Introduction to Python

Chen Lin [1]
Modified by Na Meng

Overview

• Development Environments
• Global and Local Variables
• Data Types/Structures
• Control Flow
• File I/O
• Functions
Development Environments

1. PyDev with Eclipse
2. Komodo
3. Emacs
4. Vim
5. TextMate
6. Gedit
7. Idle
8. PIDA (Linux)(VIM Based)
9. NotePad++ (Windows)
10. BlueFish (Linux)
Python Interactive Shell

% python
Python 2.6.1 (r261:67515, Feb 11 2010, 00:51:29)
[GCC 4.2.1 (Apple Inc. build 5646)] on darwin
Type "help", "copyright", "credits" or "license" for more information.

>>> 2+3*4
14
>>> name = "Andrew"
>>> name
'Andrew'
>>> print ("Hello", name)
Hello Andrew
>>>
Global and Local Variables [2]

• An example
def f():
    print(s)
s = "I hate spam"
f()
• s is a global variable
• What is the output?

Global and Local Variables

• Another example
def f():
    s = "Me too."    s is a local variable.
    print(s)
s = "I hate spam." s is a global variable.
f()
    print(s)
• What is the output?
Global and Local Variables

• A third example
  
  ```python
def f():
    print(s)
    s = "Me too."
    print(s)
    s = "I hate spam."
    f()
    print(s)
```

• What is the output?

Global and Local Variables

• UnboundLocalError: local variable 's' referenced before assignment

• Python assumes that we want a local variable due to the assignment to s in f()

• How can we tell Python that we want to use the global variable?
Global and Local Variables

• Correction
def f():
    global s
    print(s)
    s = “Me too.”
    print(s)
    s = “I hate spam.”
f()
    print(s)

Data Types/Structures

• List
• String
• Tuple
• Dictionary
• Set
List

A compound data type:
[0]
[2.3, 4.5]
[5, "Hello", "there", 9.8]
[]

Use len() to get the length of a list

>>> names = ["Ben", "Chen", "Yaqin"]
>>> len(names)
3

Use [ ] to index items in the list

>>> names[0]
'Ben'
[0] is the first item.
>>> names[1]
'Chen'
[1] is the second item
>>> names[2]
'Yaqin'
... Out of range values
>>> names[3]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: list index out of range
Use [ ] to index items in the list

>>> names[-1]
'Yaqin'
>>> names[-2]
'Chen'
>>> names[-3]
'Ben'

Negative values go backwards from the last element.

Strings share many features with lists

>>> smiles = "C(=N)(N)N.C(=O)(O)O"
>>> smiles[0]
'C'
>>> smiles[1]
'('}
>>> smiles[-1]
'O'
Strings share many features with lists

```python
>>> smiles[1:5]
' (=N)'
>>> smiles[10:-4]
'C(=O)'
```

Use “slice” notation to get a substring

String Methods: find, split, ...

```python
smiles = "C(=N)(N)N.C(=O)(O)O"
>>> smiles.find("(O)")
15
>>> smiles.find("."")
9
>>> smiles.find(".", 10)
-1
>>> smiles.split(".")
['C(=N)(N)N', 'C(=O)(O)O']
```

Use “find” to find the start of a substring.

Start looking at position 10.

Split the string into parts with “.” as the delimiter.
String operators: in, not in

if "Br" in “Brother”:
    print "contains brother"

email_address = “clin”
if "@" not in email_address:
    email_address += "@brandeis.edu"

Lists are mutable

>>> ids = ["9pti", "2plv", "1crn"]
>>> ids.append("1alm")
append an element
>>> ids
['9pti', '2plv', '1crn', '1alm']
>>> ids.extend(L)
append a list
Extend the list by appending all the items in the given list; equivalent to a[len(a):]
= L.
Lists are mutable

>>> del ids[0]  # remove an element
>>> ids
['2plv', '1crn', '1alm']

>>> ids.sort()  # sort by default order
>>> ids
['1alm', '1crn', '2plv']

>>> ids.reverse()  # reverse the elements in a list
>>> ids
['2plv', '1crn', '1alm']

Lists are mutable

>>> ids.insert(0, "9pti")  # insert an element at some specified position.
>>> ids
['9pti', '2plv', '1crn', '1alm']

(Slower than .append())
Zipping Lists Together

```python
>>> names
['ben', 'chen', 'yaqin']

>>> gender = [0, 0, 1]

>>> zip(names, gender)
[('ben', 0), ('chen', 0), ('yaqin', 1)]
```

Tuple: Like Immutable List

```python
>>> yellow = (255, 255, 0) # r, g, b
>>> one = (1,)
>>> yellow[0]
255
>>> yellow[1:]
(255, 0)
>>> yellow[0] = 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```
Tuple: Like Immutable List

Very common in string interpolation:

```python
>>> "%s lives in %s at latitude %.1f" %
("Andrew", "Sweden", 57.7056)
'Andrew lives in Sweden at latitude 57.7'
```

Dictionary

- Dictionaries are lookup tables
- They map from a "key" to a "value".

```python
symbol_to_name = {
    "H": "hydrogen",
    "He": "helium",
    "Li": "lithium",
    "C": "carbon",
    "O": "oxygen",
    "N": "nitrogen"
}
```
Dictionary

- Duplicate keys are not allowed
- Duplicate values are just fine
- Keys can be any immutable value
  numbers, strings, tuples, not list, dictionary, set, ...

