

**More on
strings, lists, dictionaries and files**

Operations on Sequences

The following operations apply to all sequences

Strings, lists, and tuples.

$s + r$	Concatenation
$s * n, n * s$	Makes n copies of s where n is an integer
$s \% d$	String formatting (s is a string)
$s[i]$	Indexing
$s[i:j]$	Slicing
$x \text{ in } s$	Membership test
$x \text{ not in } s$	Membership test
$\text{len}(s)$	Length
$\text{min}(s)$	Minimum item
$\text{max}(s)$	Maximum item

Lists

Lists are **ordered sequences** of arbitrary objects

May contain mixed types.

Lists are mutable (can be changed).

List **Methods**

- ▶ **s.append(x)** # Append element **x** to a list
- ▶ **s.extend(r)** # Appends list **r** to the end of **s**
- ▶ **s.count(x)** # Count occurrences of **x** in **s**
- ▶ **s.index(x)** # Return smallest **i** where **s[i] == x**
- ▶ **s.insert(i,x)** # Insert an element into a list
- ▶ **s.pop([i])** # Pops element **i** from the end of the list.
- ▶ **s.remove(x)** # Searches for **x** in the list and removes it
- ▶ **s.reverse()** # Reverses the items of **s** **in place**
- ▶ **s.sort([cmpfunc])** # Sorts the items of **s** **in place**.

Slices

The slicing operator $s[i:j]$

Extracts all elements $s[n]$ where $i \leq n < j$

$a = [0,1,2,3,4,5,6,7,8]$

$b = a[3:6]$ # $b = [3,4,5]$

If either index is omitted, beginning or end of sequence is assumed

$c = a[:3]$ # $c = [0,1,2]$

$d = a[5:]$ # $d = [5,6,7,8]$

Negative index is taken from the end of the sequence

$e = a[2:-2]$ # $e = [2,3,4,5,6]$

$f = a[-4:]$ # $f = [5,6,7,8]$

No indices just makes a copy (which is sometimes useful)

$g = a[:]$ # $g = [0,1,2,3,4,5,6,7,8]$

List Operations

The following operations are applicable **ONLY** to lists

`s[i] = x` Element assignment

`s[i:j] = r` Slice assignment

`del s[i]` Element deletion

`del s[i:j]` Slice deletion

Examples

`a = [0,1,2,3,4,5,6]`

`a[3] = -10`

`# a = [0,1,2,-10,4,5,6]`

`a[1:4] = [20,30,40,50]`

`# a = [0,20,30,40,50,4,5,6]`

`a[1:5] = [100]`

`# a = [0,100,4,5,6]`

`del a[1]`

`# a = [0,4,5,6]`

`del a[2:]`

`# a = [0,4]`

Note:

Strings and tuples are **immutable** and can't be modified.

Dictionaries

Dictionaries are **associative arrays** (hashes)

Provide a **mapping between keys and objects**.

Keys may be any **immutable** object (strings, tuples, numbers)

Unordered (unlike sequences)

Methods

- ▶ **d.clear()** # Remove all items
- ▶ **d.copy()** # Makes a copy of the dictionary
- ▶ **d.has_key(k)** # Tests for existence of a key
- ▶ **d.items()** # Return a list of (key,value) pairs
- ▶ **d.keys()** # Return a list of keys
- ▶ **d.values()** # Return a list of values
- ▶ **d.update(b)** # Adds all objects in dictionary b to d
- ▶ **d.get(k [,f])** # Returns d[k] if found.
#Otherwise, return f.

