CSc 120

Introduction to Computer Programing II

Adapted from slides by
Dr. Saumya Debray

01-b: Python review

Lists of Lists

a list can consist of elements of many types, including lists

```
>>> x = [ [1,2,3], [4], [5, 6]]
>>> x
[[1, 2, 3], [4], [5, 6]]
>>>
>>>
>>> >>> y = [ ['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'], ['hh', 'ii', 'jj']]
>>> >>> y
[['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'], ['hh', 'ii', 'jj']]
>>>
```

a list of lists is called a 2-d list

Lists of Lists

```
>>> x = [ [1,2,3], [4], [5, 6]]
>>> x
[[1, 2, 3], [4], [5, 6]]
>>>
>>>
>>> >>> y = [ ['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'], ['hh', 'ii', 'jj']]
>>> >>> y
[['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'], ['hh', 'ii', 'jj']]
>>>
```

a list can consist of elements of many types, including lists

a list of lists is called a 2-d list

if the number of rows and columns are equal, it is a grid

Lists of Lists

```
>>> y
[['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'], ['hh', 'ii', 'jj']]
>>>
>>> y[0]
['aa', 'bb', 'cc']
>>> y[1]
['dd', 'ee', 'ff']
>>> y[2]
['hh', 'ii', 'jj']
>>>
>>> len(y)
3
>>> len(y[0])
3
>>>
```

a list can consist of elements of many types, including lists

a list of lists is called a 2-d list

if the number of rows and columns are equal, it is a grid

*must check the length of each row

Lists of Lists

a list can consist of elements of many types, including lists

```
>>> x = [ [1,2,3], [4], [5, 6]]
>>> x
[[1, 2, 3], [4], [5, 6]]
>>>
>>>
>>> >>> y = [ ['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'], ['hh', 'ii', 'jj']]
>>> >>> y
[['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'], ['hh', 'ii', 'jj']]
>>>
```

this is not a grid

EXERCISE

```
>>> y
[['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'], ['hh', 'ii', 'jj']]
>>>
>>> y[0]
['aa', 'bb', 'cc']
>>> y[1]
['dd', 'ee', 'ff']
>>> y[2]
['hh', 'ii', 'jj']
>>> y[0][1]
'bb'
>>>
```

how do we access 'bb'?

EXERCISE

write the code to sum the first column of x

```
 >>> x = [ [18, 25, 36], [23, 25, 18], [20, 54, 7] ]
>>> x
[ [18, 25, 36], [23, 25, 18], [20, 54, 7] ]
>>>
>> r, total = 0, 0
>>> while r < len(x):
            total += x[r][0]
            r += 1

>>> total
61
>>>
```

Lists

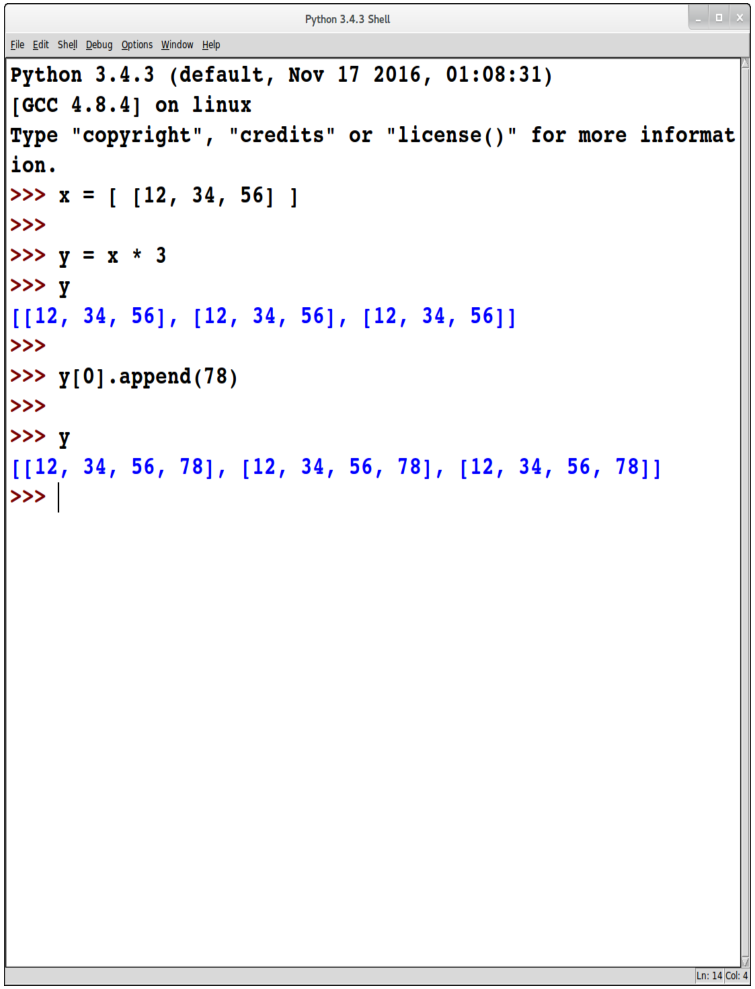
concatenation (+ and *) : similar to strings

these operators create “shallow” copies

- due to list mutability, this can cause unexpected behavior

```
>>> x = [ [12, 34, 56] ]
>>> y = x * 3
>>> y
[[12, 34, 56], [12, 34, 56], [12, 34, 56]]
>>>
>>> y[0].append(78)
>>>
>>> y
[[12, 34, 56, 78], [12, 34, 56, 78], [12, 34, 56, 78]]
>>>
```

Lists



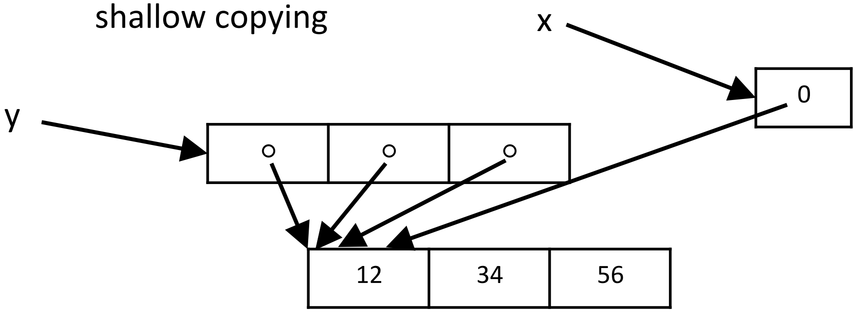
```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = [ [12, 34, 56] ]
>>>
>>> y = x * 3
>>> y
[[12, 34, 56], [12, 34, 56], [12, 34, 56]]
>>>
>>> y[0].append(78)
>>>
>>> y
[[12, 34, 56, 78], [12, 34, 56, 78], [12, 34, 56, 78]]
>>> 
```

concatenation (+ and *) : similar to strings

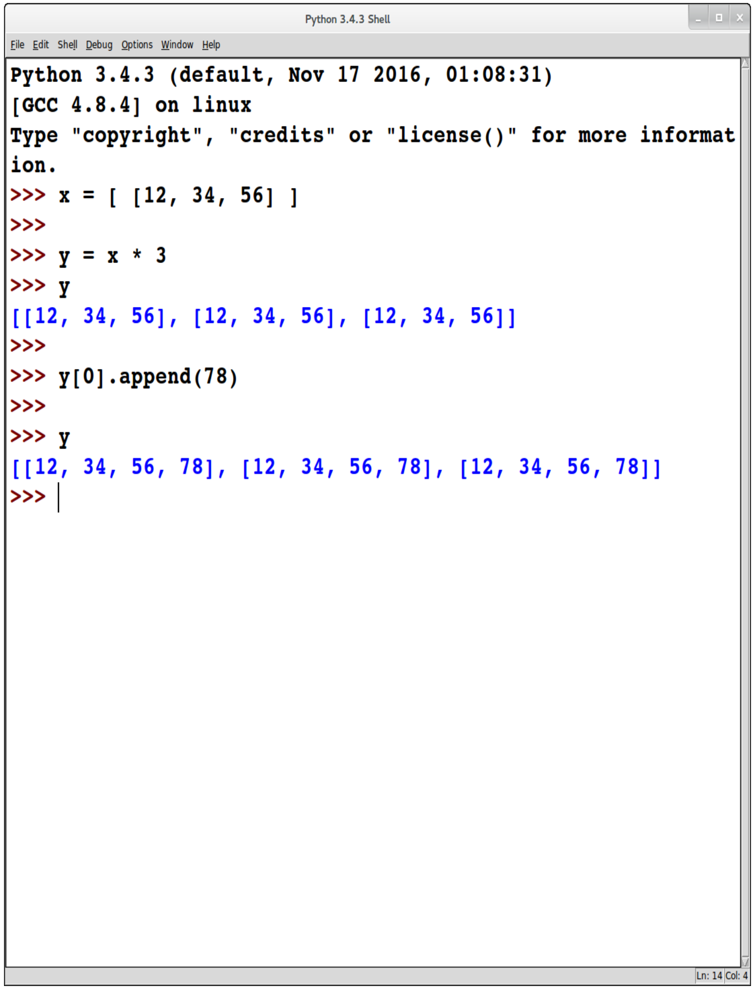
these operators create “shallow” copies

- due to list mutability, this can cause unexpected behavior

shallow copying



Lists

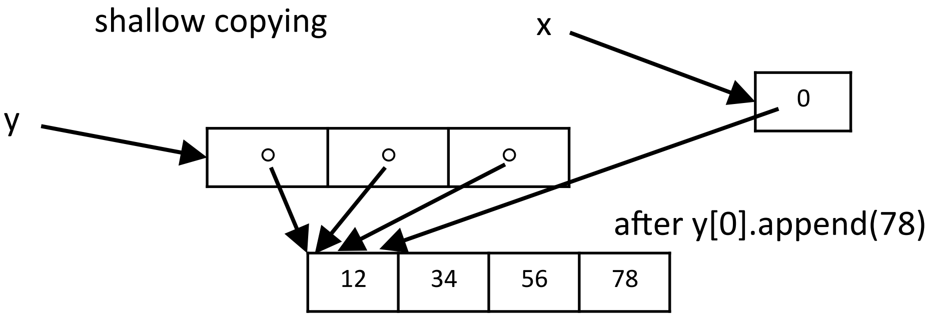


```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = [ [12, 34, 56] ]
>>>
>>> y = x * 3
>>> y
[[12, 34, 56], [12, 34, 56], [12, 34, 56]]
>>>
>>> y[0].append(78)
>>>
>>> y
[[12, 34, 56, 78], [12, 34, 56, 78], [12, 34, 56, 78]]
>>> 
```

