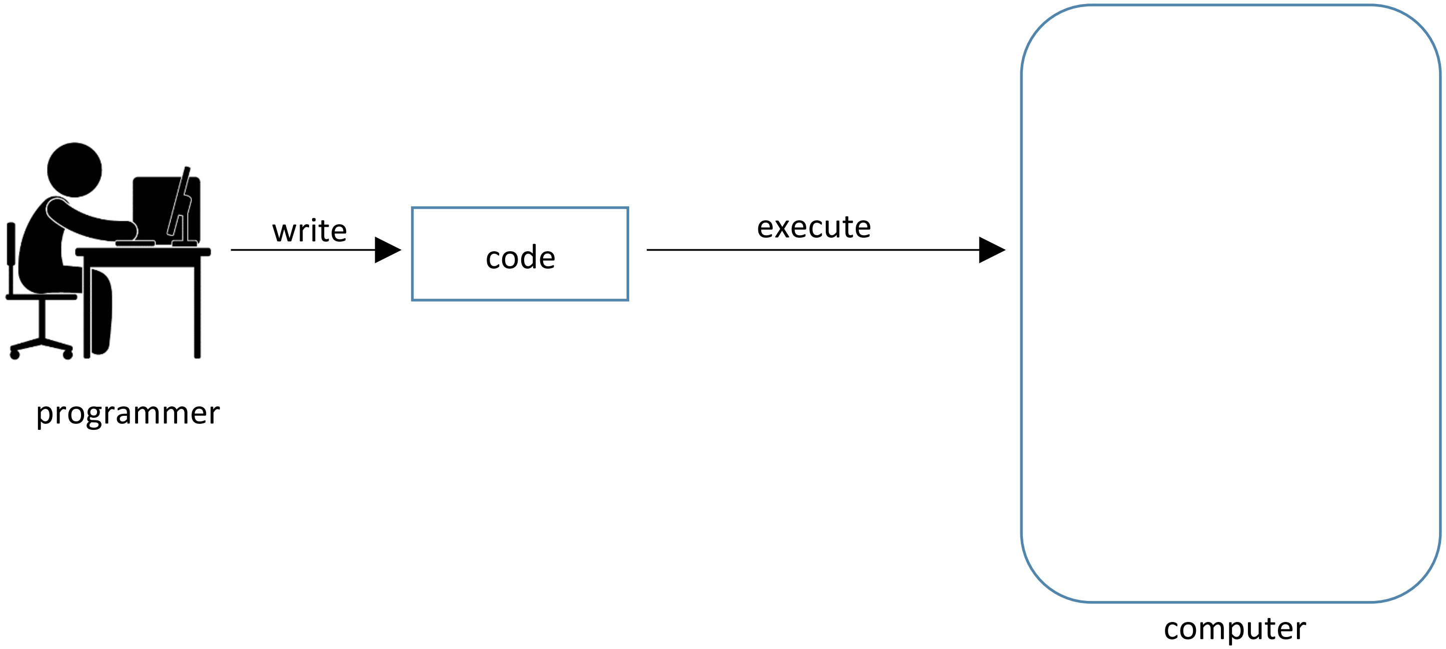
CSc 120
Introduction to Computer Programing II

