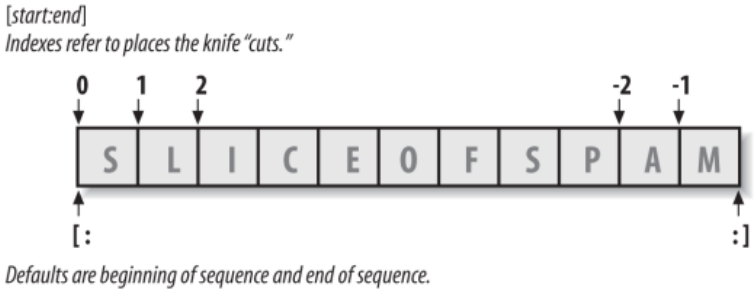
Basic Operations

- You can iterate over strings in loops using for statements

```
>>> myjob = "hacker"
>>> for c in myjob: print(c, end=' ')    # h a c k e r
```

- You can alose test membership for both characters and substrings with the in expression operator, which is essentially a search.

```
>>> "k" in myjob    # True
>>> "z" in myjob    # False
```



- You can access their components by position

```
>>> S = 'spam'
>>> S[0], S[−2]                  # ('s', 'a')
>>> S[1:3], S[1:], S[:−1]        # ('pa', 'pam', 'spa')
>>> S[::-1]                      # maps
```

Immutable

- You cannot change a string in place.

```
>>> S = 'spam'
>>> S[0] = 'x'
TypeError: 'str' object does not support item assignment
```

- To change a string, you generally need to build and assign a new string using tools such as concatenation and slicing, and then, if desired, assign the result back to the string's original name:

```
>>> S = S + 'SPAM!'
>>> S
'spamSPAM!'
```

```
>>> S = S.replace('pa', 'wi')
>>> S
'swimSPAM!'
```

- It's also possible to build up new text values with string formatting expressions.

```
>>> 'That is {0} {1} bird!'.format(1, 'dead')
'That is 1 dead bird!'
```

Conversion

- You cannot add a number and a string together in Python.

```
>>> "42" + 1
TypeError: Can't convert 'int' object to str implicitly
```

- It is also possible to convert a single character to its underlying integer code by passing it to the built-in ***ord*** function, or ***chr*** inversely.

```
>>> ord('A')
65
```

```
>>> chr(65)
'A'
```

```
>>> chr(ord('A') + 1)
'B'
```

- Binary and Hex conversion tasks

```
>>> int('1101', 2)
13
>>> bin(13)
'0b1101'
```

```
>>> int('AA', 16)
170
>>> hex(170)
'0xaa'
```

Character Encoding Schemes (1/2)

- When text is stored on files, for example, its character set determines its format.
  - **Character sets** are standards that assign integer codes to individual characters so they can be represented in computer memory.
  - For example, **ASCII** standard defines character codes from 0 through 127 and allows each character to be stored in one 8-bit byte.
  - ***ord*** function gives the binary identifying value for a character, and ***chr*** returns the character for a given integer code value.
- **Unicode** text is sometimes referred to as "wide-character" strings, because characters may be represented with multiple bytes.
  - Some alphabets define so many characters that it is impossible to represent each of them as one byte.
  - Unicode is typically used in internationalized programs, to represent European, Asian, and other non-English character sets.
- **Encoding** is the process of translating a string of characters into its raw bytes form, according to a desired encoding name.
- **Decoding** is the process of translating a raw string of bytes into its character string form, according to its encoding name.

Character Encoding Schemes (2/2)

- **UTF-8** encoding allows a wide range of characters to be represented by employing a variable-sized number of bytes.
  - Character codes less than 128 are represented as a single byte
  - Codes between 128 and 0x7ff (2047) are turned into 2 bytes
  - Codes above 0x7ff are turned into 3- or 4-byte
- ***str*** for representing decoded Unicode text (including ASCII)
- UTF-16 and UTF-32 format text with a fixed-size 2 and 4 bytes per each character scheme, respectively.

```
>>> S = 'ni'
>>> S.encode('ascii')          # b'ni'
>>> len(S.encode('ascii'))     # 2
```

```
>>> S.encode(utf8')            # b'ni'
>>> len(S.encode(utf8'))       # 2
>>> len(S.encode('utf16'))     # 6
```

```
>>> S = '가'
>>> S.encode('ascii')
UnicodeEncodeError: 'ascii' codec can't encode character '₩uac00'
>>> S.encode('utf8')           # b'₩xea₩xb0₩x80'
```

```
>>> import sys
>>> sys.getdefaultencoding() # utf-8
```

Q&A

Parsing Text

- Text parsing is analyzing structure and extracting substrings.

```
>>> line = 'aaa bbb ccc'
>>> cols = line.split()
['aaa', 'bbb', 'ccc']
```

```
>>> line = 'bob,hacker,40'
>>> cols = line.split(',')
['bob', 'hacker', '40']
```

- To change a string, you generally need to build and assign a new string using tools such as concatenation and slicing, and then, if desired, assign the result back to the string's original name:

```
>>> S = S + 'SPAM!'
>>> S
'spamSPAM!'
```

```
>>> S = S.replace('pa', 'wi')
>>> S
'swimSPAM!'
```

- It's also possible to build up new text values with string formatting expressions.

```
>>> 'That is {0} {1} bird!'.format(1, 'dead')
'That is 1 dead bird!'
```

Raw Strings

- The special treatment of backslashes for introducing escapes can lead to trouble.

```
myfile = open('C:₩new₩text.dat', 'w')  # C:(newline)ew(tab)ext.dat
```

- If the letter r (uppercase or lowercase) appears just before the opening quote of a string, it turns off the escape mechanism.

```
myfile = open( r'C:₩new₩text.dat', 'w' )
# myfile = open('C:₩₩new₩₩text.dat', 'w')

print( 'c₩new₩text' )
```

```
c
ew      ext
```

```
print( r'c₩new₩text' )      # c₩new₩text
```