concatenation (+ and *) : similar to strings

these operators create “shallow” copies

- due to list mutability, this can cause unexpected behavior



Lists: sorting

```
>>> x = [1, 4, 3, 2, 5]
>>> x
[1, 4, 3, 2, 5]
>>> x.sort()
>>> x
[1, 2, 3, 4, 5]
>>>
>>> y = [1, 4, 3, 2, 5]
>>> y
[1, 4, 3, 2, 5]
>>> sorted(y)
[1, 2, 3, 4, 5]
>>> y
[1, 4, 3, 2, 5]
>>>
```

sort() : sorts a list

Lists: sorting

```
>>> x = [1, 4, 3, 2, 5]
>>> x
[1, 4, 3, 2, 5]
>>> x.sort()
>>> x
[1, 2, 3, 4, 5]
>>>
>>> y = [1, 4, 3, 2, 5]
>>> y
[1, 4, 3, 2, 5]
>>> sorted(y)
[1, 2, 3, 4, 5]
>>> y
[1, 4, 3, 2, 5]
>>>
```

sort() : sorts a list

sorted() : creates a sorted copy of a list;
the original list is not changed

python review:
for loops

Loops II: **for**

- The for loop iterates over the items of any sequence in order
- **for**-statement syntax:

```
for  Var in Expr :
        $stmt_1$
        …
        $stmt_n$
```

- *Expr* is evaluated. $stmt_1$ … $stmt_n$ are executed for each element of the sequence that *Expr* produces; *Var* is assigned to each successive element.

Loops II: **for**

```
>>> nums = [18, 3, 24, 63, 18, 4, 7]
>>>
>>> evens = []
>>> for n in nums:
        if n % 2 == 0:
           evens.append(n)

>>> evens
[18, 24, 18, 4]
>>>
```

- sequence: a list or string
  (there are more, as you will see)

range

- **range** generates generates a sequence of numbers

- **range** syntax:

range(start, stop, step)

range(start, stop)

Produces the sequence of integers from ***start*** to ***stop*** (exclusive). If ***step*** is omitted, it defaults to 1.

for with range

```
>>> nums = [18, 3, 24, 63, 18, 4, 7]
>>>
>>> evens = []
>>> for i in range(0,len(nums)):
        if nums[i] % 2 == 0:
           evens.append(nums[i])

>>> evens
[18, 24, 18, 4]
>>>
```

- generates the numbers 0,1,2,3,4,5,6

EXERCISE

```
 >>> x = [ [18, 25, 36], [23, 25, 18], [20, 54, 7] ]
>>> x
[ [18, 25, 36], [23, 25, 18], [20, 54, 7] ]
>>>
>>> total = 0
>>> for i in range(0, len(x)):
        total += x[i][0]

>>> total
61
>>>
```

write the code to sum the first column of x *using for and range*

EXERCISE

```
>>> x = [ [18, 25, 36], [23, 25, 18], [20, 54, 7] ]
>>> x
[ [18, 25, 36], [23, 25, 18], [20, 54, 7] ]
>>>
>>> total = 0
>>> for row in x:
        total += row[0]

>>> total
61
>>>
```

write the code to sum the first column of x *using for (no range)*

python review:
lists ↔ strings

Strings → lists

```
>>> names = "John, Paul, Megan, Bill, Mary"
>>> names
'John, Paul, Megan, Bill, Mary'
>>>
>>> names.split()
['John,', 'Paul,', 'Megan,', 'Bill,', 'Mary']
>>>
>>> names.split('n')
['Joh', ', Paul, Mega', ', Bill, Mary']
>>>
>>> names.split(',')
['John', ' Paul', ' Megan', ' Bill', ' Mary']
>>>
```

split() : splits a string on whitespace
returns a list of strings

Strings → lists

```
>>> names = "John, Paul, Megan, Bill, Mary"
>>> names
'John, Paul, Megan, Bill, Mary'
>>>
>>> names.split()
['John,', 'Paul,', 'Megan,', 'Bill,', 'Mary']
>>>
>>> names.split('n')
['Joh', ', Paul, Mega', ', Bill, Mary']
>>>
>>> names.split(',')
['John', ' Paul', ' Megan', ' Bill', ' Mary']
>>>
```

split() : splits a string on whitespace
returns a list of strings

split(*delim*) :
delim, splits the string
on *delim*

Lists → strings

```
>>> x = ['one', 'two', 'three', 'four']
>>>
>>> "-".join(x)
'one-two-three-four'
>>>
>>> "!.!".join(x)
'one!.!two!.!three!.!four'
>>>
```

delim.join(*list*) : joins the strings in *list* using the string *delim* as the delimiter

returns a string

String trimming

```
>>> x = '    abcd      '
>>>
>>> x.strip()
'abcd'
>>>
>>> y = "Hey!!!"
>>>
>>> y.strip("!")
'Hey'
>>> >>> z = "*%^^stuff stuff stuff^%%%**"
>>>
>>> z.strip("*^%")
'stuff stuff stuff'
```

x.strip() : removes whitespace from either end of the string *x*

returns a string

String trimming

```
>>> x = '     abcd      '
>>>
>>> x.strip()
'abcd'
>>>
>>> y = "Hey!!!"
>>>
>>> y.strip("!")
'Hey'
>>> >>> z = "*%^^stuff stuff stuff^%%%**"
>>>
>>> z.strip("*^%")
'stuff stuff stuff'
```

x.strip() : removes whitespace from either end of the string *x*

returns a string

x.strip(*string*) : given an optional argument *string*, removes any character in *string* from either end of *x*

String trimming

x.strip() : removes whitespace from
either end of the string *x*

x.strip(*string*) : given an optional
argument *string*, removes
any character in *string* from
either end of *x*

rstrip(), lstrip() : similar to strip() but
trims from one end of
the string

EXERCISE

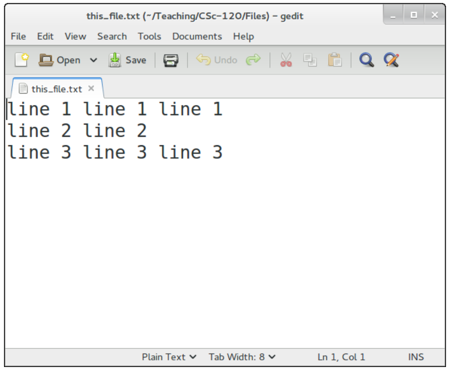
create a list of words with no punctuation

```
>>> text = "Bear Down, Arizona. Bear Down, Red and Blue."
>>> text_lst = text.split()
>>> text_lst
['Bear', 'Down,', 'Arizona.', 'Bear', 'Down,', 'Red', 'and', 'Blue.']
>>> words_lst = []
>>> for w in words:
        words_lst.append(w.strip(".,"))

>>> words_lst
['Bear', 'Down', 'Arizona', 'Bear', 'Down', 'Red', 'and', 'Blue']
>>>
```

python review: reading user input II: file I/O

Reading user input II: file I/O

suppose we want to read (and process) a file "this_file.txt"

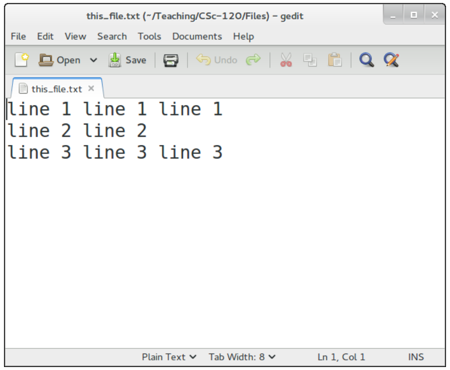


this_file.txt (~/Teaching/CSc-120/Files) - gedit
File Edit View Search Tools Documents Help
Open
Save
Undo
this_file.txt
Plain Text
Tab Width: 8
Ln 1, Col 1
INS

```
line 1 line 1 line 1
line 2 line 2
line 3 line 3 line 3
```

Reading user input II: file I/O

```
>>> infile = open("this_file.txt")
>>>
>>> for line in infile:
            print(line)



line 1 line 1 line 1


line 2 line 2


line 3 line 3


>>>
```



```
line 1 line 1 line 1
line 2 line 2
line 3 line 3
```

- open() the file
- read and process the file

Reading user input II: file I/O

```
>>> infile = open("this_file.txt")
>>>
>>> for line in infile:
        print(line)


line 1 line 1 line 1

line 2 line 2

line 3 line 3

>>>
```

- *fileobj* = **`open`**(*filename*)
  - *filename*: a string
  - *fileobj*: a file object

Reading user input II: file I/O

```
>>> infile = open("this_file.txt")
>>>
>>> for line in infile:
            print(line)


line 1 line 1 line 1


line 2 line 2


line 3 line 3


>>>
```

- *fileobj* = **`open`**(*filename*)
  - *filename*: a string
  - *fileobj*: a file object
- **`for`** *var* **`in`** *fileobj*:
  - reads the file a line at a time
  - assigns the line (a string) to *var*

Reading user input II: file I/O

```
>>> infile = open("this_file.txt")
>>>
>>> for line in infile:
        print(line)



line 1 line 1 line 1

line 2 line 2

line 3 line 3

>>>
```

- *fileobj* = **`open`**(*filename*)
  - *filename*: a string
  - *fileobj*: a file object
- **`for`** *var* **`in`** *fileobj*:
  - reads the file a line at a time
  - assigns the line (a string) to *var*

Note that each line read ends in a newline ('\n') character

Reading user input II: file I/O

```
>>> infile = open("this_file.txt")
>>>
>>> for line in infile:
        print(line)



line 1 line 1 line 1

line 2 line 2

line 3 line 3

>>>
```

At this point we've reached the end of the file and there is nothing left to read