Example

```
d = { 'name': 'Dave', 'uid':104 }
```

```
k = d.keys() # Returns list ['name','uid']
```

```
v = d.values() # Returns list ['Dave',104]
```

```
i = d.items() # Returns list of (key, value) pair tuple  
                [('name','Dave'),('uid',104)]
```

Type Conversion

The following functions convert between types

- ▶ **int(x)** Convert x to an integer
- ▶ **long(x)** Convert x to a long integer
- float(x)** Convert x to a float
- complex(r,i)** Creates a complex number
- ▶ **str(x)** Create a string
- ▶ **repr(x)** Create an expression string
- ▶ **eval(str)** Evaluate a string and return an object
- tuple(s)** Convert sequence s to a tuple
- ▶ **list(s)** Convert sequence s to a list
- chr(x)** Convert an integer to a character
- ord(x)** Convert single character to an integer code
- hex(x)** Convert x to a hex string
- oct(x)** Convert x to an octal string

Files

`open(filename [, mode])`

Opens a file and returns a **file object**.

Mode is one of the following

"r" - Open for reading (the default)

"w" - Open for writing (destroys old contents if any)

"a" - Open for append

"r+" - Open for read/write (updates)

"w+" - Open for read/write (destroys old contents first).

An optional **"b"** may be appended to specify binary data.

Example:

```
f = open("foo")    # Open for reading
```

```
g = open("bar","w") # Open for writing
```

```
h = open("spam","rb") # Open for reading (binary)
```

Files

- The `open()` function

```
f = open("foo","w")    # Open a file for writing  
g = open("bar","r")    # Open a file for reading
```

- Reading and writing data

```
f.write("Hello World")  
data = g.read()        # Read all data  
line = g.readline()   # Read a single line  
lines = g.readlines() # Read data as a list of lines
```

- Formatted I/O

Use the `%` operator for strings (works like C `printf`)

```
for i in range(0,10):  
    f.write("2 times %d = %d\n" % (i, 2*i))
```

File Methods

The following **methods** may be applied to a file object

- ▶ **f.read([n])** Read at most n bytes
- ▶ **f.readline()** Read a single line of input
- ▶ **f.readlines()** Read all input lines
- ▶ **f.write(s)** Write string s
- ▶ **f.writelines(l)** Write all strings in list l.
- ▶ **f.close()** Close the file
- f.tell()** Return the current file pointer
- f.seek(offset [,where])** Seek to a new position
- f.isatty()** Returns 1 if an interactive terminal
- f.flush()** Flush output buffers
- f.truncate([n])** Truncate file to at most n bytes
- f.fileno()** Return integer file descriptor

Standard Input, Output, Error

Standard Files

Found in the `sys` module

`sys.stdin` - Standard input

`sys.stdout` - Standard output

`sys.stderr` - Standard error

Many built-in functions use these
print outputs to `sys.stdout`

- ▶ `input()` and `raw_input()` read from `sys.stdin`
- ▶ `s = raw_input("type a command : ")`
- ▶ `print "You typed ", s`

Error messages and the interactive prompts go to `sys.stderr`

You can replace these files with other files if you want

```
import sys  
sys.stdout = open("output","w")
```

Formatted I/O

The % operator for strings

```
print "%d %f %s" % (42, 3.1415, 'Hello')  
s = "Your name is '%s %s'" % (firstname, lastname)
```

Each of the format codes (preceded by %) are replaced by elements of a tuple

Try “\n” and “\t” within a print statement

Built-in Functions

Contained in the **__builtin__** module (always available)

abs()	eval()	intern()	open()	str()
apply()	execfile()	isinstance()	ord()	tuple()
callable()	filter()	issubclass()	pow()	type()
chr()	float()	len()	range()	vars()
cmp()	getattr()	list()	raw_input()	xrange()
coerce()	globals()	locals()	reduce()	
compile()	hasattr()	long()	reload()	
complex()	hash()	map()	repr()	
delattr()	hex()	max()	round()	
dir()	id()	min()	setattr()	
divmod()	input()	oct()	slice()	