Adapted from slides by
Dr. Saumya Debray

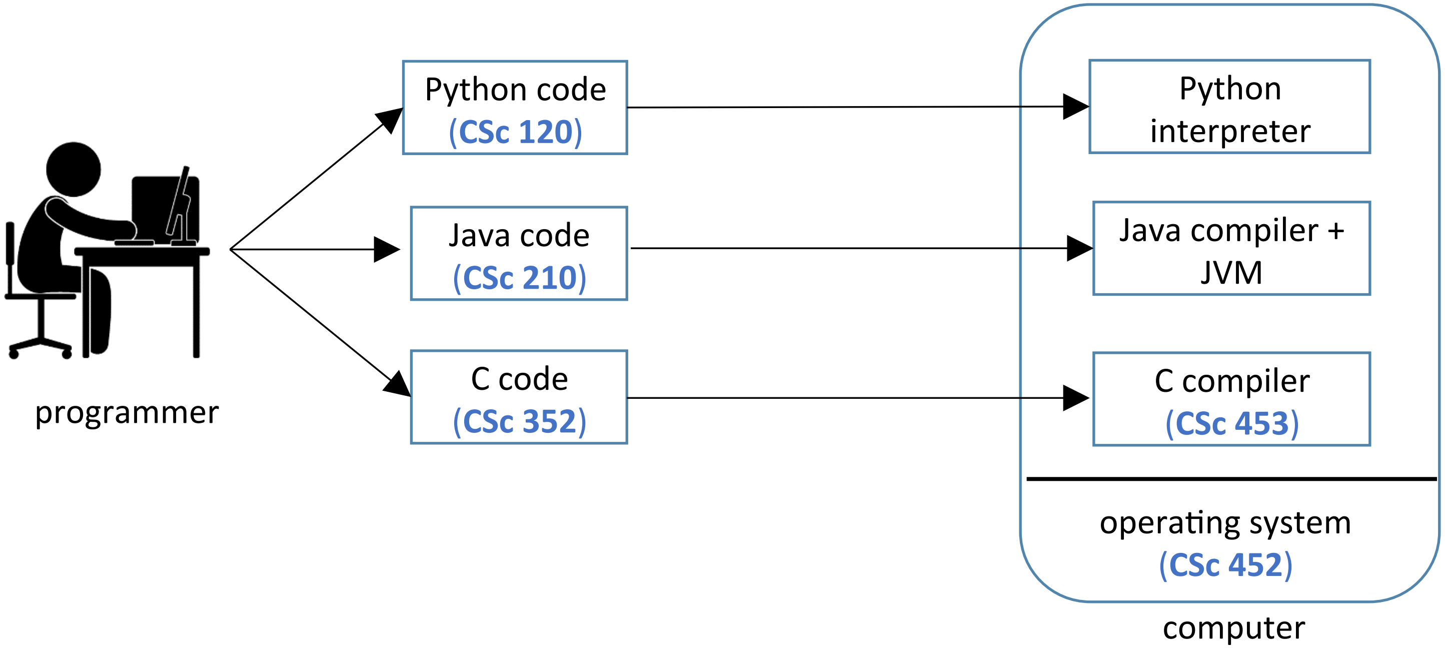
01: Python review

this class in context

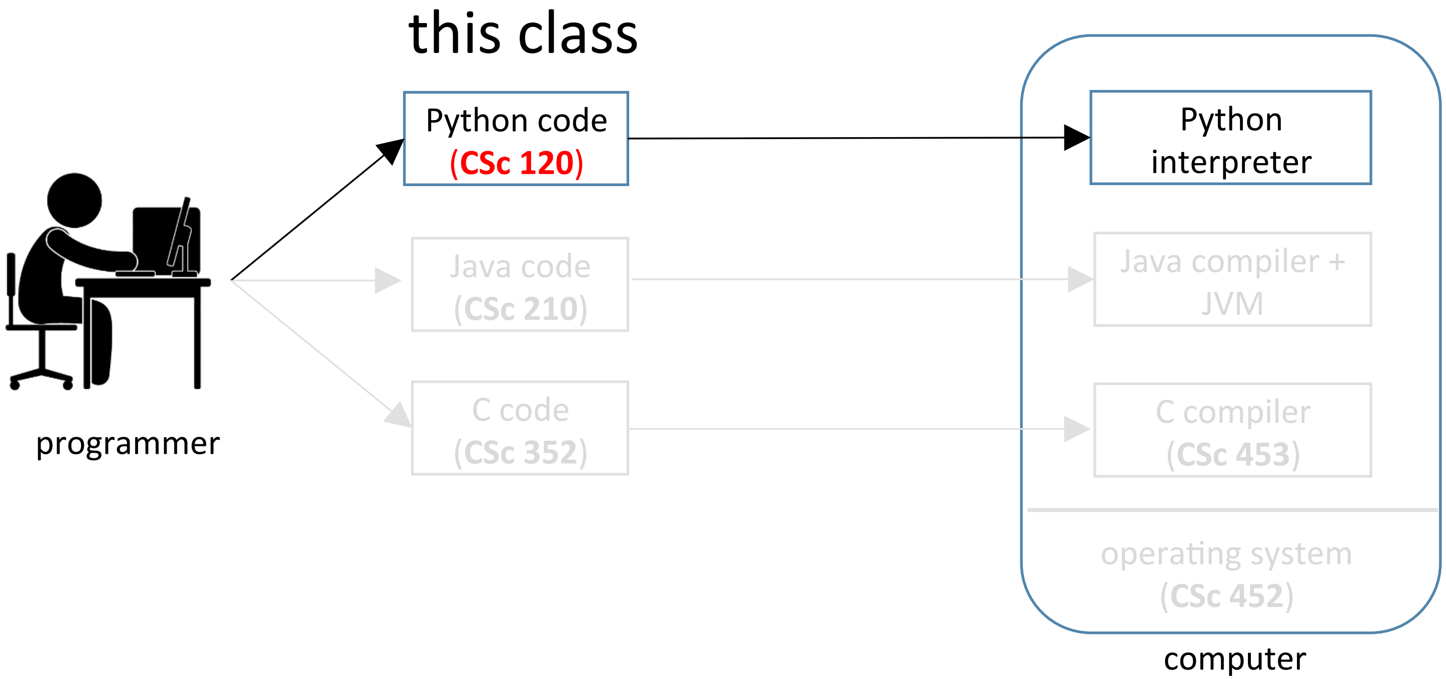
Computer programming



Computer programming



Computer programming



this class

Python code
(CSc 120)

Java code
(CSc 210)

C code
(CSc 352)

programmer

Python
interpreter

Java compiler +
JVM

C compiler
(CSc 453)

operating system
(CSc 452)

computer

getting started

Python language and environment

- Language: Python 3
  - free download from https://www.python.org/downloads
  - documentation available at https://www.python.org/doc
    - tutorial
    - beginner's guide
    - language reference
    - setup and usage, HOWTOs, FAQs

Python language and environment

- Programming environment: idle (or idle3)
  - comes bundled with Python download
  - provides:
    - interactive Python shell
    - debugger
    - execution from a file

Surprises if coming from C, C++, Java

- No variable declarations
- Indentation instead of `{}`
- Flexible `for` loop
- Built-in data structures (lists, dictionaries, tuples, sets)
- Arbitary-precision integers
- Devision differences
- Garbage collection (also in Java)
    - no explicit allocation/deallocation

python review: variables, expressions, assignment

python basics

```
>>> x = 4
>>> y = 5
>>> z = x + y
>>> x
4
>>> y
5
>>> z
9
>>> y = z * 2
>>> y
18
>>>
```

python basics

```
>>> x = 4
>>> y = 5
>>> z = x + y
>>> x
4
>>> y
5
>>> z
9
>>> y = z * 2
>>> y
18
>>>
```

>>> : python interpreter's prompt
black: user input (keyboard)
blue: python interpreter output

python basics

```
>>> x = 4
>>> y = 5
>>> z = x + y
>>> x
4
>>> y
5
>>> z
9
>>> y = z * 2
>>> y
18
>>>
```

