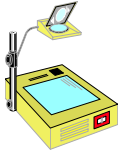




## Python: Chapter 2 Types and Operators



Learning Python, Lutz & Ascher

1

## Structure of a Python Program

- Programs are composed of modules
- Modules contain statements
- Statements create and process objects

2

## Lots of built-in types

- To enable rapid prototyping
- Avoid re-making the “wheel”
- Avoid the “grunt” work
- They are more efficient than user-defined
- **They work!**

3

## preview

Object type	Example usage
numbers	3.1415, 999L, 3+4j
Strings	'Spam', "you're"
Lists	[1, [2, "three"], 4]
Dictionaries	{1:'one', 2, 'two'}
Tuples	(1, 'one', 2, 'two')
Files	Text=open('file', 'r').read()

4

## numbers

- Usually C long
- Can use extended precision: l or L
- Complex numbers
- Operators:
  - +, -, \*, /
  - 2\*\*3
  - x << y, shift x left y bits
  - -x, +x, ~x: unary negation, identity, bitwise compliment
  - x and y, logical and
  - Is, is not: identity tests
  - In, not in: sequence membership
  - x & y: bitwise and
- Mixed types: convert up, same as C

```
x = 1
x << 2
x | 2
x & 3
```

```
import math
math.pi
abs(-12)
```

5

## Strings

S = ""	Empty string
S = "cat's"	
s1 + s2	Concatenation
s * 3	repeat
s[i]	index
s[i:j]	slice
len(s)	Length of s
for x in s	iteration
'm' in s	membership

6

## String slice

- String offsets start at 0 and end 1 less than the length
- Negative offsets are added to the length of the string to get the offset, or
- Can think of a negative offset as counting backward from the end (right)

```
s = "cat"
print s[-3:3]
print s[0:], s[:]
```

```
import sys
print sys.argv[1:]
```

7

## String: immutable sequence!

Safety first:

Strings are immutable: cannot change in place:

```
>>> s = "cat"
>>> s[0] = 'x'
      ERROR
>>> s = "cat"
>>> s = 'x' + s[1:]
```

Can use C formatting

```
>>> 'That is %d %s bird!' % (1, 'dead')
```

8

## string examples

```
>>> import string
>>> s = 'cat'
>>> string.upper(s)
CAT
>>> String.find(s, 'a')
1

>>> x = "79"
>>> y = string.atoi(x)
>>> printn y + 1

>>> z = 'you\'re cat'
```

String backslash characters:

\a	bell
\b	backspace
\n	newline
\t	tab

9

## Generic type concepts

- A few general ideas usually apply to lots of situations
- For built-in types, operations work the same for all types on a category

10

## Three type (and operation) categories

- Numbers support addition, multiplication, etc.
- Sequences support indexing, slicing, concatenation, etc.
- Mappings support indexing by key, etc

Consider sequence objects X and Y:

X + Y makes a new sequence w/ contents of both  
X \* N makes a new sequence with N copies of X

11

## lists

- Most flexible ordered collection object
- Can contain any object: numbers, strings, lists
- Accessed by offset
- Variable length, heterogeneous
- Mutable
- Are arrays of object references

12

## List constants & operations

L = []	An empty list
L = [0, 1, 2]	3 items, index 0 to 2
L = ['ab', ['cd', 'ef']]	Nested sublists
L1[i], L2[i][j]	index
L[i:j]	slice
len(L)	length
L * 3	repeat
for x in L	Iteration
3 in L	membership

13

## List constants & operations

L.append(4)	Methods: grow
L.sort()	
L.index(1)	Search
L.reverse()	
del L[k]	shrink
L[i:j] = []	Shrink
L[i] = 1	Assignment
L[i:j] = [4, 5, 6]	Slice assignment
Range(4), xrange(0,4)	Make lists/tuples of ints

14

## List examples

```
>>>len([1, 2])
>>>[1, 2] + [5]
>>>['A']*4
>>>for x in [1,2,3]: print x
>>>L = [1,2,3]
>>>L[-1]

>>>L=['eat', 'more', 'eggs']
>>>L[1] = 'less'
>>>print L[0:2]
>>>L.append('please')
>>>L
```

15

## Append versus concatenation

- Append tacks on a single item to the end
- L.append(X) changes L in place
- L + [X] makes a new list
- The sort method orders a list in-place; uses default comparison but can pass comparison test
- Append & sort change list but don't return the list as a result (return None)
- There are lots of other list operations

16

## Dictionaries

- Unordered collections of arbitrary objects
- Variable length, heterogeneous
- Mutable mapping
- Stored/fetched by key (not offset)
- Can replace many search algorithms
- Do the work of records and symbol tables
- Tables of object references