```python
>>> symbol_to_name["C"]
'carbon'
>>> "O" in symbol_to_name, "U" in symbol_to_name
(True, False)
```

Set

- Sets are lists with no duplicate entries

```python
>>> a = set(["Jake", "John", "Eric"])
>>> b = set(["John", "Jill"])
>>> a.intersection(b)
set(["John"])  # set(['John'])
>>> a.difference(b)
set(["Jake", 'Eric'])
>>> a.symmetric_difference(b)
set(["Jill", 'Jake', 'Eric'])
```
Control Flow

Things that are False
• The boolean value False
• The numbers 0 (integer), 0.0 (float) and 0j (complex).
• The empty string "".
• The empty list [], empty dictionary {} and empty set set().

Control Flow

Things that are True
• The boolean value True
• All non-zero numbers.
• Any string containing at least one character.
• A non-empty data structure.
Examples

```python
>>> smiles = "BrC1=CC=C(C=C1)NN.CI"
>>> bool(smiles)
True
>>> not bool(smiles)
False
```

If-statement

```python
>>> if not smiles:
...     print "The SMILES string is empty"
...
• Indentation is important to indicate program structure
• The “else” case is always optional
```
"elif" is used to chain subsequent tests

```python
>>> mode = "absolute"
>>> if mode == "canonical":
...     smiles = "canonical"
...     elif mode == "isomeric":
...         smiles = "isomeric"
...     elif mode == "absolute":
...         smiles = "absolute"
...     else:
...         raise TypeError("unknown mode")
... >>> smiles
```

"raise" is the Python way to raise exceptions

Boolean Logic

Python expressions can have "and"s and "or"s:

```python
if (ben <= 5 and chen >= 10 or chen == 500 and ben != 5):
    print "Ben and Chen"
```
For-statement

>>> names = ["Ben", "Chen", "Yaqin"]
>>> for name in names:
...     print name
...  Ben
  Chen
  Yaqin

Tuple Assignment in For Loops

data = [ ("C20H20O3", 308.371),
        ("C22H20O2", 316.393),
        ("C24H40N4O2", 416.6),
        ("C14H25N5O3", 311.38),
        ("C15H20O2", 232.3181)]
for (formula, mw) in data:
    print "The molecular weight of %s is %s" % (formula, mw)
Tuple Assignment in For Loops

The molecular weight of C20H20O3 is 308.371
The molecular weight of C22H20O2 is 316.393
The molecular weight of C24H40N4O2 is 416.6
The molecular weight of C14H25N5O3 is 311.38
The molecular weight of C15H20O2 is 232.3181

break, continue

>>> for value in [3, 1, 4, 1, 5, 9, 2]:
...     print "Checking", value
...     if value > 8:
...         print "Exiting for loop"
...         break
...     elif value < 3:
...         print "Ignoring"
...         continue
...     print "The square is", value**2
...
Range()

- “range” creates a list of numbers in a specified range
- `range([start,] stop[, step])` -> list of integers
- When step is given, it specifies the increment (or decrement).

```python
>>> range(5)
[0, 1, 2, 3, 4]
>>> range(5, 10)
[5, 6, 7, 8, 9]
>>> range(0, 10, 2)
[0, 2, 4, 6, 8]
```
Reading Files

```python
>>> f = open("names.txt")
>>> f.readline()
'Yaqin
'
```

Read Lines in a Loop

```python
>>> lst = [ x for x in open("text.txt","r").readlines() ]
```
File Output

```python
input_file = open("in.txt")
output_file = open("out.txt", "w")
for line in input_file:
    output_file.write(line)
```

- `"w"` = “write mode”
- `"a"` = “append mode”
- `"wb"` = “write in binary”
- `"r"` = “read mode” (default)
- `"rb"` = “read in binary”
- `"U"` = “read files with Unix or Windows line endings”

Functions

- Python provides many built-in functions like print(), etc.
- But you can also create your own functions—user-defined functions
```python
# Function definition is here
def printme(str):
    print str
    return;

# Now you can call printme function
printme("I'm first call to user defined function!")
printme("Again second call to the same function")
```

### Nested Functions

```python
def outer(num1):
    def inner_increment(num1):
        return num1 + 1
    num2 = inner_increment(num1)
    print(num1, num2)
```

Program Assignment 1

• An Evaluator for Logical Expressions Written in Postfix Notation
  – Write a Python program that computes the value of logic expressions provided in postfix notation.
  – E.g., given the string “0!1&” (infix: “!0&1”), where “0” means “False” and “1” means “True”. Your calculator will compute “1” as the answer
  – All strings provided will be valid

Program Assignment 1

• The logic manipulation operators are:
  "!" logical NOT  RIGHT associative
  "&" logical AND  LEFT associative
  "/" logical NOT EQUAL  LEFT associative
  "=" logical EQUAL  LEFT associative
  "|" logical OR  LEFT associative
Program Assignment 1

• Your calculator will use stack to compute/store all intermediate computations
• You will implement your own push and pop stack operations
• The ONLY library or built-in method that you can use is len()
• Download python 3.5.2 from www.python.org

Program Assignment 1

• Write your program following the structure and specs given in the assignment descriptions
  – Define your own stack structure
  – Use the specified variables in the template
  – Define nested functions
  – ...
• Due date: 10/6/16, 12:30pm
• Submit both an electronic copy via canvas, and a hard copy in-class
Reference