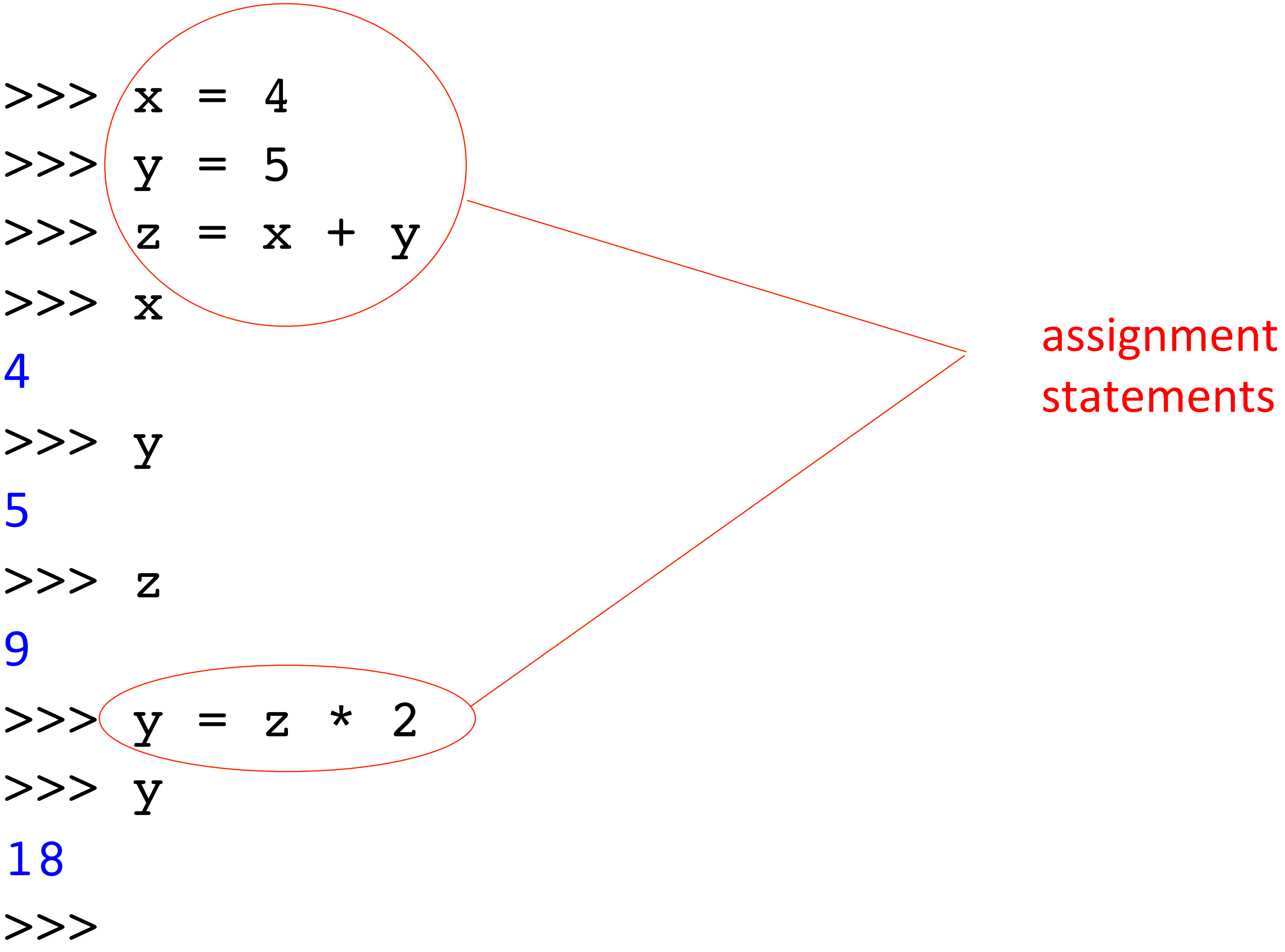
variables

python basics

```
>>> x = 4
>>> y = 5
>>> z = x + y
>>> x
4
>>> y
5
>>> z
9
>>> y = z * 2
>>> y
18
>>>
```

expressions

python basics



```
>>> x = 4
>>> y = 5
>>> z = x + y
>>> x
4
>>> y
5
>>> z
9
>>> y = z * 2
>>> y
18
>>>
```

assignment statements

python basics

```
>>> x = 4
>>> y = 5
>>> z = x + y
>>> x
4
>>> y
5
>>> z
9
>>> y = z * 2
>>> y
18
>>>
```

typing in an expression causes its value to be printed

python basics

```
>>> x = 4
>>> y = 5
>>> z = x + y
>>> x
4
>>> y
5
>>> z
9
>>> y = z * 2
>>> y
18
>>>
```

- variables:
  - names begin with letter or '_'
  - don't have to be declared in advance
    - type determined at runtime
- expressions:
  - all the usual arithmetic operators

Multiple (aka parallel) assignment

```
>>>
>>> x, y, z = 11, 22, 33
>>> x
11
>>> y
22
>>> z
33
>>>
```

Assigns to multiple variables at the same time

$$x_1, x_2, \ldots, x_n = exp_1, exp_2, \ldots, exp_n$$

Behavior:

1. $exp_1, \ldots, exp_n$ evaluated (L-to-R)
2. $x_1, \ldots, x_n$ are assigned (L-to-R)