Reading user input II: file I/O

```
>>> infile = open("this_file.txt")
>>>
>>> for line in infile:
            print(line)


line 1 line 1 line 1

line 2 line 2

line 3 line 3

>>>
>>> infile.close()
>>>infile = open("this_file.txt")
```

at this point we've reached the end of the file so there's nothing left to read

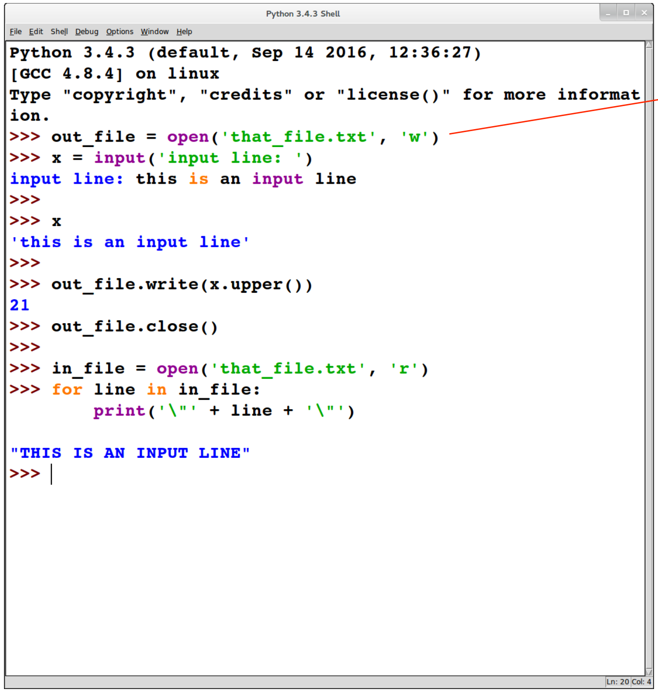
to re-read the file, we have to close it and then re-open it

Reading user input II: file I/O

```
>>> infile = open("this_file.txt")
>>>
>>> for line in infile:
        print(line.strip())


line 1 line 1 line 1
line 2 line 2
line 3 line 3


>>>
```

NOTE: we can use strip() to get rid of the newline character at the end of each line

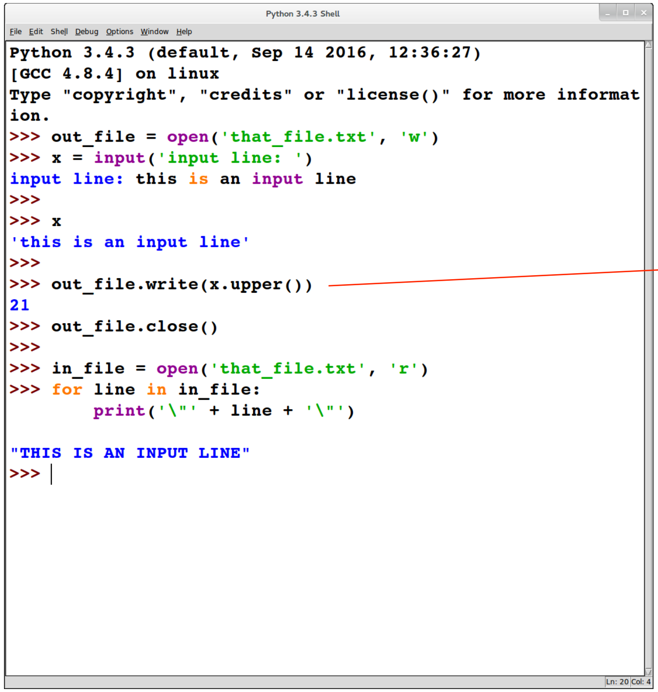
Writing output to a file



```
Python 3.4.3 (default, Sep 14 2016, 12:36:27)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> out_file = open('that_file.txt', 'w')
>>> x = input('input line: ')
input line: this is an input line
>>> 
>>> x
'this is an input line'
>>> 
>>> out_file.write(x.upper())
21
>>> out_file.close()
>>> 
>>> in_file = open('that_file.txt', 'r')
>>> for line in in_file:
        print('\"' + line + '\"')

"THIS IS AN INPUT LINE"
>>> 
```

`open`(*filename*, **`"w"`**) : opens *filename* in write mode, i.e., for output

Writing output to a file

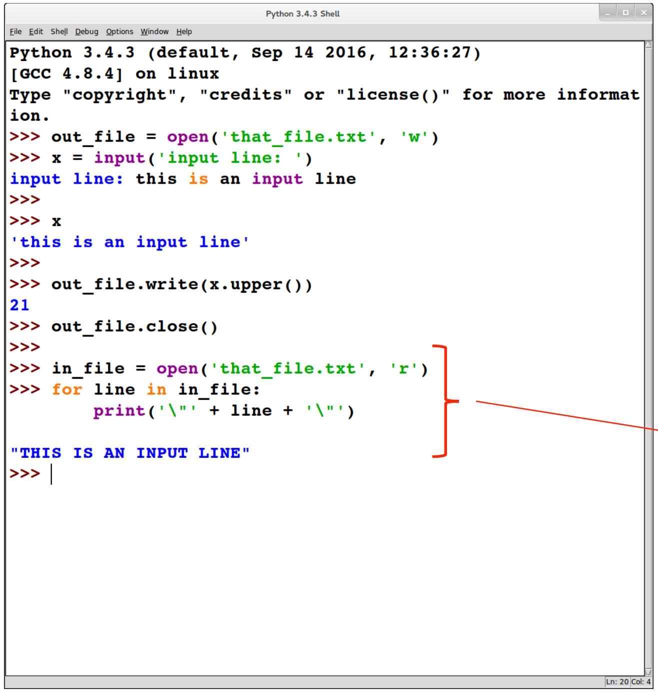


```
Python 3.4.3 (default, Sep 14 2016, 12:36:27)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> out_file = open('that_file.txt', 'w')
>>> x = input('input line: ')
input line: this is an input line
>>>
>>> x
'this is an input line'
>>>
>>> out_file.write(x.upper())
21
>>> out_file.close()
>>>
>>> in_file = open('that_file.txt', 'r')
>>> for line in in_file:
        print('\"' + line + '\"')

"THIS IS AN INPUT LINE"
>>> 
```

`open`(*filename*, **`"w"`**) : opens *filename* in write mode, i.e., for output

fileobj **`.write`**(*string*) : writes *string* to *fileobj*

Writing output to a file



```
Python 3.4.3 (default, Sep 14 2016, 12:36:27)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> out_file = open('that_file.txt', 'w')
>>> x = input('input line: ')
input line: this is an input line
>>> 
>>> x
'this is an input line'
>>> 
>>> out_file.write(x.upper())
21
>>> out_file.close()
>>> 
>>> in_file = open('that_file.txt', 'r')
>>> for line in in_file:
        print('\"' + line + '\"')

"THIS IS AN INPUT LINE"
>>> 
```

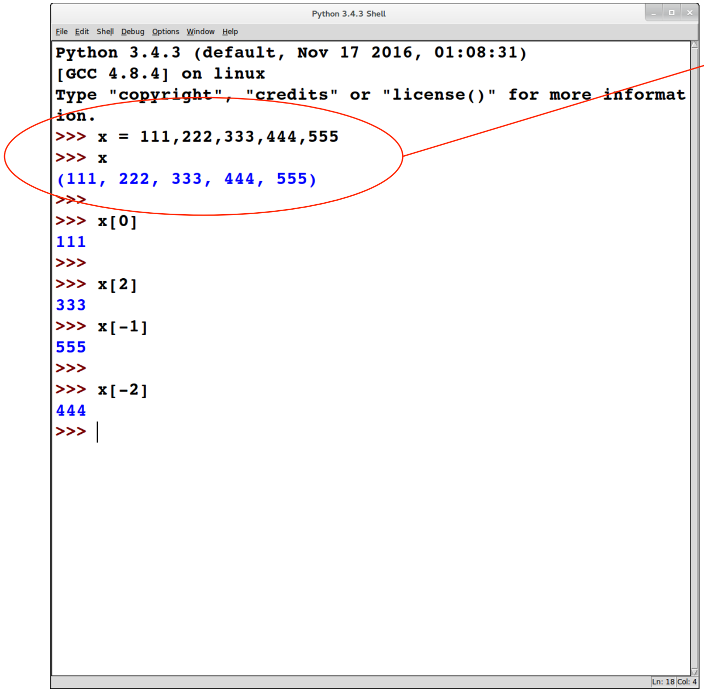
open(*filename*, **"w"**) : opens *filename* in write mode, i.e., for output

fileobj **.write**(*string*) : writes *string* to *fileobj*

open the file in read mode ("r") to see what was written

python review: tuples

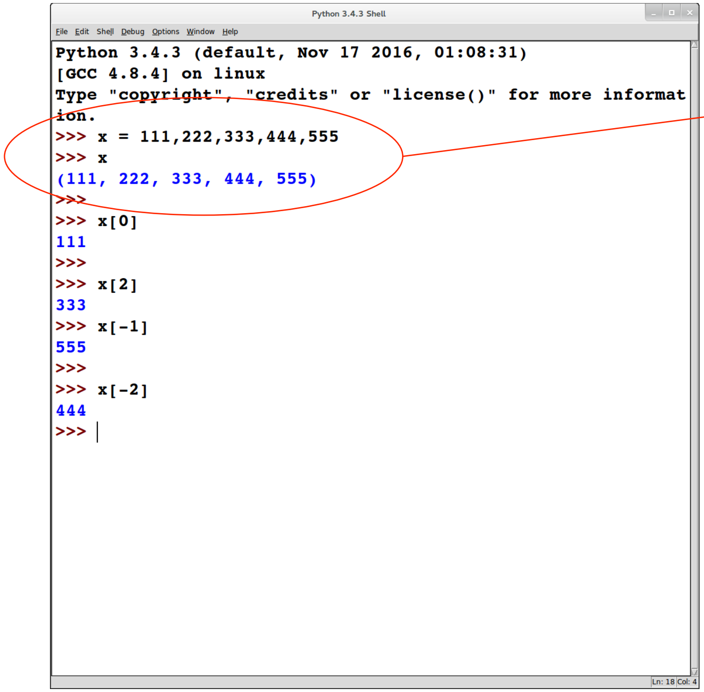
Tuples



```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = 111,222,333,444,555
>>> x
(111, 222, 333, 444, 555)
>>>
>>> x[0]
111
>>>
>>> x[2]
333
>>> x[-1]
555
>>>
>>> x[-2]
444
>>> 
```

a tuple is a sequence of values (like lists)

