DM502
Programming A

Peter Schneider-Kamp

petersk@imada.sdu.dk

http://imada.sdu.dk/~petersk/DM502/

TUPLES

Tuples as Immutable Sequences

- tuple = immutable sequence of values
- like lists, tuples are indexed by integers

- tuples can be enclosed in parentheses "(" and ")"
- Example:

```
t1 = "D", "o", "u", "g", "l", "a", "s"
t2 = (65, 100, 97, 109, 115)
t3 = 42,        # or (42,) - but not (42)
```

- tuples can be created from sequences using tuple(iterable)
- Example:

```
t1 == tuple("Douglas")
tuple(["You", 2]) == ("You", 2)
```

Tuples as Immutable Sequences

- tuple = immutable sequence of values
- like lists, tuples are indexed by integers

- tuples can be accessed using indices and slices
- Example:

```
t = "D", "o", "u", "g", "l", "a", "s"
t[3] == "g"
t[1:3] == ("o", "u")
```

- tuples cannot be changed, but they can be concatenated
- Example:

```
u = ("d",) + t[1:]
```

Tuple Assignment

- remember, how to exchange two values:
  - Solution 1 (new variable): `z = y; y = x; x = z`
  - Solution 2 (parallel assign.): `x, y = y, x`
- now, we see that this is a tuple assignment
- assignment to a tuple is assignment to each tuple element
- works not only with other tuple, but with any sequence
- Example:

```
x, y, z = [23, 42, -3.0]
name = "Peter Schneider-Kamp"
first, last = name.split()
```

Tuples as Return Values

- useful to return more than one value in a function
- but functions only return one value
- tuples can be used to contain multiple values
- Example 1: built-in function divmod(x,y)

```
t = divmod(10, 3)
print t
quot, rem = divmod(101, 17)
```

- Example 2: extract username, hostname, and domain

```
def decompose(email):
    username, rest = email.split("@")
    rest = rest.split(".")
    return username, rest[0], ".".join(rest[1:])
```

Variable-Length Argument Tuples

- functions can take a variable number of arguments
- arguments are passed as one tuple (*gather*)
- Example 1: function that works similar to print statement

```
def printf(*args):          # * indicates variable arguments
    for arg in args:        # iterates through tuple
        print arg,          # prints one argument
    print                   # prints new line
```

- Example 2: prints all arguments n times

```
def printn(n, *args):
    for arg in args * n:
        print arg
```

Tuples instead of Arguments

- tuples cannot directly be used instead for normal parameters
- Example:

```
t = (42, 23)
print divmod(t)        # gives TypeError
```

- using "*" we can declare that a tuple should be *scattered*
- Example:

```
print divmod(*t)       # prints (1, 19)
```

Lists and Tuples

- built-in function zip() combines two sequences
- Example 1:

```
zip([1, 2, 3], ["c", "b", "a"]) == [(1, "c"), (2, "b"), (3, "a")]
```

- Example 2:

```
zip("You", "suck!") == [("Y", "s"), ("o", "u"), ("u", "c")]
```

- iteration through list of tuples using tuple assignment
- Example:

```
t = [(1, "c"), (2, "b"), (3, "a")]
for num, ch in t:
    print "we have paired", num, "and", ch
```

Lists and Tuples

- with zip(), for loop, and tuple assignment we can iterate through two sequences in parallel
- Example 1: sum of product of elements (*dot product*)

```
def dot_product(x, y):
    res = 0
    for a, b in zip(x, y):
        res += a*b
    return res
dot_product([1, 4, 3], [4, 5, 6])
```

- Example 2: the same shorter …

```
def dot_product(x, y):
    return reduce(lambda x, y: x + y[0] * y[1], zip(x, y), 0)
```

Dictionaries and Tuples

- dictionaries return a list of tuples with the items() method
- Example: d = {"a" : 3, "b" : 2, "c" : 1}
  d.items() == [("a", 3), ("c", 1), ("b", 2)]