EXERCISE

```
>>> x = 3
>>> y = 4
>>> z = (2*x – 1 == y+1)
>>> z
```

← *what value is printed out for z?*

EXERCISE

```
>>> x = 3
>>> y = 4
>>> sum, diff, prod  =  x + y,  x – y,  x * y
>>> prod+diff
```

← *what is the value printed out?*

python review:
reading user input I:
input()

Reading user input I: **input()**

```
>>> x = input()
13579
>>> x
'13579'
>>> y = input('Type some input: ')
Type some input: 23
>>> y
'23'
>>> z = input('More input: ')
More input: 567
>>> z
'567'
>>>
```

Reading user input I: **input()**

```
>>> x = input()
13579
>>> x
'13579'
>>> y = input('Type some input: ')
Type some input: 23
>>> y
'23'
>>> z = input('More input: ')
More input: 567
>>> z
'567'
>>>
```

input statement:

- reads input from the keyboard
- returns the value read
  - (a string)

Reading user input I: **input()**

```
>>> x = input()
13579
>>> x
'13579'
>>> y = input('Type some input: ')
Type some input: 23
>>> y
'23'
>>> z = input('More input: ')
More input: 567
>>> z
'567'
>>>
```

input statement:

- reads input from the keyboard
- returns the value read
  - (a string)

- takes an optional argument
  - if provided, serves as a prompt

Reading user input I: **input()**

```
>>>
>>> x = input()
12
>>> x
'12'
>>> y = x / 2
Traceback (most recent call last):
  File "<pyshell#59>", line 1, in <module>
    y = x / 2
TypeError: unsupported operand type(s) for /: 'str' and 'int'
>>>
```

the value read in is represented as a string

- string ≡ sequence of characters

Reading user input I: **input()**

```
>>>
>>> x = input()
12
>>> x
'12'
>>> y = x / 2
Traceback (most recent call last):
  File "<pyshell#59>", line 1, in <module>
    y = x / 2
TypeError: unsupported operand type(s) for /: 'str' and 'int'
>>>
```

the value read in is represented as a string

- string ≡ sequence of characters

- TypeError: indicate an error due to wrong type

Reading user input I: **input()**

```
>>>
>>> x = input()
12
>>> x
'12'
>>> y = x / 2
Traceback (most recent call last):
  File "<pyshell#59>", line 1, in <module>
    y = x / 2
TypeError: unsupported operand type(s) for /: 'str' and 'int'
>>> y = int (x) / 2
>>> y
6.0
>>>
```

the value read in is represented as a string

- string ≡ sequence of characters
- TypeError: indicates an error due to a wrong type

- Fix: explicit type conversion

python review: basics of strings

Basics of strings

```
>>> x = "abcd"
>>> y = 'efgh'
>>> z = "efgh"
>>>
```

Basics of strings

```
>>> x = "abcd"
>>> y = 'efgh'
>>> z = "efgh"
>>>
```

either single-quotes (at both ends)
or double-quotes (at both ends)

Basics of strings

```
>>> text = input('Enter a string: ')
Enter a string: abcdefghi
>>> text
'abcdefghi'
>>> text[0]
'a'
>>> text[1]
'b'
>>> text[27]
Traceback (most recent call last):
  File "<pyshell#153>", line 1, in <module>
    text[27]
IndexError: string index out of range
>>>
```

a string is a sequence (array) of characters

- we can index into a string to get the characters

Basics of strings

```
>>> text = input('Enter a string: ')
Enter a string: abcdefghi
>>> text
'abcdefghi'
>>> text[0]
'a'
>>> text[1]
'b'
>>> text[27]
Traceback (most recent call last):
  File "<pyshell#153>", line 1, in <module>
    text[27]
IndexError: string index out of range
>>>
```

a string is a sequence (array) of characters

- we can index into a string to get the characters

indexing beyond the end of the string gives an `IndexError` error

Basics of strings

```
>>> text = input('Enter a string: ')
Enter a string: abcdefghi
>>> text
'abcdefghi'
>>> text[0]
'a'
>>> text[1]
'b'
>>> text[27]
Traceback (most recent call last):
  File "<pyshell#153>", line 1, in <module>
    text[27]
IndexError: string index out of range
>>>
```

a string is a sequence (array) of characters

- we can index into a string to get the characters
- each character is returned as a string of length 1

Intuitively, a *character* is a single letter, digit, punctuation mark, etc.

