Introduction to Dictionaries

Professor Hugh C. Lauer

CS-1004 — Introduction to Programming for Non-Majors

(Slides include materials from *Python Programming: An Introduction to Computer Science*, 2nd edition, by John Zelle and copyright notes by Prof. George Heineman of Worcester Polytechnic Institute)

Dictionary

- **Important data type in *Python***
- **Read and study §11.6**
  - And all of Chapter 11!
- **Definition:– "Dictionary"**
  - A *Python* data type for collections, capable of storing and retrieving key-value pairs, …
    - … where keys and values can be *any* type, …
      - … data is unordered!
- **Called a *hash table* in most other languages**
  - Not a built-in data type (in those languages)

HW4

- **"Write the function `PlayManyGames(n)` needed by Homework #HW4.**
- **This must call your previous function `PlayOneGame()`**
- **For each call to `PlayOneGame()`, *use the result to index into a list and increment the value stored that location in the list*."**

- **A hassle!**

- **Using indexing into list to store values not directly associated with indexes!**
  - Win in 1 roll ⇔ winList[0]
  - Win in 2 rolls ⇔ winList[1]
  - …
  - Win in *k* rolls ⇔ winList[k-1]
- **Have to create empty list entries for long games**

Lab 4 and HW4 (continued)

- **Would really like something like:–**

```
w, nThrows = playOneGame()

if nThrows in W:
   W[nThrows] += 1
else:
   W[nThrows] = 1
```

  - Remember that **`PlayOneGame()`** returns a tuple (win/loss, # of throws)

- **With dictionaries, we can do this!**

Using Dictionaries

- **Creating a dictionary** **Also called a "mapping"**

  `D = {}`

  - Remember:–

    Tuple ⇒ parentheses — ()

    Lists ⇒ square brackets — [ ]

    Dictionaries ⇒ curly brackets — { }

- **Adding an item to dictionary**

  `D[key] = value`

  - **`key`** may number, string, tuple
    - i.e., anything 'immutable'
  - **`value`** can be anything at all!
  - **`D[key]`** acts like a variable

Getting item from Dictionary

```
variable = D[key]
```

- **Very fast, efficient "lookup" of `key`**
- **Uses technique in computing called "hashing"**
  - **`key`** is scrambled
  - Scrambled value used to index into big list
  - Easy to find by simple search from there
- **Does *not* preserve order**
  - Organized internally for speed of access

Examples

Operators, functions, & methods

- `key in dict`
  - `True` if `key` is in the dictionary, `False` if not
- `dict.keys()`
  - List of `keys` in the dictionary
- `dict.values()`
  - List of `values` in dictionary
- `dict.items()`
  - List of `(key,value)` pairs
- `dict.get(key,default)`
  - Returns `dict[key]` if present, `default` otherwise
- `del dict[key]`
  - Deletes `key` and `dict[key]` from dictionary
- `for k in dict:`
  - Loop over all keys in the dictionary
  - Internal order of storage

More Examples

Questions?

What can we do with dictionaries

- **Store stuff!**
  - Easily
  - So it is easily retrievable
- **Organize data by different criteria**
  - Pull out non-numeric data
- **…**

Example – HW#5

- `dict` is a dictionary of `word-count` pairs
  - Indexed by `word`

1. Read file, split into words, strip, lower-case

2. For each `word` of input
   - If `word` in dictionary, increment `dict[word]`
   - If not, `dict[word] = 1`

3. Get list of unique words
   - `uniqueWords = list(dict.keys())`

4. Sort
   - `uniqueWords.sort()`

See §11.6.3

5. Get counts
   - ```
     for word in uniqueWords():
         write(dict[word], word)
     ```
     ```

How to read lines from a file

```
f = open(filename, mode)
```

- **`filename`** is a string
- Relative to current directory!
- **`mode`** should be 'r' (i.e., read)

```
for line in f:
    # process line here

f.close()  # finished with file!
```

- **Each line is a string ending in `'\n'`**

Extracting words from string

- **Let `line` be the string**

'brought forth on this continent, a new nation,\n'

  - (without the enclosing quotes)

- **Then `line.split()` returns the list:–**

```
['brought', 'forth', 'on', 'this',
'continent,', 'a', 'new', 'nation,']
```

  - I.e., partitioned at white-space

Note embedded commas

- **Definition — white-space**
  - Space, tab, line feed, newline, form feed, and vertical tab
  - See *Python* documentation > *Python* standard library > Text, §6.1

Note: `line.split()` method is more general
Can split at any set of characters!

How to read and process lines from multiple files

