Basic Python types

Reuven M. Lerner, PhD
reuven@lerner.co.il

Basic types

- Python comes with many useful data types
- Knowing how and when to use them is key
- You can use just these, most of the time
- Objects and larger data structures are built out of these
- Find out the type of a variable with `type(x)`

None

- Like nil in Lisp, or NULL in SQL
- Different from False!
  - (But false when forced to be boolean)
- Also distinct from empty string and 0
- Test equality with “is None” (or == None)

True and False

- Don't use 1 and 0! That's a weird C thing
- All objects in Python are true in a boolean context, except:
  - None, False
  - 0 of a numeric type (e.g., 0.0)
  - empty sequence or mapping

Numbers

- Common: int, float
- Rare: long, complex
- Yes, complex numbers are built in:

```
>>> a = 5 + 6j

>>> b = 6 + 7j

>>> a + b
```

5

Floats

```
type(1)      #    int

type(1.0)    #    float

float("1")   #    1.0

float(1.5)   #    1.5

10e+3        #    10000.0

0.1 + 0.7    #    0.7999999999999999
```

6

Numeric operations

- + (addition), - (subtraction)
- * (multiplication), / (division)
- % (modulus), ** (exponentiation)
- 3/2 ==> 1 (integer math!)
- 3/2.0 ==> 1.5 (floating-point math)
- 3 // 2.0 ==> 1.0 (truncated floating-point math)

For C programmers

```
x = 5
  x += 1
x
  6
x++
  SyntaxError
```

8

Creating ints

- Create integers from strings with int()

```
int("50")

  50

int("50", 8)

  40

int("50", 16)

  80

int("ab50", 16)

  43856
```

Or, use prefixes

```
0b100100

  36

050

  40

0x50

  80

0xab50

  43856
```

Output in different bases

```
bin(100)

  '0b1100100'

oct(100)

  '0144'

hex(100)

  '0x64'
```

Strings

- Strings are a first-class object in Python
- There are no characters — only strings with one element
- Very important: Strings are immutable
- In Python 2, strings are sequences of bytes
- In Python 3, strings have an encoding!

Special characters

`\a`	ASCII Bell (BEL)
`\b`	ASCII Backspace (BS)
`\f`	ASCII Formfeed (FF)
`\n`	ASCII Linefeed (LF)
`\r`	ASCII Carriage Return (CR)
`\t`	ASCII Horizontal Tab (TAB)
`\v`	ASCII Vertical Tab (VT)
`\ooo`	ASCII character with octal value ooo
`\xhh`	ASCII character with hex value hh

Backslashes

- Use a backslash to "escape" characters:

```
s = 'abc\'def' # \' gives a literal quote

s = "abc'def"  #  No backslash needed

s = 'abc\ndef' # \n is newline

s = 'abc\\ndef'# literal \, followed by n
```

14

Strings

Single quote	`'Reuven'`
Double quote	`"Reuven"`
Raw string	`r'Reuven\n'`
Unicode string	`u'שלום'`
Triple-quoted string	`'''Reuven` `Lerner'''`

String operations

- Concatenate with +

```
"hello" + "world"

"hello" "world"      # Don’t!

len                  # Builtin function

index, find, strip   # String methods
```

str.strip()

```
s = '    abc   def   ghi   '

>>> s.strip()    # all whitespace, both sides

'abc   def   ghi'

>>> s.lstrip()   # all whitespace, left side

'abc   def   ghi   '

>>> s.rstrip()   # all whitespace, right side

'    abc   def   ghi'
```

17

string in string

- You can use “in” to locate a string in another string:

```
>>> 'a' in 'abc'
  True
>>> 'ab' in 'abc'
  True
>>> 'cba' in 'abc'
  False
```

18

Slicing strings

First element	`s[0]`
Second element	`s[1]`
Final element	`s[-1]`
First 5 elements	`s[0:5] or s[:5]`
Final 5 elements	`s[-5:]`

Old-style interpolation

- Double quotes aren’t like Perl/PHP/Ruby
- Interpolation with % operator:

```
"hello, %s" % "Reuven"

  'hello, Reuven'

"Hi, %s %s" % ("R", "L")

  'Hi, R L'
```

20

str.format

```
'first {0}, last {1}'.format("Reuven",
                             "Lerner")


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

   'abc a b c'
```

21

Or, in Python 2.7 +

```
'first {}, last {}'.format("Reuven",
                           "Lerner")


'abc {} {} {}'.format('a', 'b', 'c')

   'abc a b c'
```

22

Keyword arguments

```
'first {a}, last {b}'.format(a="Reuven",
                       b="Lerner")
```

23

str.format examples

- A guide to str.format is now available at
- http://pyformat.info/
- It contains lots of great examples of what you can do with str.format

Simple replacement

```
>>> 'reuven'.replace('e', 'z')
'rzuvzn'
```

- This replaces strings, not characters
- No, you cannot use regexps here

Multiply strings

```
'a' * 5

    'aaaaa'

'abc' * 3

    'abcabcabc'

3 * 'abc'

    'abcabcabc'
```

26

for loops

```
for VAR in SEQUENCE:
 do_something_with(VAR)
```

- Sequences are lists, tuples, and strings (among others)
- The iteration variable remains defined after the loop exits!

Example

```
letters = 'abc'

for letter in letters:
     print letter

a
b
c
```

With index

```
letters = 'abc'

for index, letter in enumerate(letters):
    print "{}:{}".format(index, letter)

0: a
1: b
2: c
```

29

Loop n times

```
for index in range(3):
    print index

0
1
2
```

30

while loops

```
x = 5
while x > 0:
    print "[{}] Hello".format(x)
    x = x - 1
```

31

Loop control flow

`break`	Exits the loop
`continue`	Exits the current iteration (but continues with the loop, if more iterations remain)
`pass`	No-op (placeholder for future code)
`else`	After the loop body, executes if exit was not from break

32

```
import random
x = random.randint(1,100)
for i in range(10):
    if i < x:
        print "{} is too low".format(i)
    elif i == x:
        print "Guessed {} correctly".format(i)
        break
    else:
        print "Missed: i= {}, x ={}".format(i, x)
else:
    print "The number {} was not guessed".format(x)
```

Print without newline

- You can force print *not* to skip to the next line by putting a comma (,) at the end of the line.

```
print "a", ; print "b"

    a b

print "a" ; print "b"

    a

    b
```

34