E.g.: 'a'
'5'
'$'

Basics of strings

```
>>> x = '0123456789'
>>>
>>> x[0]
'0'
>>> x[1]
'1'
>>> x[2]
'2'
>>>
>>> x[-1]
'9'
>>> x[-2]
'8'
>>> x[-3]
'7'
>>>
```

x[*i*] : if *i* ≥ 0 (i.e., non-negative values):

- indexing is done from the beginning of the string
- the first letter has index 0

x[*i*] : if *i* < 0 (i.e., negative values):

- indexing is done from the end of the string
- the last letter has index -1

Basics of strings

```
>>> x = '0123456789'
>>>
>>> x[0]
'0'
>>> x[1]
'1'
>>> x[2]
'2'
>>>
>>> x[-1]
'9'
>>> x[-2]
'8'
>>> x[-3]
'7'
>>>
```

x[i] : if $i \geq 0$ (i.e., non-negative values):

- indexing is done from the beginning of the string
- the first letter has index 0

x[i] : if $i < 0$ (i.e., negative values):

- indexing is done from the end of the string
- the last letter has index -1

EXERCISE

```
>>> x = 'a'
>>> x == x[0]
```

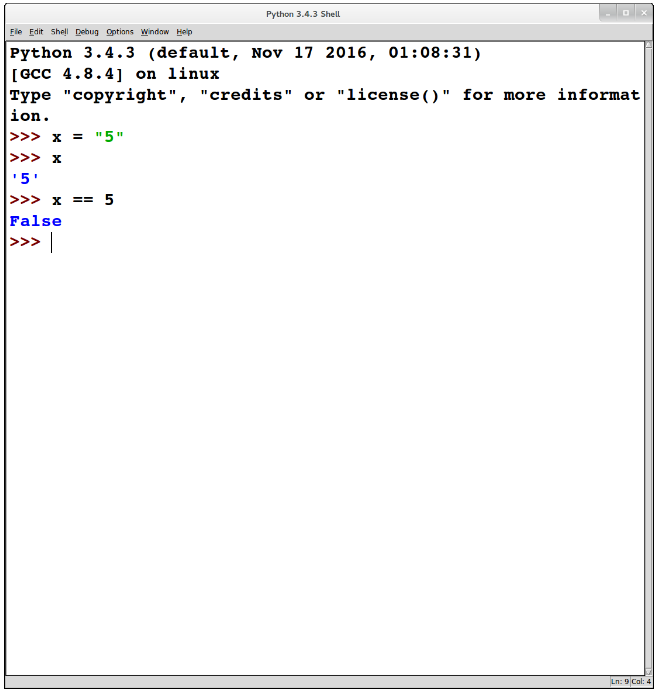
what do you think will be printed here?

EXERCISE

```
>>> x = 'apple'
>>> x[2] == x[-2]
```

what do you think will be printed here?

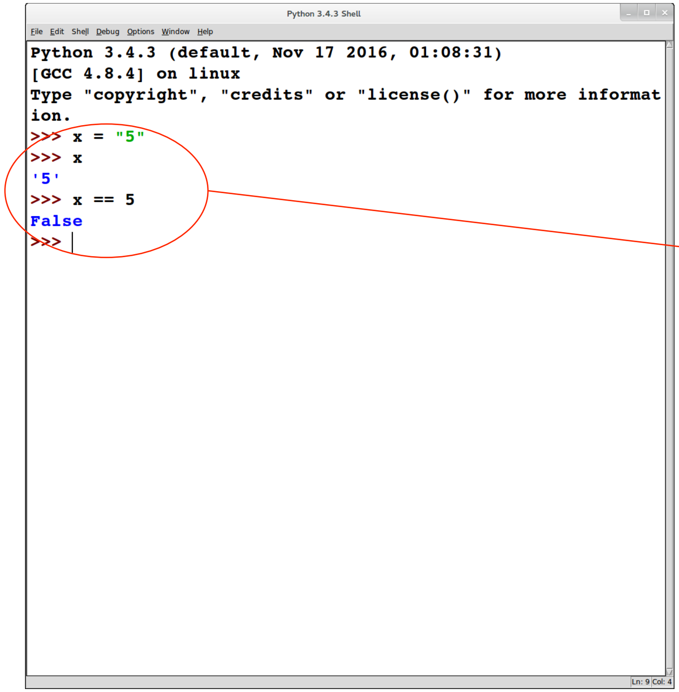
Basics of strings



```
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 = "5"
>>> x
'5'
>>> x == 5
False
>>> 
```

