Lecture 8

CORE ELEMENTS PART VI:

DICTIONARIES
&
HANDLING ERRORS

A word (or more) of Wisdom

It's the journey, not the destination.

Could not find the author

Sometime it's the journey that teaches you a lot about your destination

Drake (Rapper)

Overview

- Review on Iterables (list, strings, tuples) and loops
- Dictionaries
  - A new data type
- Handling Exceptions
  - try-except-else-finally
- Raising exceptions
  - raise

Iterating through Lists

- We have seen two ways to traverse an iterable (list, string, tuple).
- Which one should we choose?

The variable `val` will take the value `'a'`, then `'b'`, ...

READ

Code 1

```
lst = ['a','b','c','d']
for val in lst:
    print val.upper()
```

The variable `val` will take the value 0, then 1, ...

READ/ASSIGN

Code 2

```
lst = ['a','b','c','d']
for val in xrange(len(lst)):
    lst[val] = lst[val].upper()
```

Iterating through Lists

- For the second code snippet, use a better name for the variable containing the index in the array.
  - index, pos, row, col, …

The variable `val` will take the value `'a'`, then `'b'`, …

READ

Code 1

```
lst = ['a','b','c','d']
for val in lst:
    print val.upper()
```

The variable `index` will take the value 0, then 1, …

READ/ASSIGN

Code 2

```
lst = ['a','b','c','d']
for index in xrange(len(lst)):
    lst[index] = lst[index].upper()
```

Iterating through Lists

- When using while remember to:
  - Initialise the index variable
  - Change the index variable (to avoid infinite loop)

Code 2

```
lst = ['a','b','c','d']
for index in xrange(len(lst)):
    lst[index] = lst[index].upper()
```

Code 3

```
lst = ['a','b','c','d']
index = 0
while index < len(lst):
    lst[index] = lst[index].upper()
    index = index + 1
```

Dictionaries

9

ANOTHER PYTHON BUILT-IN TYPE

Caesar Cipher

- We have seen it is easy to hack the Caesar cipher using a computer
  - It's a simple form of substitution cipher
  - At most size of the alphabet (n) possibilities
  - Can use brute force
- Consider the more general form of the substitution cipher
  - At most (n! ≅ 4.03 E^{26}) at least 300 years of computation of the fastest machine (Tianhe-2) using brute force

Alphabet	a	b	c	d	e	f	g	…
Key	c	a	z	b	g	d	x	…

Be Cleverer than Brute Force

- Frequency analysis:
  - A way to identify symbols with similar frequencies
- Which Data representation?
  - [('a', 8.167), ('b', 1.492)...] ?
- How to create the frequency table of the encrypted message?
  - Is our data representation efficient?
- We may need a new data structure

Letter	Relative frequency in the English language
a	8.167%
b	1.492%
c	2.782%
d	4.253%
e	12.702%
f	2.228%
g	2.015%
h	6.094%
i	6.966%
j	0.153%
k	0.772%
l	4.025%
m	2.406%
n	6.749%
o	7.507%
p	1.929%
q	0.095%
r	5.987%
s	6.327%
t	9.056%
u	2.758%
v	0.978%
w	2.360%
x	0.150%
y	1.974%
z	0.074%

Dictionaries

- A new built-in data structure
- Is MUTABLE
- Map a key to a value
- Keys **must** be **immutable** objects
  - int, float, string, tuple
- Values can be any object
  - list, dictionary, int, float, string, …

How to Create/Use a Dictionary

- General form

```
dict_name = {key1:value1, key2:value2, …}
```

- Retrieving a value:

```
dict_name [key]
```

- adding/modifying a value:

```
dict_name [key] = value
```

- deleting a value:

```
del dict_name [key]
```

Dictionaries

- A fast way to retrieve a value in a collection of values
  - Better than list, if we don't know the index of the element to retrieve
- Adding an new element to a list is faster than adding it to a dictionary
  - As long as the list is not sorted

Dictionaries

- How to apply this structure to our decoding problem?
  - What would be the keys?
  - What would be the values?

Exceptions

16

HANDLING ERRORS

Why Use Exceptions?

To Jump around arbitrarily large chunks of a program

- Error Handling
  - When an error is detected at runtime
- Event Notification
  - Can also be used to signal a valid condition
- Special-case Handling
  - When some condition may happen rarely

A Simple Program

- see code ‘`user division first attempt`’

A Simple Program

- Dealing with user inputs
  - What if the user enters a invalid number format
  - An error occurs: ValueError
  - The program crashes

- Handling errors with try-except

Code

```
try:
    <statements A>

except ErrorType:
    <statements B>
```

  - see code ‘`user division second & third attempt`’

A Simple Program

- Handling many different type of errors with try-except
  - **At most one** handler will be executed

Code

```
try:
    <statements A>
except ErrorTypeOne:
    <statements B>
except ErrorTypeTwo:
    <statements C>
except (ErrorTypeThree, ErrorTypeFour, ErrorTypeFive) :
    <statements D>
```

- see code ‘`user division final attempt`’

try-except-else

- The `try ... except` statement has an <u>optional</u> `else` *clause*,
  - must follow all except clauses.
  - It is useful for code that must be executed if the try clause **<u>does not</u>** raise an exception.

Code

```
try:
    f = open(arg, 'r')
except IOError:
    print 'cannot open', arg
else:
    print arg, 'has', len(f.readlines()), 'lines'
    f.close()
```

try-finally

- The try statement has another optional clause
  - intended to define clean-up actions that must be executed under all circumstances.
  - A `finally` *clause* is always executed before leaving the try statement, whether an exception has occurred or not

Code

```
try:
    f = open(arg, 'r')
except IOError:
    print 'cannot open', arg
else:
    print arg, 'has', len(f.readlines()), 'lines'
finally:
    f.close()
```

raise

- The `raise` statement allows the programmer to force a specified exception to occur.

Code

```
def myDiv(numerator, divisor):
    if divisor == 0:
        raise ValueError('argument divisor must not be 0')
    else:
        return numerator/divisor
```

raise

- If you need to determine whether an exception was raised but don't intend to handle it, a simpler form of the `raise` statement allows you to re-raise the exception

Code

```
try:
    f = open(arg, 'r')
except IOError:
    print 'cannot open', arg
    raise
```

Summary

- We have seen how to:
  - handle exception
  - raise exception

Reading

- Handling Exceptions
  - http://docs.python.org/2/tutorial/errors.html
- built-in Exceptions list
  - http://docs.python.org/2.7/library/exceptions.html#bltin-exceptions

Odds and Ends

- When an exception occurs, it may have an associated value, also known as the exception's *argument*.
- The presence and type of the argument depend on the exception type.

Code

```
try:
    f = open(arg, 'r')
except IOError as instExc: # instExc is a variable that
                           # can be inspected
    print 'cannot open', arg
    print 'errors is:', instExc
else:
    print arg, 'has', len(f.readlines()), 'lines'
    f.close()
```