Tuples



```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = 111,222,333,444,555
>>> x
(111, 222, 333, 444, 555)
>>>
>>> x[0]
111
>>>
>>> x[2]
333
>>> x[-1]
555
>>>
>>> x[-2]
444
>>> 
```

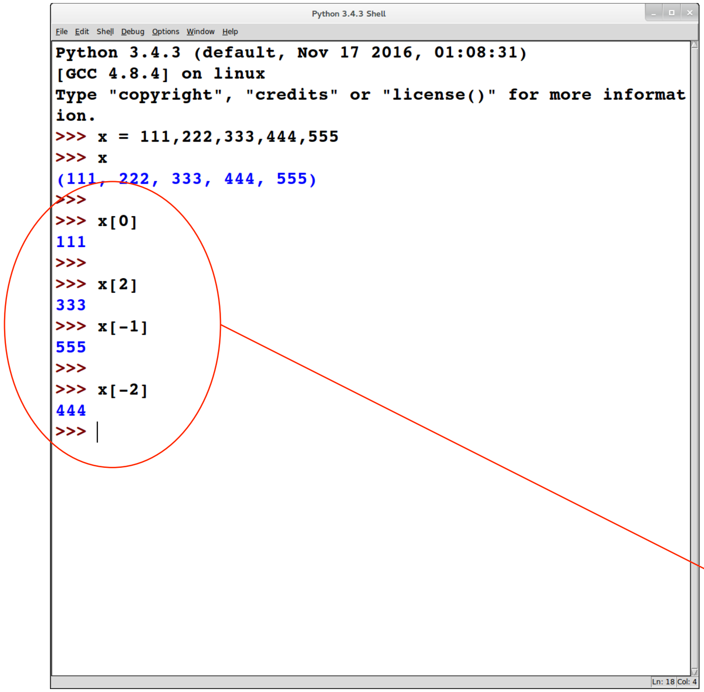
a tuple is a sequence of values (like lists)

tuples use parens ()

- by contrast, lists use square brackets [ ]
  - parens can be omitted if no confusion is possible
- special cases for tuples:
  - empty tuple: ()
  - single-element tuple: must have comma after the element:

    (111,)

Tuples



```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = 111,222,333,444,555
>>> x
(111, 222, 333, 444, 555)
>>>
>>> x[0]
111
>>>
>>> x[2]
333
>>> x[-1]
555
>>>
>>> x[-2]
444
>>> 
```

a tuple is a sequence of values (like lists)

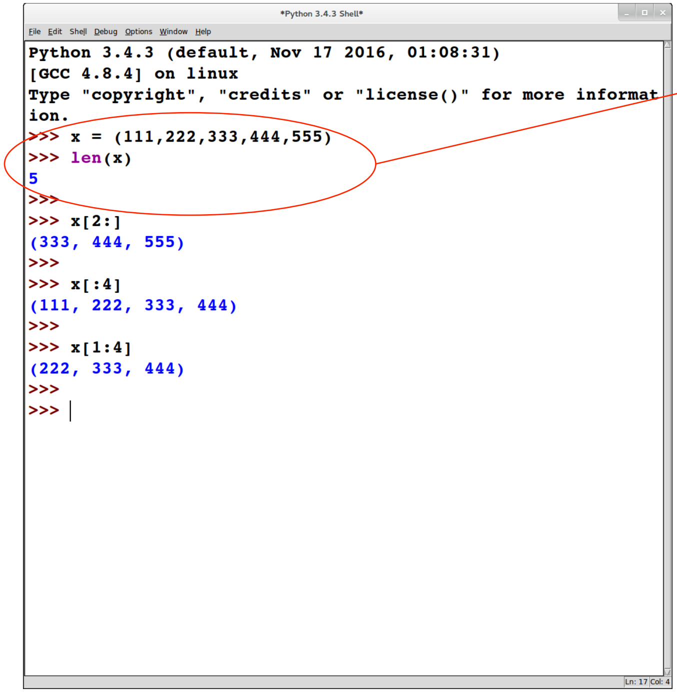
tuples use parens ()

- by contrast, lists use square brackets [ ]
  - parens can be omitted if no confusion is possible
- special cases for tuples:
  - empty tuple: ()
  - single-element tuple: must have comma after the element:

    (111,)

indexing in tuples works similarly to strings and lists

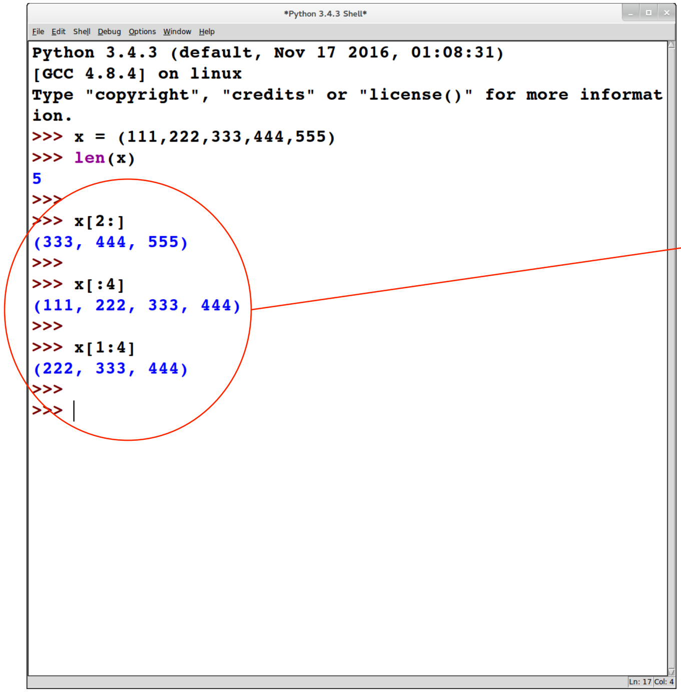
Tuples



```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = (111,222,333,444,555)
>>> len(x)
5
>>>
>>> x[2:]
(333, 444, 555)
>>>
>>> x[:4]
(111, 222, 333, 444)
>>>
>>> x[1:4]
(222, 333, 444)
>>>
>>> 
```

computing a length of a tuple: similar to strings and lists

Tuples

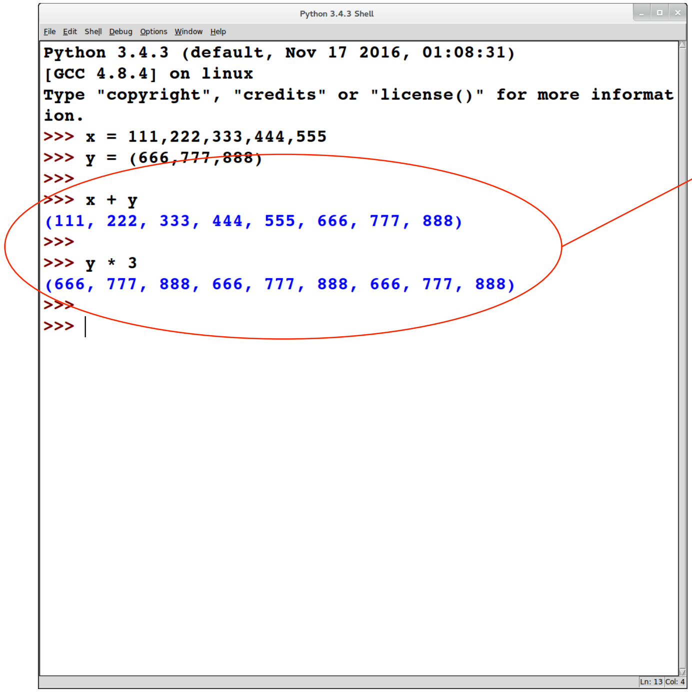


```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = (111,222,333,444,555)
>>> len(x)
5
>>>
>>> x[2:]
(333, 444, 555)
>>>
>>> x[:4]
(111, 222, 333, 444)
>>>
>>> x[1:4]
(222, 333, 444)
>>>
>>> 
```

computing a length of a tuple: similar to strings and lists

computing slices of a tuple: similar to strings and lists

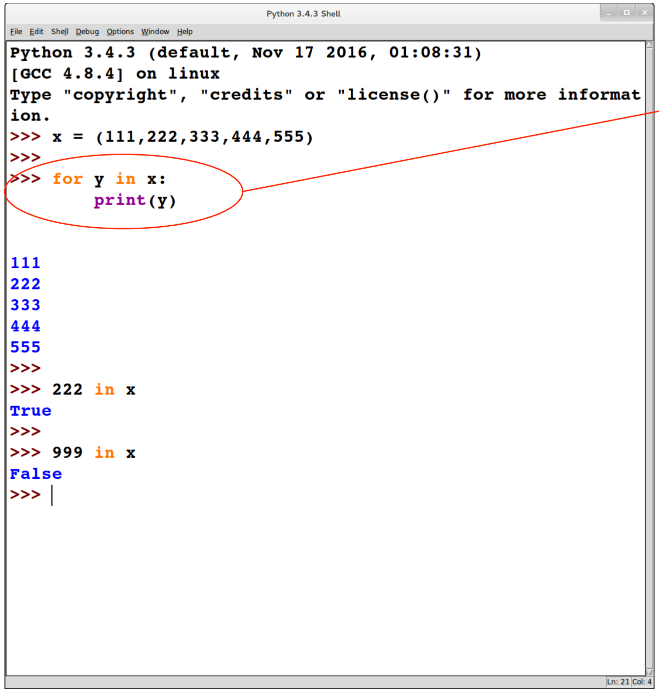
Tuples



```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = 111,222,333,444,555
>>> y = (666,777,888)
>>>
>>> x + y
(111, 222, 333, 444, 555, 666, 777, 888)
>>>
>>> y * 3
(666, 777, 888, 666, 777, 888, 666, 777, 888)
>>>
>>> 
```