Inside a computer, a character is represented as a number
(its "ASCII value")

Basics of strings



```
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 = "5"
>>> x
'5'
>>> x == 5
False
>>> 
```

Inside a computer, a character is represented as a number (its "ASCII value")

the ASCII value of a digit is not the same as the digit itself:

'5' ≠ 5

EXERCISE

```
>>> x = 27
>>> y = 'x'
>>> x == y
```

What do you think will be printed here?
Why?

Basics of strings

```
>>> x = input()
abcDE_fgHIJ_01234
>>> x
'abcDE_fgHIJ_01234'
>>>
>>> len(x)
17
>>> y = x.lower()
>>> y
'abcde_fghij_01234'
>>>
>>> x = y.upper()
>> x
'ABCDE_FGHIJ_01234'
>>>
```

len(x) : length of a string x

Basics of strings

```
>>> x = input()
abcDE_fgHIJ_01234
>>> x
'abcDE_fgHIJ_01234'
>>>
>>> len(x)
17
>>> y = x.lower()
>>> y
'abcde_fghij_01234'
>>>
>>> x = y.upper()
>> x
'ABCDE_FGHIJ_01234'
>>>
```

len(x) : length of a string x

x.lower(), x.upper() : case conversion on the letters in a string x

- note that non-letter characters are not affected

Basics of strings

```
>>> x = input()
abcDE_fgHIJ_01234
>>> x
'abcDE_fgHIJ_01234'
>>>
>>> len(x)
17
>>> y = x.lower()
>>> y
'abcde_fghij_01234'
>>>
>>> x = y.upper()
>> x
'ABCDE_FGHIJ_01234'
>>>
```

len(x) : length of a string x

x.lower(), x.upper() : case conversion on the letters in a string x

- note that non-letter characters are not affected

Python supports a wide variety of string operations

- see www.tutorialspoint.com/python3/ python_strings.htm

Basics of strings

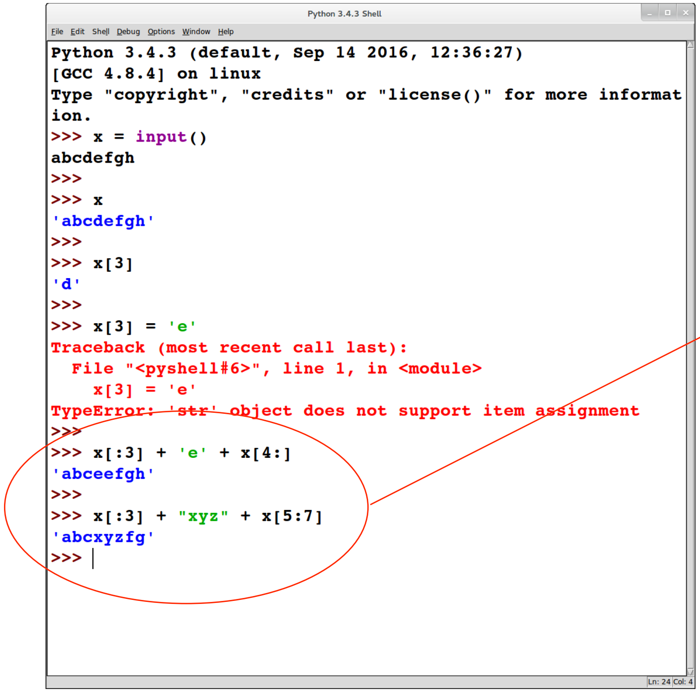
```
>>> x = input()
abcdefgh
>>> x
'abcdefgh'
>>> x[3]
'd'
>>>
>>> x[3] = 'z'
Traceback (most recent call last):
  File "<pyshell#193>", line 1, in <module>
    x[3] = 'z'
TypeError: 'str' object does not support item assignment
>>>
```

