String: a sequence of characters.
Indicated with quotes: ‘ ’ or " "
**Triple quotes**: preserve both the vertical and horizontal formatting of the string.

Allows you to type tables, paragraphs, whatever and preserve the formatting:

```
""" this is
 a test
 today"""
```

**Basic String Operations**

```python
s = 'spam'
- length operator `len()`
  `len(s)`
  4
- + is concatenate
  `new_str = 'spam' + '-' + 'spam-'`
  `print(new_str)`
  `spam-spam-`
- * is repeat, the number is how many times
  `new_str * 3`
  `spam-spam-spam-spam-spam-spam-`
```
in: check if a substring exists in the string. Returns True or False

```python
my_str = 'aabbccdd'
'a' in my_str     True
'abb' in my_str   True
'x' in my_str     False
'aba' in my_str   False
```

Indexing Strings

- every character has an index, starting at 0
- the index operator is []

```python
my_str = 'hello world'
my_str[2]  =>  'l'
my_str[-1] =>  'd'
my_str[11] =>  ERROR, index out of range
```
Slicing: select a subsequence.

Syntax is `[start : finish]`, where:
- `start` is the index that starts the subsequence
- `finish` is the index of one after subsequence end

If either `start` or `finish` is not provided,
- `start` defaults to the beginning of the sequence,
- `finish` defaults to the end of the sequence.

\[
\begin{array}{cccccccccc}
\text{char} & \text{index} & \h & \text{e} & \text{l} & \text{l} & \text{o} & \text{w} & \text{o} & \text{r} & \text{l} & \text{d} \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\end{array}
\]

\[
\text{my\_str} = \text{‘hello world’} \\
\text{my\_str}[6:10] \Rightarrow \text{‘worl’}
\]
Defaults

my_str[6:] ⇒ ‘world’

Slicing also takes three arguments:
[start : finish : countBy]

Default for countBy is 1
my_str = ‘hello world’
my_str[::2] ⇒ ‘hlowrd’
Python *idioms* are “phrases” for a common task.

- **copy a string:**
  ```python
  my_str = 'hi mom'
  new_str = my_str[:]
  ```

- **reverse a string**
  ```python
  my_str = "madam I’m adam"
  reverse_str = my_str[::-1]
  ```

**Indexing $S[i]$ Overview**

$S[0]$ fetches the first item.

Negative indices count backwards from end.

$S[-2]$ is second from end
Slicing $S[i:j]$ Overview

- Upper bound is not inclusive.
- Defaults are 0 and end, if omitted
- Examples
  - $S[1:4]$ fetches from 1 up to, but not including 4
  - $S[1:]$ fetches from 1 to end
  - $S[:4]$ fetches from beginning up to, but not including 4
  - $S[:1]$ fetches from beginning up to, but not including the last item
  - $S[:]$ fetches all
    (A top-level copy of $S$ – more later.)

Exercise 6-1
Strings are immutable:
you cannot change one once you create it.

but, you can use it to make another string
(copy it, slice it, etc.)

For example, the reverse string idiom
doesn’t change the existing string,
it simply makes a new string
with the letters reversed!

Immutable strings

- Can’t change a string once it is created.

```python
>>> a_str = 'spam'
>>> a_str[1] = 'l'  # => Error
```

`a_str` namespace

`spam`
Immutable strings

- Can’t change a string once it is created.
  ```python
  >>> a_str = 'spam'
  >>> a_str[1] = 'l'  => Error
  ```

- You can use it to make new strings
  ```python
  >>> x = a_str[:1] + 'l' + a_str[2:]
  >>> print(a_str, x)
  spam slam
  ```

- You can also reassign it
  ```python
  >>> a_str = a_str.replace('p', 'c')
  >>> print(a_str)
  scam
  ```

Note: nothing refers to ‘spam’
replace

s.replace(old, new [,max])

s = 'steel'
y = s.replace('e', 'o')
print(y)

stool
print(s)  # s did not change
steel

Method

Methods operate on an object.
For example, s.replace('l', 'x')
does a replace on string s (a.k.a. object s)
Find

x = 'hello'
x.find('l')    # find index of first 'l' in x
2

Method 'find' operates on the string object 'x' and the two are associated by using the "dot" notation: x.find('l'). Terminology: the thing(s) in parenthesis, i.e. the 'l' in this case, is called an argument.

Split
(possibly the most useful string method)

s = 'Always look on the bright side of life.'
y = s.split()
print(y)
    o ['Always', 'look', 'on', 'the', 'bright', 'side', 'of', 'life.]
print(y[1])
    o look
More methods

- `s.capitalize()`
- `s.center(width)`
- `s.count(sub[,start [,end]])`
- `s.ljust(width)`
- `s.lower()`
- `s.upper()`
- `s.lstrip()`
- `s.rfind(sub, [,start [,end]])`
- `s.splitlines([keepends])`
- `s.strip()`
- `s.translate(table [, delchars])`

more at http://docs.python.org/lib/string-methods.html

Exercise 6-2
Iteration:
another type of repetition.