+ and * work similarly on tuples as for lists and strings

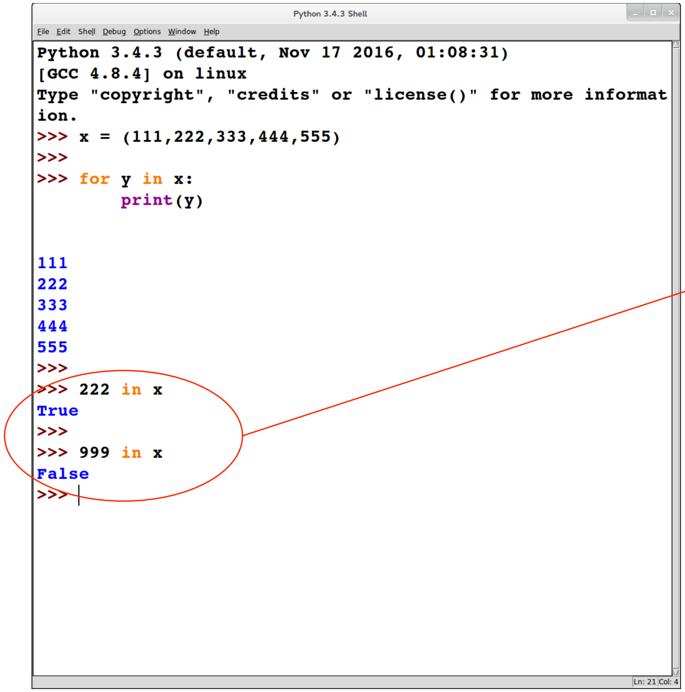
Tuples



```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = (111,222,333,444,555)
>>>
>>> for y in x:
        print(y)


111
222
333
444
555
>>>
>>> 222 in x
True
>>>
>>> 999 in x
False
>>> 
```

iterating through the elements of a tuple: similar to lists and strings

Tuples

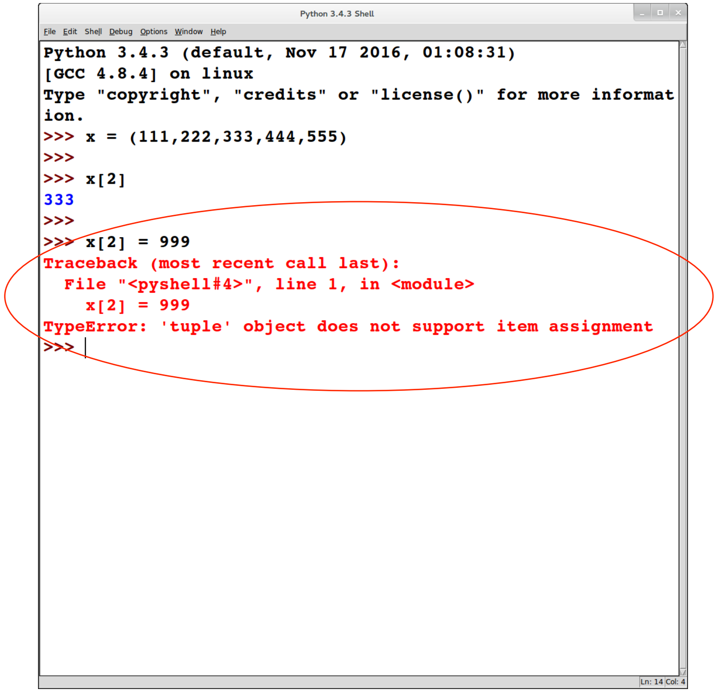


```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = (111,222,333,444,555)
>>>
>>> for y in x:
        print(y)


111
222
333
444
555
>>>
>>> 222 in x
True
>>>
>>> 999 in x
False
>>> 
```

iterating through the elements of a tuple: similar to lists and strings

checking membership in a tuple: similar to lists and strings

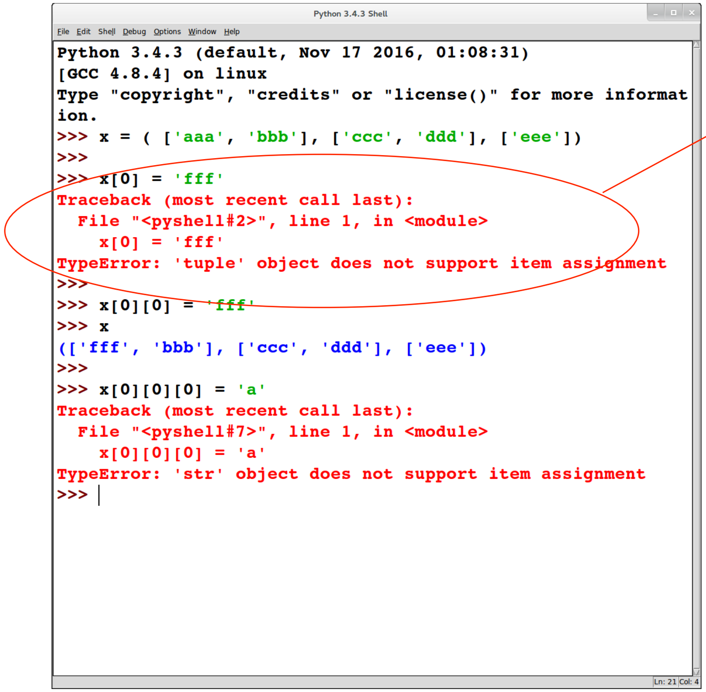
Tuples



```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = (111,222,333,444,555)
>>>
>>> x[2]
333
>>>
>>> x[2] = 999
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    x[2] = 999
TypeError: 'tuple' object does not support item assignment
>>> 
```

tuples are not mutable

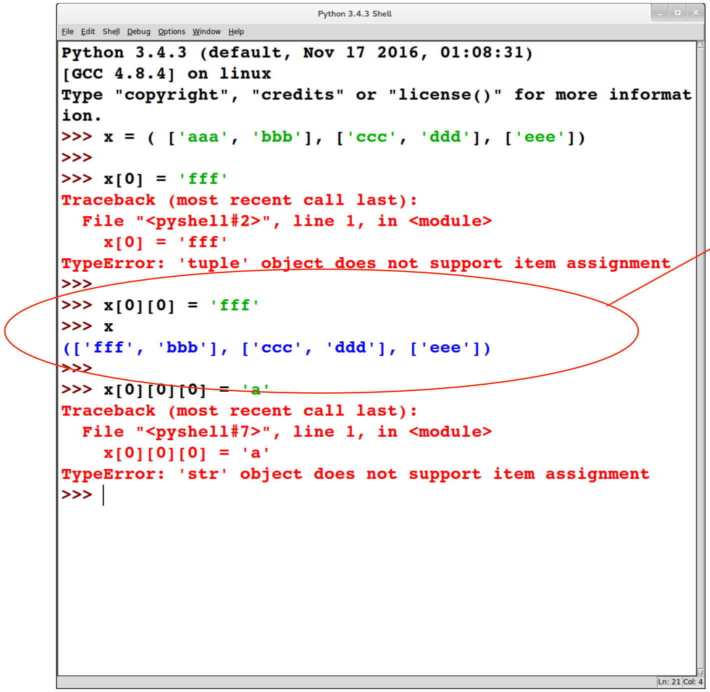
Sequence types: mutability



```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = ( ['aaa', 'bbb'], ['ccc', 'ddd'], ['eee'])
>>>
>>> x[0] = 'fff'
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    x[0] = 'fff'
TypeError: 'tuple' object does not support item assignment
>>>
>>> x[0][0] = 'fff'
>>> x
(['fff', 'bbb'], ['ccc', 'ddd'], ['eee'])
>>>
>>> x[0][0][0] = 'a'
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    x[0][0][0] = 'a'
TypeError: 'str' object does not support item assignment
>>> 
```

tuples are immutable

Sequence types: mutability

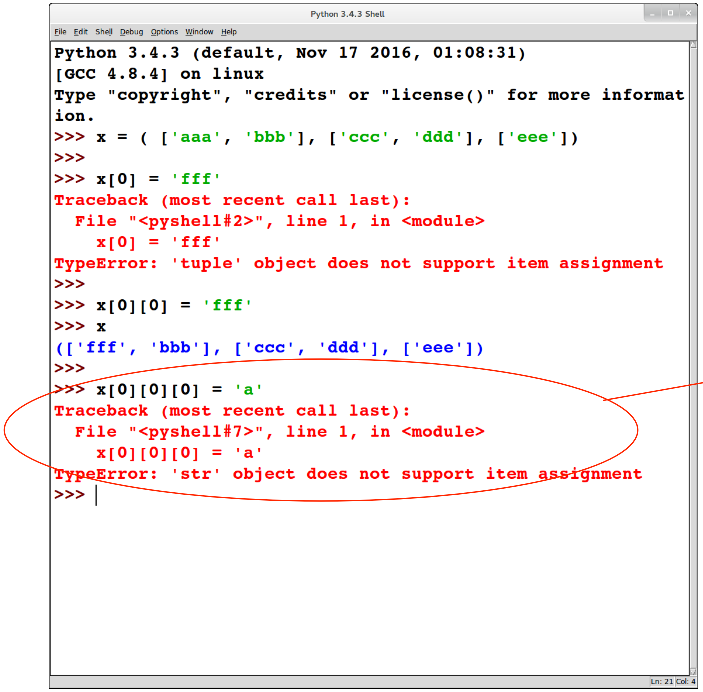


```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = ( ['aaa', 'bbb'], ['ccc', 'ddd'], ['eee'])
>>>
>>> x[0] = 'fff'
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    x[0] = 'fff'
TypeError: 'tuple' object does not support item assignment
>>>
>>> x[0][0] = 'fff'
>>> x
(['fff', 'bbb'], ['ccc', 'ddd'], ['eee'])
>>>
>>> x[0][0][0] = 'a'
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    x[0][0][0] = 'a'
TypeError: 'str' object does not support item assignment
>>> 
```

tuples are immutable

lists are mutable (even if the list is an element of a [immutable] tuple)