To get this list:

```
>>> dir(__builtin__)
```

Python Library

The string module

- Various string processing functions

`string.atof(s)` # Convert to float, **s is a defined string**

`string.atoi(s)` # Convert to integer

`string.atol(s)` # Convert to long

▶ `string.count(s, pattern)` # Count occurrences of pattern in s

▶ `string.find(s, pattern)` # Find pattern in s

▶ `string.split(s, sep)` # Split a string

▶ `string.join(strlist, sep)` # Join a list of string

▶ `string.replace(s, old, new)` # Replace occurrences of old with new

- Examples

```
s = "Hello World"
```

```
a = string.split(s) # a = ['Hello','World']
```

```
b = string.replace(s,"Hello","Goodbye")
```

```
c = string.join(["foo","bar"],":") # c = "foo:bar"
```

Remember you have to first do the import dance >>> import string

Calling string methods

```
>>> dir("")
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__eq__',  
'__ge__', '__getattr__', '__getitem__', '__getslice__', '__gt__',  
'__hash__', '__init__', '__le__', '__len__', '__lt__', '__mul__',  
'__ne__', '__new__', '__reduce__', '__repr__', '__rmul__',  
'__setattr__', '__str__',  
'capitalize', 'center', 'count', 'decode', 'encode', 'endswith',  
'expandtabs', 'find', 'index', 'isalnum', 'isalpha', 'isdigit',  
'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower',  
'lstrip', 'replace', 'rfind', 'rindex', 'rjust', 'rstrip', 'split', 'splitlines',  
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper']
```

```
>>> s = 'my python is cool'
```

```
>>> s.title()
```

```
'My Python Is Cool'
```

```
>>> s = s.upper()
```

```
'MY PYTHON IS COOL'
```

```
>>> s.replace('OO', 'xx')
```

```
'MY PYTHON IS CxxL'
```

>>> dir([])

**#Guess it it will print ? Learn to ignore *name*
for now**

>>> dir({})

#Guess it it will print ?

>>> dir()

#Guess it it will print ?

>>> dir(file)

#Guess it it will print ?

The sys module

- **Functions and variables used by the interpreter**

sys.argv	# Command line arguments
sys.path	# Module search path
sys.maxint	# Maximum integer value
sys.stdin	# Standard input
sys.stdout	# Standard output
sys.stderr	# Standard error
sys.ps1	# Interactive interpreter prompt (>>>)
sys.ps2	# Interactive interpreter prompt (...)
sys.exc_info()	# Information about last raised exception
sys.exit(n)	# Exit from the interpreter

- **In addition, the module provides version and installation information**

sys.version	# Version string
sys.prefix	# Installation prefix
sys.copyright	# Copyright message
sys.executable	# Name of executable

Command Line Options

Reading command line options

- Passed as a list in `sys.argv`

```
# printopt.py
import sys
for i in range(0,len(sys.argv)):
    print "sys.argv[%d] = %s" % (i, sys.argv[i])
```

- Example:

```
unix % python printopt.py foo bar 34 -p
sys.argv[0] = printopt.py
sys.argv[1] = foo
sys.argv[2] = bar
sys.argv[3] = 34
sys.argv[4] = -p
```

Note: program name is passed in `sys.argv[0]`

The End

Format codes

- d,I** Decimal integer
- u** Unsigned integer
- o** Octal integer
- x** Hex integer
- X** Hex integer (uppercase letters)
- f** Floating point as [-]m.dddddd
- e** Floating point as [-]m.dddddde+xx
- E** Floating point as [-]m.ddddddeE+xx
- g,G** Use e or f depending on the size of the exponent.
- s** String or any object
- c** Single character
- %** Literal character

Character Escape Codes

Python supports the standard character escape code

<code>\</code>	Newline Continuation
<code>\\</code>	Backslash
<code>\'</code>	Single quote
<code>\"</code>	Double quote
<code>\a</code>	Bell
<code>\b</code>	Backspace
<code>\e</code>	Escape
<code>\n</code>	Line feed
<code>\v</code>	Vertical tab
<code>\t</code>	Horizontal tab
<code>\r</code>	Carriage return
<code>\0</code>	Null
<code>\0XX</code>	Octal character value
<code>\xXX</code>	Hex character value

Note: Escape codes are not interpreted in raw strings `a = r"\n\r"`