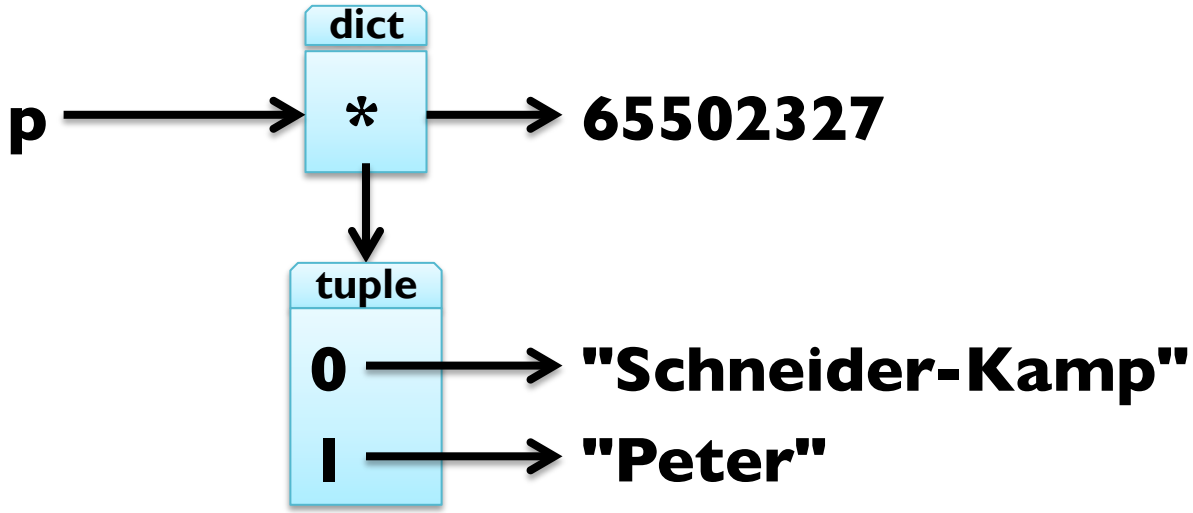
- tuples can also be used to create new dictionary using dict()
- Example: t = [("a", 3), ("c", 1), ("b", 2)]
  dict(t) == {"a" : 3, "b" : 2, "c" : 1}

- combine with zip() for easy dictionary generation
- Example: d = dict(zip("abcdefg", range(7,0,-1)))

- with tuple assignment and for loop, easy traversal
- Example: for key, val in d.items(): print key, val

Dictionaries and Tuples

- tuples can be used as dictionary keys (they are immutable)
- Example: p = {}; first = "Peter"; last = "Schneider-Kamp"
  p[last, first] = 65502327
- traversal by for loop and tuple assignment
- Example 1: for last, first in p: print first, last, p[last, first]
- Example 2: for (last, first), num in p.items(): print last, first, num



Dictionaries and Tuples

- tuples can be used as dictionary keys (they are immutable)
- Example: p = {}; first = "Peter"; last = "Schneider-Kamp"
  p[last, first] = 65502327
- traversal by for loop and tuple assignment
- Example 1: for last, first in p: print first, last, p[last, first]
- Example 2: for (last, first), num in p: print last, first, num

Comparing Tuples

- comparing tuples same as comparing any sequence
- like with strings, sequences are compared lexicographically
- Example: (3,) > (2, 2, 2)
  (1, 2, 3, 4, 5) < (1, 2, 3, 5,5)
- tuples can be used to sort lists after arbitrary criteria
- Example: sort list of words after their length, shortest last

```
def sort_by_length(words):
    t = []; res = []
    for word in words:          t.append((len(word), word))
    t.sort(reverse=True)
    for length, word in t:      res.append(word)
    return res
```

Comparing Tuples

- comparing tuples same as comparing any sequence
- like with strings, sequences are compared lexicographically
- Example: (3,) > (2, 2, 2)
  (1, 2, 3, 4, 5) < (1, 2, 3, 5,5)
- tuples can be used to sort lists after arbitrary criteria
- Example: sort list of words after their length, shortest last

```
def sort_by_length(words):
    t = map(lambda x: (len(x), x), words)
    t.sort(reverse=True)
    return map(lambda pair: pair[1], t)
```