Sequence types: mutability



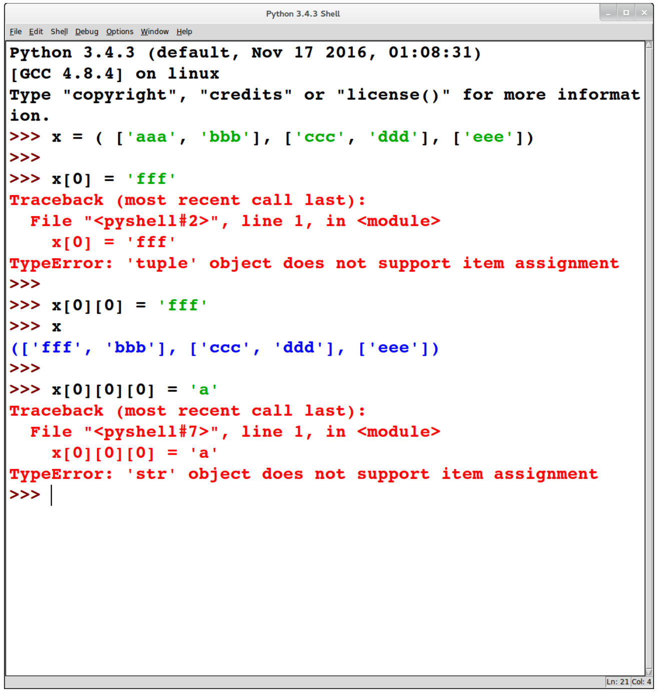
```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = ( ['aaa', 'bbb'], ['ccc', 'ddd'], ['eee'])
>>>
>>> x[0] = 'fff'
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    x[0] = 'fff'
TypeError: 'tuple' object does not support item assignment
>>>
>>> x[0][0] = 'fff'
>>> x
(['fff', 'bbb'], ['ccc', 'ddd'], ['eee'])
>>>
>>> x[0][0][0] = 'a'
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    x[0][0][0] = 'a'
TypeError: 'str' object does not support item assignment
>>> 
```

tuples are immutable

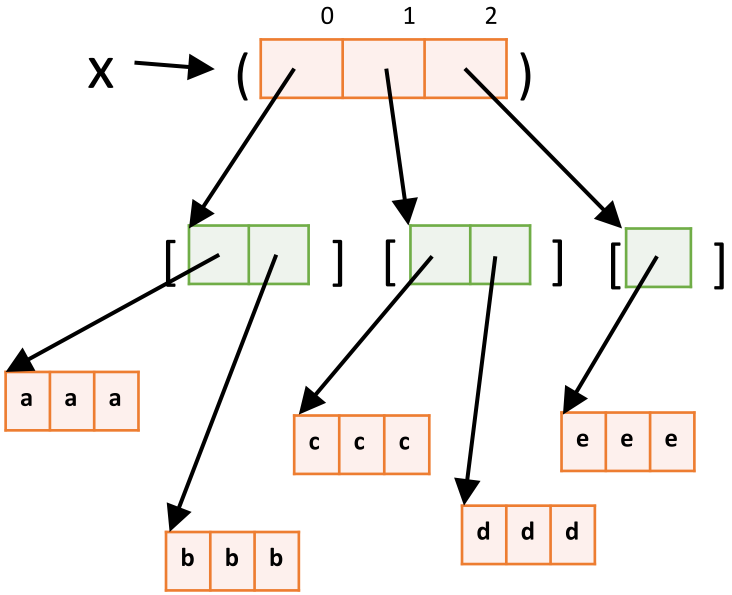
lists are mutable (even if the list is an element of a [immutable] tuple)

strings are immutable (even if the string is an element of a [mutable] list)

Sequence types: mutability



```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = ( ['aaa', 'bbb'], ['ccc', 'ddd'], ['eee'])
>>>
>>> x[0] = 'fff'
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    x[0] = 'fff'
TypeError: 'tuple' object does not support item assignment
>>>
>>> x[0][0] = 'fff'
>>> x
(['fff', 'bbb'], ['ccc', 'ddd'], ['eee'])
>>>
>>> x[0][0][0] = 'a'
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    x[0][0][0] = 'a'
TypeError: 'str' object does not support item assignment
>>> |
```

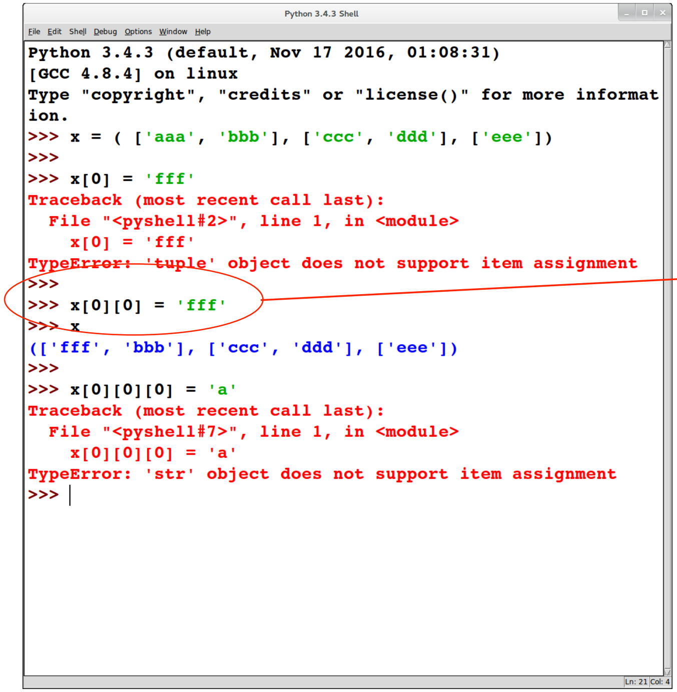


tuple
(immutable)

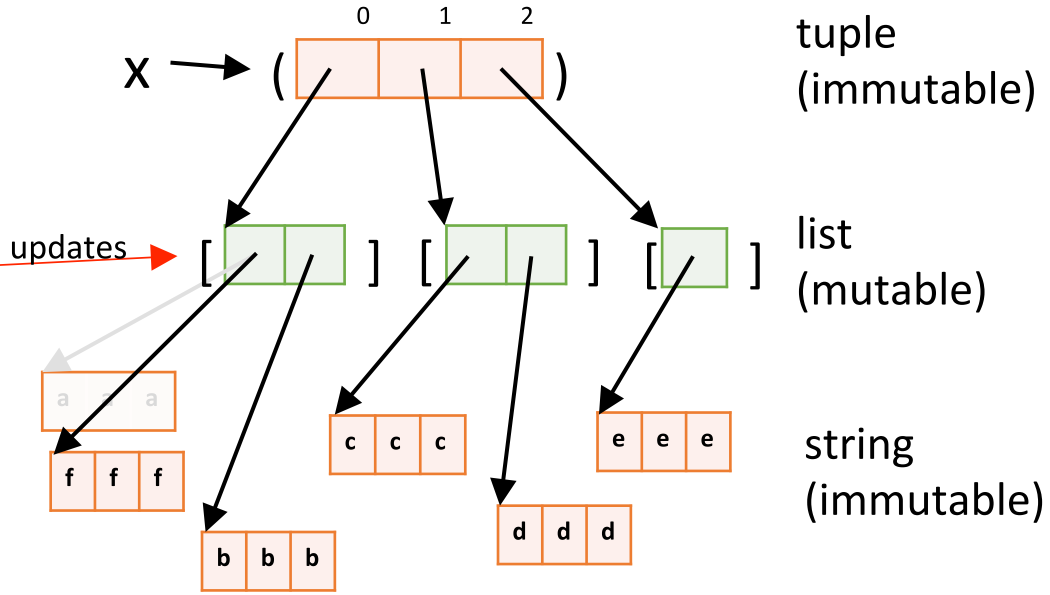
list
(mutable)

string
(immutable)

Sequence types: mutability



```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = ( ['aaa', 'bbb'], ['ccc', 'ddd'], ['eee'])
>>>
>>> x[0] = 'fff'
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    x[0] = 'fff'
TypeError: 'tuple' object does not support item assignment
>>>
>>> x[0][0] = 'fff'
>>> x
(['fff', 'bbb'], ['ccc', 'ddd'], ['eee'])
>>>
>>> x[0][0][0] = 'a'
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    x[0][0][0] = 'a'
TypeError: 'str' object does not support item assignment
>>> 
```



Why use tuples?

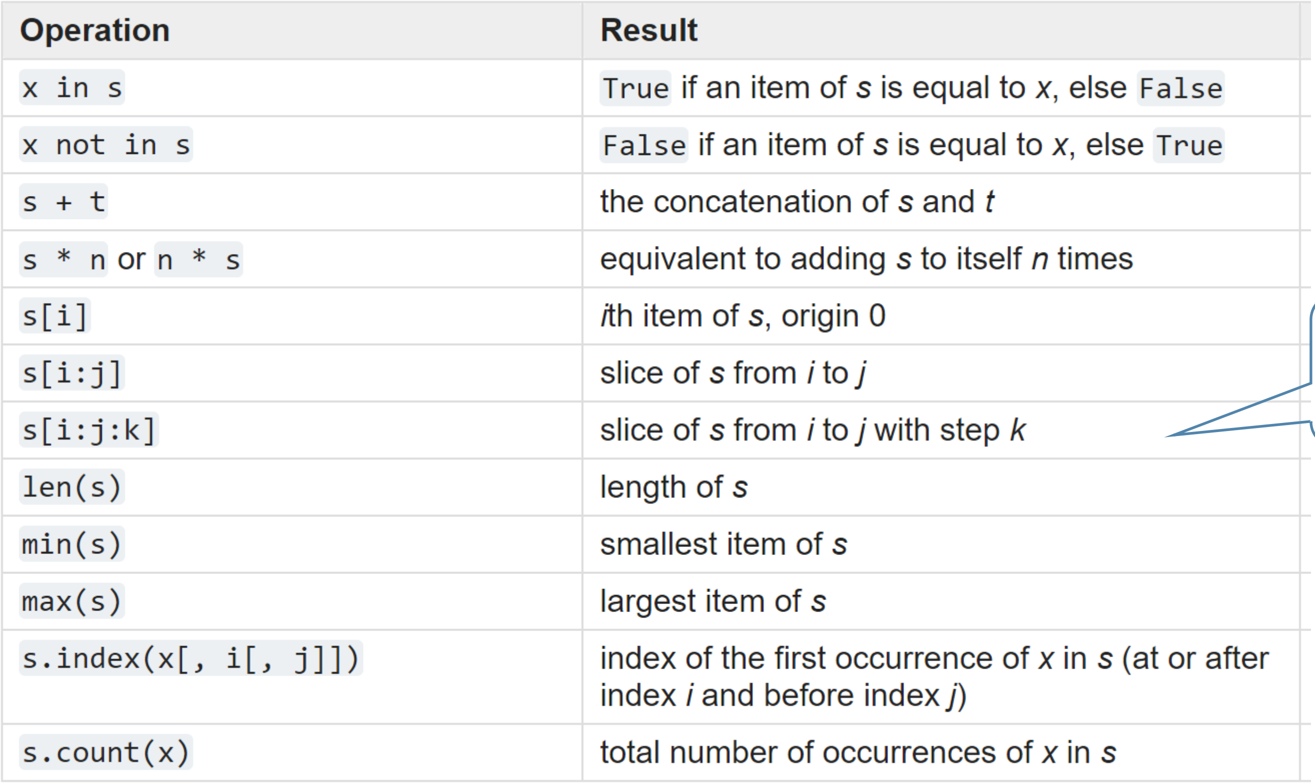
At the implementation level, tuples are much simpler than lists:

- lists are mutable; tuples are immutable
  - this means that the implementation can process tuples without having to worry about the possibility of updates
- lists have methods (e.g., append); tuples do not have methods

⇒ Tuples can be implemented more efficiently than lists

Summary: sequence types

Sequence types include: strings, lists, and tuples



Operation	Result
`x in s`	`True` if an item of *s* is equal to *x*, else `False`
`x not in s`	`False` if an item of *s* is equal to *x*, else `True`
`s + t`	the concatenation of *s* and *t*
`s * n` or `n * s`	equivalent to adding *s* to itself *n* times
`s[i]`	*i*th item of *s*, origin 0
`s[i:j]`	slice of *s* from *i* to *j*
`s[i:j:k]`	slice of *s* from *i* to *j* with step *k*
`len(s)`	length of *s*
`min(s)`	smallest item of *s*
`max(s)`	largest item of *s*
`s.index(x[, i[, j]])`	index of the first occurrence of *x* in *s* (at or after index *i* and before index *j*)
`s.count(x)`	total number of occurrences of *x* in *s*

The elements are: *i, i+k, i +2k, …*

Source: https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range

EXERCISE

```
>>> x = [ (1, 2, 3), (4, 5, 6), (7, 8, 9) ]
>>> x[0][0] = (2, 3, 4)
```

what do you think will be printed out?

```
>>> x[0] = [ 2, 3, 4 ]
```

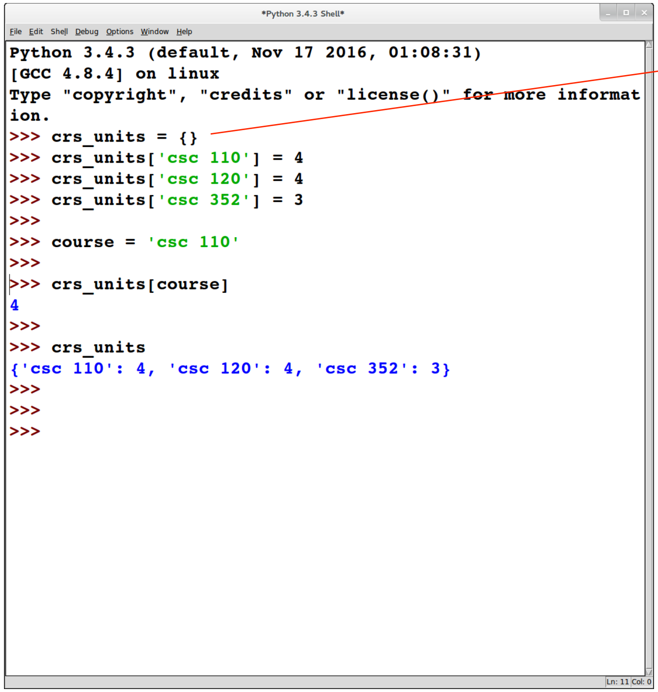
what do you think will be printed out?

python review: dictionaries

Dictionaries

- A dictionary is like an array, but it can be indexed using strings (or numbers, or tuples, or any immutable type)
  - the values used as indexes for a particular dictionary are called its *keys*
  - think of a dictionary as an unordered collection of *key* : *value* pairs
  - empty dictionary: {}
- It is an error to index into a dictionary using a non-existent key

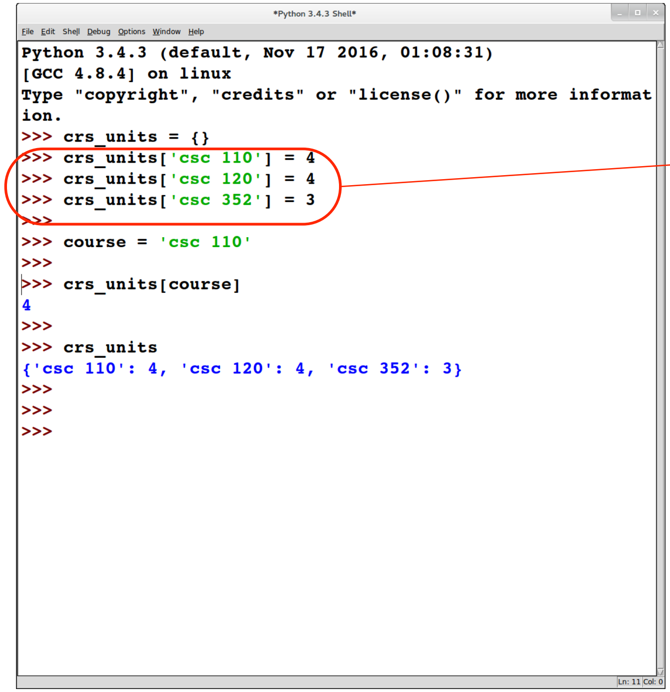
Dictionaries



```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> crs_units = {}
>>> crs_units['csc 110'] = 4
>>> crs_units['csc 120'] = 4
>>> crs_units['csc 352'] = 3
>>>
>>> course = 'csc 110'
>>>
>>> crs_units[course]
4
>>>
>>> crs_units
{'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
>>>
>>>
>>>
```

empty dictionary

Dictionaries



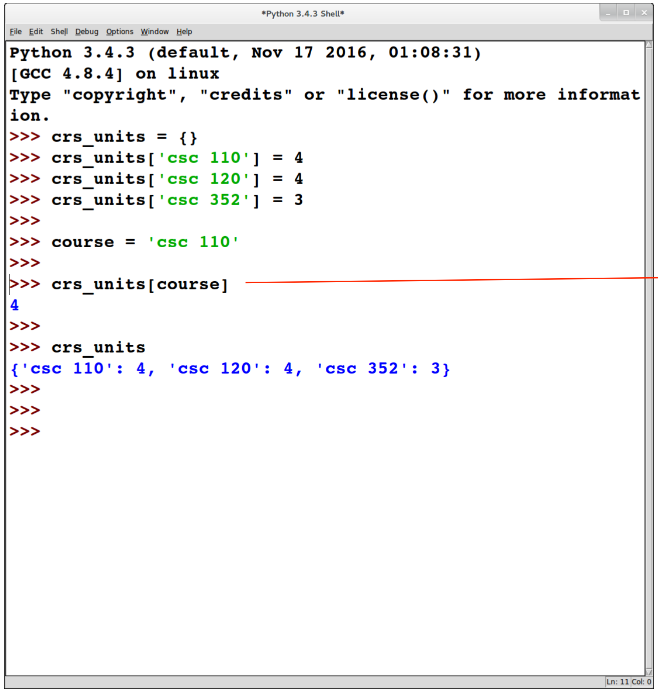
```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> crs_units = {}
>>> crs_units['csc 110'] = 4
>>> crs_units['csc 120'] = 4
>>> crs_units['csc 352'] = 3
>>>
>>> course = 'csc 110'
>>>
>>> crs_units[course]
4
>>>
>>> crs_units
{'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
>>>
>>>
>>>
```

empty dictionary

populating the dictionary

- in this example, one item at a time

Dictionaries



```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> crs_units = {}
>>> crs_units['csc 110'] = 4
>>> crs_units['csc 120'] = 4
>>> crs_units['csc 352'] = 3
>>>
>>> course = 'csc 110'
>>>
>>> crs_units[course]
4
>>>
>>> crs_units
{'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
>>>
>>>
>>>
```

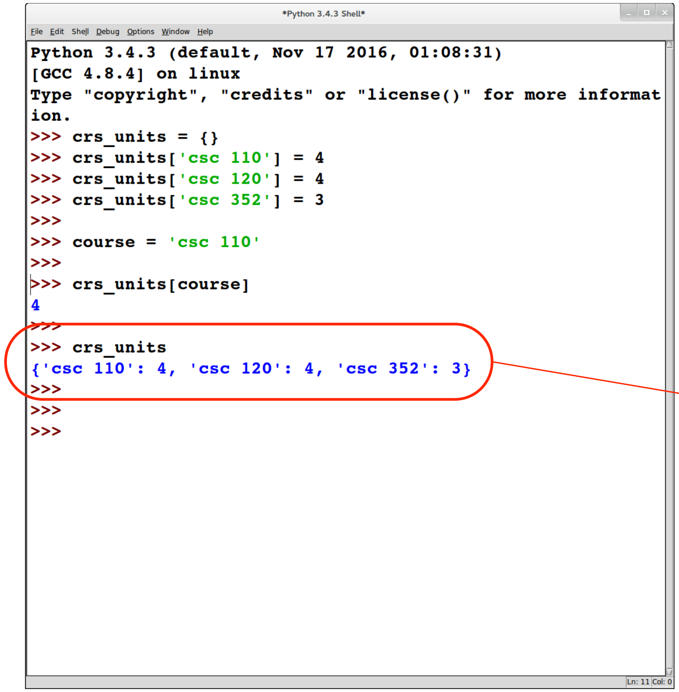
empty dictionary

populating the dictionary

- in this example, one item at a time

looking up the dictionary (indexing)