17

## Common constants & operations

D = {}	Empty dictionary
D = {'x':3, 'sum':1}	Two-item dictionary
D = {'food':{'ham':1, 'egg':2}}	nesting
D.has_key('x')	Membership test
D.keys()	List the keys
D.values()	List values
Len(D)	
D[key] = new	Adding, changing
Del D[key]	deleting

18

## Dictionary examples

```
>>> D = {'sum':2, 'x':3}
>>> D['x']
>>> len(D)
>>> D.has_key('x')
>>> D.keys()
>>> D['x'] = ['x', 'y', 'z']
>>> del D['x']
>>> D['x'] = 7
```

19

## Better dictionary example

```
>>> table = {'Python': 'Guido van Rossum',
            'Perl': 'Larry Wall',
            'Tel': 'John Ousterhout'}
>>> language = 'Python'
>>> creator = table[language]
>>> creator
>>> for lang in table.keys(): print lang, '\t', table[lang]
```

Notice the for statement: Since Dictionaries aren't sequences, you Can't iterate over them directly. But keys() returns a list of keys!

20

## Summary of dictionaries

- They are mappings, not sequences – no notion of order
- Assigning to new indexes adds entries
- Keys need not always be strings

21

## tuples

- Build simple groups of objects
- Like lists, except cannot be changed in place: immutable
- Usually written as items in parens (not square brackets)
- Share most properties with lists

22

## Examples of tuples

T = ()	An empty tuple
T = (0,)	1-item tuple; comma says not an expression
T = (0, 1, 2, 3)	Four item tuple
T = 0, 1, 2, 3	Same as above: can omit parens if unambiguous

23

## Why lists and tuples

- Tuples are immutable
- Lists are mutable
- Some built-in operations require tuples-not lists; e.g., argument lists

24

## files

- *open* creates a python file object with link to actual file
- After open, can read/write to file object using built-in methods
- A thin wrapper over C stdio system

25

## Using files

Output = open('data', 'w')	Create output file for w
Input = open('data', 'r')	Create input file for r
S = input.read()	read entire file into string S
S = input.read(n)	Read n bytes (1 or more)
S = input.readline()	Read next line, thru \n
L = input.readlines()	Read entire file into list of strings
output.write(S)	Write string S onto file
output.writelines(L)	Write strings in L onto file
output.close()	Manual close

26

## Using files

```
>>>myfile = open('data.dat', 'w')
>>>myfile.write('Hello world\n')
>>>myfile.close()

>>>myfile.open('data.dat', 'r')
>>>myfile.readline()
'Hello world\n'
>>>myfile.readline()
''
>>>myfile.seek(0)
>>>myfile.readline()
'Hello world\n'
```

There are lots  
More file methods,  
See Python manual

27

## Type categories summary

Object type	Category	Mutable?
Numbers	Numeric	No
Strings	Sequence	No
Lists	Sequence	No
Dictionaries	Mapping	Yes
Tuples	Sequence	No
Files	Extension	N/A

28

## generality

- Lists, dictionaries and tuples can hold any kind of object
- Lists, dictionaries and tuples can be arbitrarily nested
- Lists and dictionaries can dynamically grow and shrink

29

## Indexing a nested object

```
>>>L = ['abc', [(1, 2), ([3], 4)], 5]
>>>L[1]
[(1, 2), ([3], 4)]
>>>L[1][1]
([3], 4)
>>>L[1][1][0]
[3]
>>>L[1][1][0][0]
3
```

30

## Shared references

```
>>>X = [1,2,3]
>>>L = ['a', X, 'b']
>>>D = {'x':X, 'y':2}

>>>X[1] = 'surprise'
>>>L
['a', [1, 'surprise', 3], 'b']
>>>D
{'x': [1, 'surprise', 3], 'y':2}
```

31

## comparisons

```
>>>L1 = [1, ('a', 3)]
>>>L2 = [1, ('a', 3)]
>>>L1==L2, L1 is L2
(1, 0)

>>>L1 = [1, ('a', 3)]
>>>L2 = [1, ('a', 2)]
>>>L1 < L2, L1==L2, L1>L2
(0, 0, 1)
```

32

## Object truth values

Object	value
"hello"	True
""	False
[]	False
{}	False
1	True
0.0	False
None	false

33

## Assignment creates references, not copies

```
>>>L=[1,2,3]
>>>M=['x', L, 'y']
>>>L[1]=0
>>>M
['x', [1,0,3], 'y']

>>>L=[1,2,3]
>>>M=['x', L[:], 'y']
>>>L[1]=0
>>>M
['x', [1,2,3], 'y']
```

Makes a copy

34