Sequences of Sequences

- most sequences can contain other types of sequences
- string is an exception, as it only contains characters
- can always use a list of characters instead of string
- lists usually preferred to tuples (they are mutable)
- in some situtations, tuples more often used:
  1. tuples are more "easy" to construct, e.g. return n, n**2
  2. tuples can be dictionary keys (they are immutable)
  3. tuples are safer due to "aliasing", so use them e.g. as sequence arguments to functions
- methods sort() and reverse() not available for tuples
- use functions sorted(iterable) and reversed(iterable) instead

Debugging Shape Errors

- lists, dictionaries, and tuples are *data structures*
- combining this, we obtain compound data structures
- this gives rise to new errors, so called shape errors
- a shape error is when the structure of the compound data structure does not fit its use
- Example:
```
d = {("Schneider-Kamp", "Peter") : 65502327}
for last, first, number in d:      print number
```
- use structshape module for debugging
- available from http://thinkpython.com/code/structshape.py
- Example:
```
from structshape import structshape
structshape(d) == "dict of 1 tuple of 2 str->int"
```

SELECTING DATA STRUCTURES

Reading and Cleaning Words

1. read file given as argument
2. break lines into words
3. strip whitespace & punctuation
4. convert to lower-case letters

- import module sys for command line arguments sys.argv
- Example: `import sys; print sys.argv`

- import module string for punctuation
- Example: `import string; print string.punctuation`

- use `translate(None, deletechars)` to remove punctuation
- Example: `"Hello World!".translate(None, "ol")`

Word Frequency in E-Books

1. use program on Project Gutenberg e-book
2. skip over beginning & end of ebook (marked "***")
3. count total number of words
4. count number of times each word is used
5. print 20 most frequently used words

- use Boolean flag to indicate when to start
- use list to gather all words (and count total number)
- use dictionary to count number of times each word is used
- use tuple comparison to sort words

Optional Parameters

- have seen functions that take variable length argument list

- also possible to make some parameters optional
- in this case, default value has to be supplied by programmer
- Example:

```
def print_most_common(hist, num = 10):
    t = most_common(hist)
    print "The most common", num, "words are:"
    for n, word in t[:num]:
        print word, "\t", n
print_most_common(freq, 20)
```

Dictionary Subtraction

1. find all words that do NOT occur in other word list

- to this end, subtract dictionaries from each other
- **Idea:** new dictionary containing with keys only in first dict
- Implementation:

```
def subtract(d1, d2):
    d = {}
    for key in d1:
        if key not in d2:
            d[key] = None
    return d
```

Random Number Generation

- to work with random numbers, import module random
- Example: import random

- function random() returns random float from 0.0 to < 1.0
- Example: for i in range(10): print random.random()

- function randint(a, b) returns random integer in range(a,b+1)
- Example: for i in range(10): print random.randint(1,10)

- function choice(seq) returns random element of a sequence
- Example: random.choice("Slartibartfast")
  random.choice([23, 42, -3.0])

Random Words

1. choose random word from histogram according to frequency

- how to ensure random choice w.r.t. frequency?
- **Idea 1:** create list with n copies of word with frequency n
- Implementation:

```
def random_word(h):
    t = []
    for word, n in h.items():
        t.extend([word] * n)
    return random.choice(t)
```

- works, but very inefficient!

Random Words

- **Idea 2:** use list with cumulative sum of frequencies
- Implementation:

```
def random_word(h):
    words = h.keys();  sum = 0;  cum = []
    for word in words:  sum += h[word];  cum.append(sum)
    num = random.randint(1, cum[-1]); low = 0; high = len(cum)-1
    while low < high:
        mid = (low+high) / 2
        if num <= cum[mid]:  high = mid
        elif num > cum[mid]:  low = mid+1
    return words[low]
```

Markov Analysis

1. generate more meaningful random texts

- word order in texts is not random
- markov analysis maps a finite number of words (prefix) to all possible following words (suffix)

- how to represent the prefixes?

- how to represent the collection of possible suffixes?

- how to represent the mapping from prefixes to suffixes?