```
dict = {}



while True:
   filename = input('Enter filename:-')
   if len(filename) == 0:
         break
   OpenAndRead(filename, dict)

outputFile = input('Enter output filename:- ')

OpenAndWrite(outputFile, dict)
```

How to read and process lines from multiple files (alternative)

```
dict = {}

outputFile = input('Enter output filename:- ')

while True:
   filename = input('Enter filename:-')
   if len(filename) == 0:
         break
   OpenAndRead(filename, dict)



OpenAndWrite(outputFile, dict)
```

Questions?

Extra Credit

- Read file names from command line
- What in *&^%$ is a *Command Line*?

Command Lines

- **Windows, Macintosh, and Linux all have "command prompt" windows**

- **Command line format:–**

```
verb arg1 arg2 arg3 ...
```

- **`verb` is name of a program that carries out command action**

- **Each `arg` is a string**
  - Delimited by spaces
  - **`arg0`** is the **`verb`**!

- **Meaning:– Apply verb to the list of arguments**
  - Don't return till finished!

Operating System's Responsibility

- **Pick apart command line**
  - Create a list of strings called "**`argv`**"
  - Number of items in list is "**`argc`**"
- **Load the program named verb (i.e., `arg0`) into a new, clean memory space.**
- **Call the function with the name `main()`, passing `argc` and `argv` as arguments**
  - i.e.

```
main(argv, argc)
```

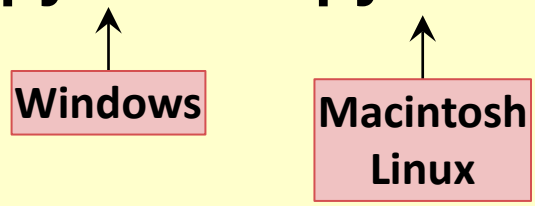
- **Wait till it returns, then continue with next command line**

Starting programs in a GUI

- **User "opens" a file or document**
- **OS or Window manager consults list of file types**
  - Finds program that opens the type of this file or document
  - Based on "extension" of file name
- **(Essentially) constructs a command line!**
  - As if it had been typed
  - Name of **`verb`** (i.e., program) as **`arg0`**
  - Name of file to be opened as **`arg1`**
  - Other arguments as needed
- **Calls `main()` function of the program!**

What about Python?

- Command must be `python` or `python3`



- Command line must be

```
python HW5.py outputFile InFile1 InFile2 …
```

- Getting the arguments into *Python*

```
import sys.argv
```

`sys.argv` is a list containing the strings:–

```
['HW5.py', 'outFile', 'InFile1', 'InFile2', …]
```

Using sys.argv

- After everything else is working:– (!!)

- ```
  If len(sys.argv) > 1:
   outputFileName = sys.argv[1]
  ```

- ```
  i = 2
  dict = {}
  while len(sys.argv) > i:
      inputFileName = sys.argv[i]
      readAndProcess(inputFileName, dict)
  ```

- ```
  prepareAndWrite(outputFileName, dict)
  ```
  ```

Questions?

string.format()

- **A method for formatting output strings**
  - To keep columns aligned
  - To manage 'field widths'
  - To manage #'s of significant digits in floats
  - Etc.

- **Let T be a *template***
  - Structure of template to be described below

- **Then**

  `T.format(value, value, value, …)`
  - Makes a copy of **T**
  - Fills in the value arguments in the "slots" of new copy of **T**
  - Formats each value argument according to specifications in each "slot"

Template

- **See §5.8.2 of textbook**
- **See 6.1.3 of *Python Documentation***
  - "Format String Syntax"
- **Similar to formatting tools in other high-level languages**
- **Example:–**

```
T = "Hello {0} {1}, you may have won ${2}"
```

- `T.format('Mr.', 'Smith', 1000)`

```
'Hello Mr. Smith, you may have won $1000'
```

Other formatting examples

- `T = 'left justification: {0:<5}'`
- `T.format("hi!")`

- `T = 'right justification: {0:>5}'`
- `T.format("lo!")`

- Numbers with decimals
- Decimal precisions
- Commas in numbers

- Locale-specific formats

References

- **Textbook, §5.8.2**
- **Python 3.4.2 Documentation > Python Standard Library > Text**
  - §6.1.2, 6.1.3
- **Online help**

Questions?