Keyword: \texttt{for}
for item in sequence:
suite

for ch in ‘abc’:
    print(ch)

1. First time through, ch = ‘a’
2. Second time through, ch = ‘b’
3. Third time through, ch = ‘c’
4. Nothing left in sequence (string), so quit.
Iteration (for) in Python is very powerful.

```python
range([start], stop,[,step])

for i in range(5):
    print(i, end = ' ')
0 1 2 3 4

for i in range(2,7):
    print(i, end = ' ')
2 3 4 5 6
```
range continued

for i in range(1,10,2):
    print(i, end = ' ')
  1 3 5 7 9

for i in range(10,1,-1):
    print(i, end = ' ')
  10 9 8 7 6 5 4 3 2

for i in range(5):
    print(i, end = '')
Exercise 6-3

Palindrome Examples

civic, kayak, radar, rotor, level
A man, a plan, a canal: Panama.
- Look at 05_execution_trace (pal1.py)
- Faster version (pal2.py)
- Fastest version (pal3.py)

String comparisons, single char

- There are two systems for representing characters: ASCII and Unicode.
- ASCII takes the English letters, numbers and punctuation marks and associates them with an integer number.
- Single character comparisons are based on that number.
Comparisons within sequence

- It makes sense to compare within a sequence (lower case, upper case, digits).
  - ‘a’ < ‘b’ True
  - ‘A’ < ‘B’ True
  - ‘1’ < ‘9’ True

- Can be weird outside of the sequence
  - ‘a’ < ‘A’ False
  - ‘a’ < ‘0’ False
Whole string comparison

Compare the first element of each string
- if they are equal, move on to the next character in each string
- if they are not equal, the relationship between those to characters are the relationship between the strings
- if one ends up being shorter (but equal), the shorter is smaller

Examples

- ‘a’ < ‘b’ True
- ‘aaab’ < ‘aaac’
  - first difference is at the last char. ‘b’<‘c’ so ‘aaab’ is less than ‘aaac’. True
- ‘aa’ < ‘aaz’
  - The first string is the same but shorter. Thus it is “smaller”. True
print("{} is {} years old".format("Bill",25))
Bill is 25 years old
The format string contains a set of format descriptors that describe how an object is to be printed.

format descriptor:
:align[min_width][.precision][descriptor]

where [ ] are optional
<table>
<thead>
<tr>
<th>descriptor</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>string</td>
</tr>
<tr>
<td>d</td>
<td>number (decimal)</td>
</tr>
<tr>
<td>f</td>
<td>floating point</td>
</tr>
<tr>
<td>e</td>
<td>floating point (exponent)</td>
</tr>
<tr>
<td>%</td>
<td>floating point as percent</td>
</tr>
</tbody>
</table>

Objects are matched *in order* with format descriptors.

```
print("{} is {} years old".format("Bill",25))
```

Bill is 25 years old
Width

print("{:>10s} is {:<10d} years old".format("Bill",25))

string 10 spaces wide including the object right justified (>)

number 10 spaces wide including the object < means left justified

Bill is 25 years old
10 spaces
10 spaces
this space in the format string already

Alignment

<table>
<thead>
<tr>
<th>alignment</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>left</td>
</tr>
<tr>
<td>&gt;</td>
<td>right</td>
</tr>
<tr>
<td>==</td>
<td>center</td>
</tr>
</tbody>
</table>
Default alignment

"{:10s} is {:10d} years old".format("Bill",25)
'Bill       is         25 years old'

Defaults:
- String default is "left"
- Number default is "right"

Precision

- print(math.pi)
  3.141592653589793

- print("{:1.4f}".format(math.pi))
  3.1416
  (4 decimal points of precision, with rounding)

- print("{:10.2f}".format(math.pi))
  3.14
  (10 spaces including the number and decimal pt.)
Position

- "{}, {}, {}".format('a','b','c')
  'a, b, c'
- "{0}, {1}, {2}".format('a','b','c')
  'a, b, c'
- "{2}, {1}, {0}".format('a','b','c')
  'c, b, a'
- "{2}, {1}, {0}, {1}".format('a','b','c')
  'c, b, a, b'

Position, width, precision

"{1:4d}, {0:6.2f}".format(math.pi,23)
' 23,   3.14'
Named

'\{lat\}, \{long\}'.format(lat='37.24N', long='-27.38W')
'37.24N, -27.38W'

coord = (3,5)
'X: \{0[0]\};  Y: \{0[1]\}'.format(coord)
'X: 3;  Y: 5'

Indexed

s = 'Hello'
t = "there"

'{0[0]},\{1[2]\}'.format(s,t)
'H,e'
Raw Strings allow you to include "\"

\nx = r"C:\Python32"

print(x)
C:\Python32