Data Structures

- for mapping, we clearly use a dictionary
- for prefixes, we need to be able to "shift" them (list?)
- we also need to use them as dictionary keys
- thus, we use tuples to present prefixes (+ slicing and "*")
- for suffixes, we need to add elements (list? dictionary?)
- we also need to efficiently generate random word (list?)
- tradeoff space vs time
  - dictionary uses less space and easy to add
  - list uses less time for generating a word
  - can change representation before generation

Debugging Hard Bugs

- bugs can be hard to find

- four popular strategies
  1. reading: re-read your code, check that it is right!
  2. running: make changes, experiment with outcome
  3. ruminating: take time to think it over (and over)
  4. retreating: revert to a known-to-be-good version

- often combination of these strategies needed
- always good to view debugging as scientific experiment

FILE HANDLING

Persistence

- persistent = keeping (some) data stored during runs
- transient = beginning from input data each time over

- most programs so far have been transient

- examples of persistent programs:
  - operating systems
  - web servers
  - most app(lication)s on recent iOS and Mac OS X

- text files are easiest way to save some program state
- alternatively, program states can be saved in databases

Writing to a File

- we know how to read a file using open(name)
- we can specify read/write mode using open(name, mode)
- Example:
```
f1 = open("anna_karenina.txt", "r")
f2 = open("myfile.txt", "w")
```

- use method write(str) of file object to append string to file
- Example:
```
f2.write("This is my first line!\n")
f2.write("This is my second line!\n")
```
- each invocation of write(str) will append, not overwrite!

- when you are finished with a file, please close() it
- Example:
```
f1.close()
f2.close()
```

Format Operator

- values need to be converted to a string for use in write(str)
- for single value, the str(object) function can be used
- Example: f.write(str(42))

- alternatively, use *format operator* "%"
- Example: f.write("%d" % 42)
  f.write("The answer is %d, my friend!" % 42)
- first argument *format string*, second argument value
- format sequence %d for integer, %g for float, %s for string

- for multiple values, use tuple as value
- Example: f.write("The %s is %g!" % ("answer", 42.0))

Directories

- file are organized in *directories*
- every program has a *current directory*
- the current directory is used by default, e.g. for open(name)
- get current directory by importing getcwd() from os module
- Example:
```
import os
print os.getcwd()
```
- change current working directory by using chdir(path)
- Example:
```
os.chdir("..")
print os.getcwd()
```
- list contents of a given directory by using os.listdir(path)
- Example:
```
print os.listdir("dm502")
```

Filenames and Paths

- path = directory & file name
- *relative paths* start from current directory
- Example:

```
path1 = "dm502/tools/anna_karenina.txt"
```

- *absolute paths* are independent from current directory
- Example:

```
path2 = "/Users/petersk/sdu/dm502/tools/anna_karenina.py"
```

- can be obtained from relative path using os.path.abspath(path)
- Example:

```
path3 = os.path.abspath(path1)
```

Operations on Paths

- check whether a directory or file exists using os.path.exists
- Example: os.path.exists(path1) == True
  os.path.exists("no_name") == False

- check whether a path is a directory using os.path.isdir
- Example: os.path.isdir(path1) == False
  os.path.isdir("..") == True

- check whether a path is a file using os.path.isfile
- Example: os.path.isfile(path1) == True
  os.path.isfile("..") == False

Traversing Directories

- build a path from directory and realtive path using os.path.join
- Example: path4 = os.path.join("..", "dm502")

- Case: recursively find all files in a directory

```
def find_files(dir):
    for name in os.listdir(dir):
        path = os.path.join(dir, name)
        if os.path.isfile(path):      # print file name
            print path
        else:                         # recursively search subdirectory
            find_files(path)
```

Catching Exceptions

- file operations are error-prone
- Example: `open("no_name")  # raises IOError`

- good idea to avoid errors using `os.path.exists` etc.
- not possible to check all possible situations

- use try-except statement to handle error situations
- Example:

```
try:
    f = open(name)
    lines = f.readlines()
except:
    lines = ["ERROR"]
```