DM550/DM857
Introduction to Programming

Peter Schneider-Kamp

petersk@imada.sdu.dk

http://imada.sdu.dk/~petersk/DM550/

http://imada.sdu.dk/~petersk/DM857/

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,end="")          # 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 sum(map(lambda x:x[0]*x[1], zip(x, y)))
```

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)]

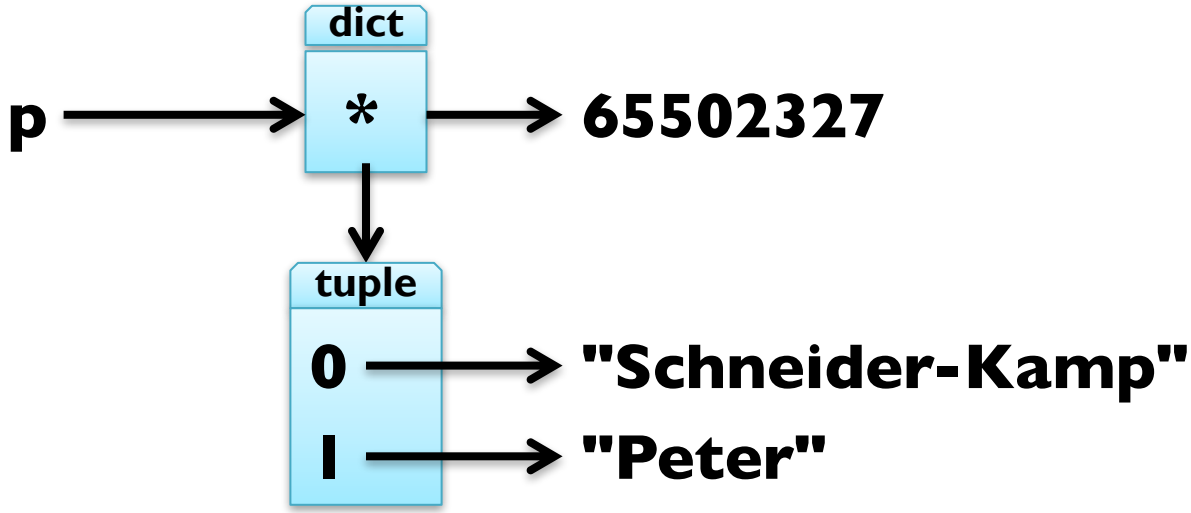
- tuples can also be used to create new dictionary using dict()
- Example: t = [("a", 3), ("c", 1), ("b", 2)]
  dict(t) == {"a" : 3, "b" : 2, "c" : 1}

- 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.items():  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(dict)` to remove punctuation
- Example: `"Hello World!".translate({ord("o"):"",ord("l"):""})`

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