Dictionaries



```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> crs_units = {}
>>> crs_units['csc 110'] = 4
>>> crs_units['csc 120'] = 4
>>> crs_units['csc 352'] = 3
>>>
>>> course = 'csc 110'
>>>
>>> crs_units[course]
4
>>>
>>> crs_units
{'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
>>>
>>>
>>>
```

empty dictionary

populating the dictionary

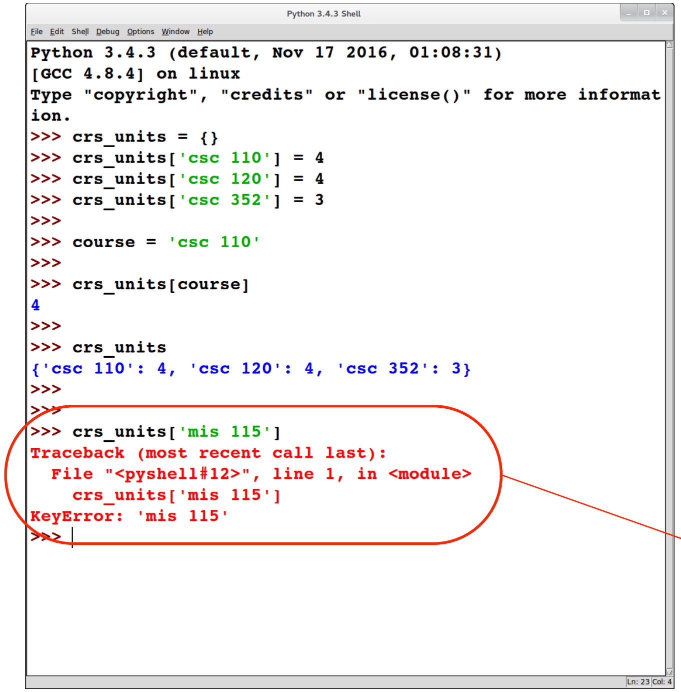
- in this example, one item at a time

looking up the dictionary (indexing)

looking at the dictionary

- we can use this syntax to populate the dictionary too

Dictionaries



```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> crs_units = {}
>>> crs_units['csc 110'] = 4
>>> crs_units['csc 120'] = 4
>>> crs_units['csc 352'] = 3
>>>
>>> course = 'csc 110'
>>>
>>> crs_units[course]
4
>>>
>>> crs_units
{'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
>>>
>>>
>>> crs_units['mis 115']
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    crs_units['mis 115']
KeyError: 'mis 115'
>>> 
```

empty dictionary

populating the dictionary

- in this example, one item at a time

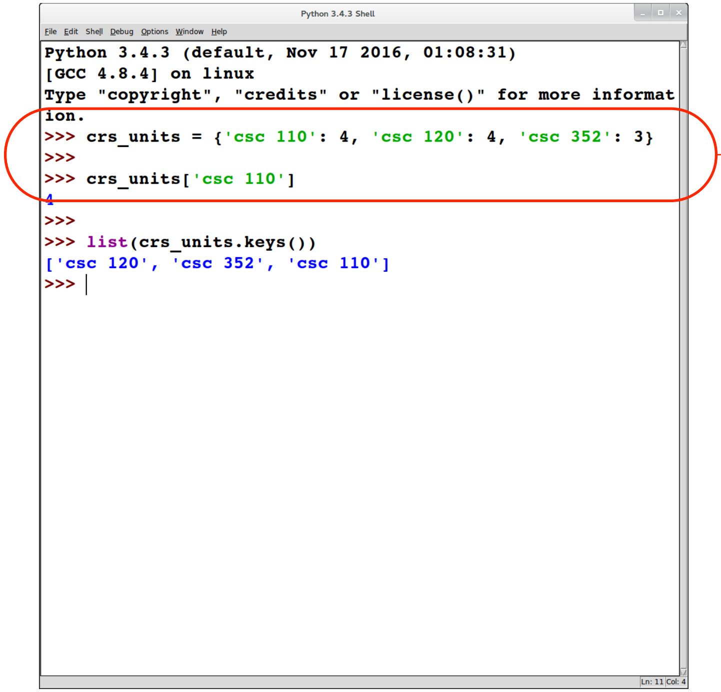
looking up the dictionary (indexing)

looking at the dictionary

- we can use this syntax to populate the dictionary too

indexing with a key not in the dictionary is an error (**`KeyError`**)

Dictionaries

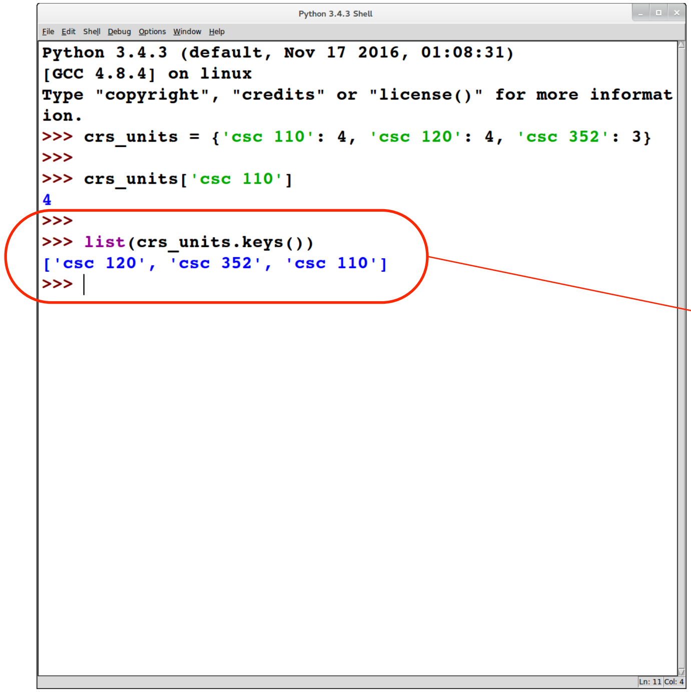


```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> crs_units = {'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
>>>
>>> crs_units['csc 110']
4
>>>
>>> list(crs_units.keys())
['csc 120', 'csc 352', 'csc 110']
>>> 
```

initializing the dictionary

- in this example, several items at once

Dictionaries



```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> crs_units = {'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
>>>
>>> crs_units['csc 110']
4
>>>
>>> list(crs_units.keys())
['csc 120', 'csc 352', 'csc 110']
>>> 
```

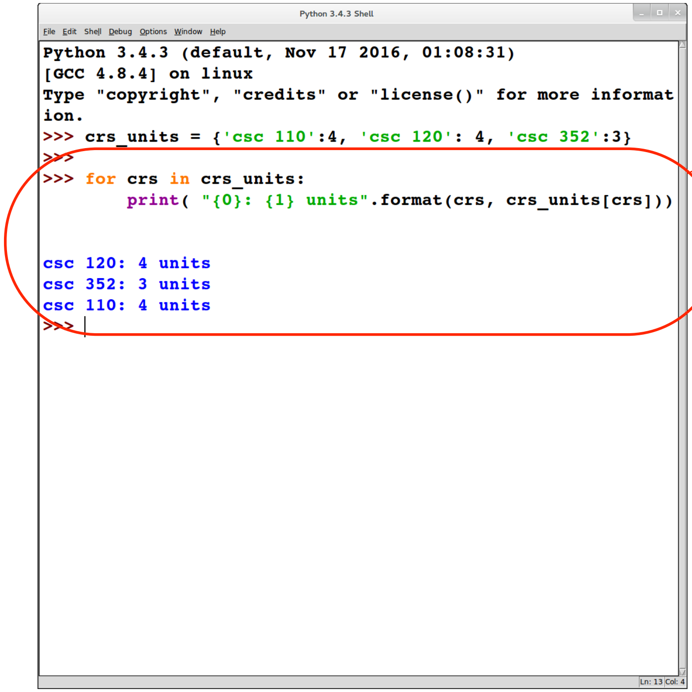
initializing the dictionary

- in this example, several items at once

getting a list of keys in the dictionary

- useful since it's an error to index into a dictionary with a key that is not in it

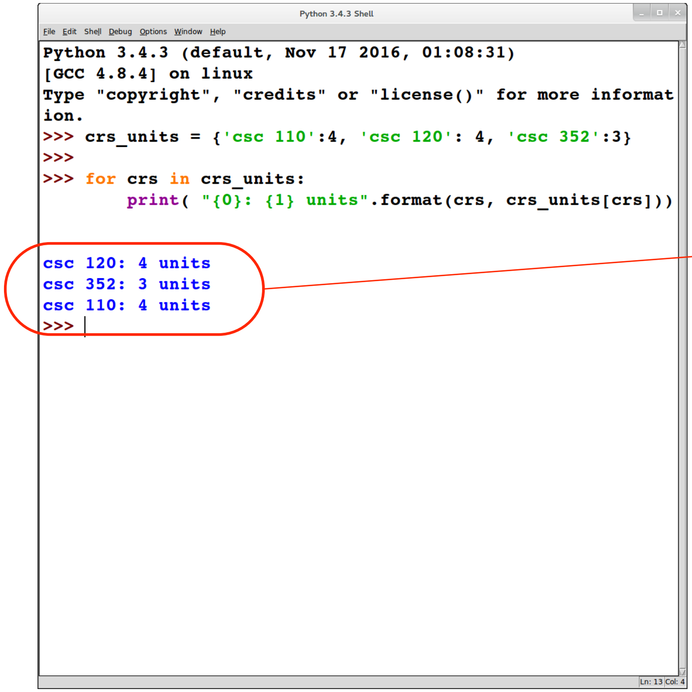
Dictionaries



```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> crs_units = {'csc 110':4, 'csc 120': 4, 'csc 352':3}
>>>
>>> for crs in crs_units:
        print( "{0}: {1} units".format(crs, crs_units[crs]))


csc 120: 4 units
csc 352: 3 units
csc 110: 4 units
>>> 
```

We can use a **for** loop to iterate through a dictionary

Dictionaries



```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> crs_units = {'csc 110':4, 'csc 120': 4, 'csc 352':3}
>>>
>>> for crs in crs_units:
        print( "{0}: {1} units".format(crs, crs_units[crs]))


csc 120: 4 units
csc 352: 3 units
csc 110: 4 units
>>> 
```

We can use a **for** loop to iterate through a dictionary

Notice that this iteration may not list the items in the dictionary in the same order as when they were inserted

EXERCISE

```
>>> crs_units = { 'csc 352' : 3, 'csc 120': 4, 'csc 110': 4 }
>>> for crs in 
        print( "{0} : {1} units".format( crs, crs_units[crs] )

csc 110 : 4 units
csc 120 : 4 units
csc 352 : 3 units
>>>
```

How can we get the dictionary contents to be printed out in sorted order of the keys?
(I.e., what goes in the box?)