Basics of strings

```
>>> x = input()
abcdefgh
>>> x
'abcdefgh'
>>> x[3]
'd'
>>>
>>> x[3] = 'z'
Traceback (most recent call last):
  File "<pyshell#193>", line 1, in <module>
    x[3] = 'z'
TypeError: 'str' object does not support item assignment
>>>
```

strings are *immutable*, i.e., cannot be modified or updated

Basics of strings

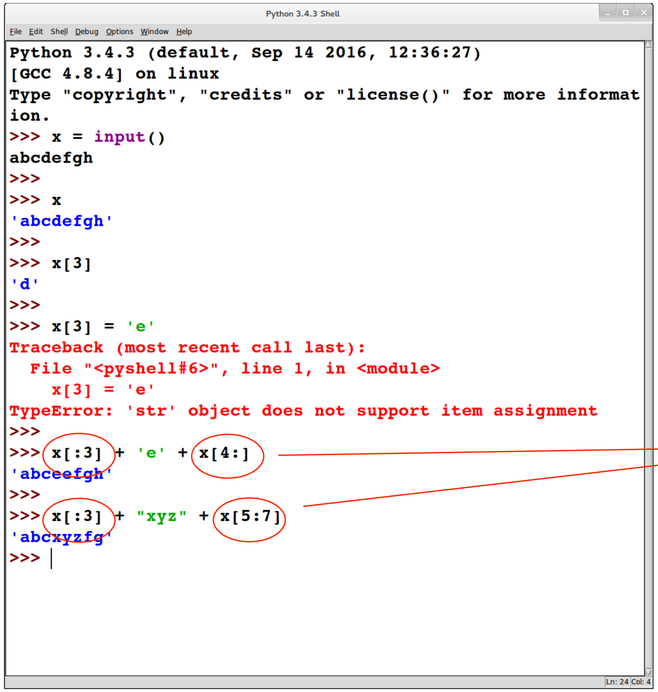


```
Python 3.4.3 (default, Sep 14 2016, 12:36:27)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = input()
abcdefgh
>>>
>>> x
'abcdefgh'
>>>
>>> x[3]
'd'
>>>
>>> x[3] = 'e'
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    x[3] = 'e'
TypeError: 'str' object does not support item assignment
>>>
>>> x[:3] + 'e' + x[4:]
'abceefgh'
>>>
>>> x[:3] + "xyz" + x[5:7]
'abcxyzfg'
>>> 
```

strings are *immutable*, i.e., cannot be modified or updated

to "modify" a string, we have to create a copy of it with the appropriate part(s) replaced by the new values

Basics of strings



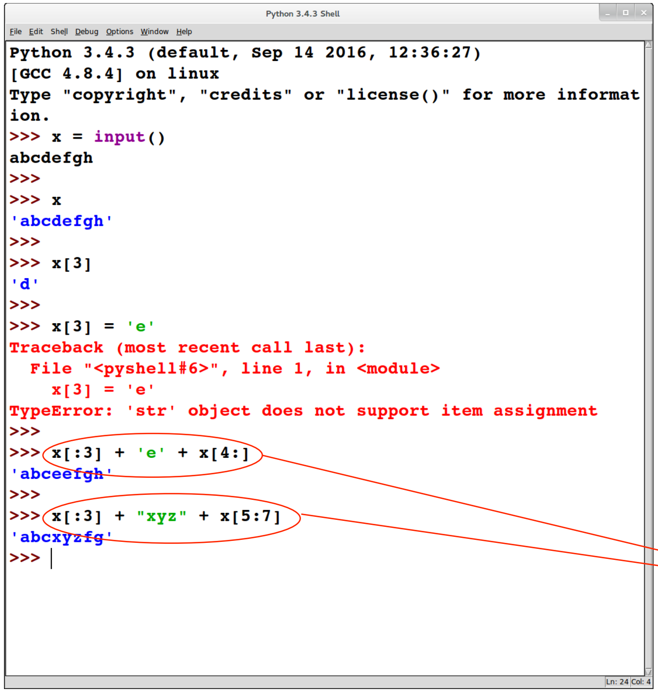
```
Python 3.4.3 (default, Sep 14 2016, 12:36:27)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = input()
abcdefgh
>>>
>>> x
'abcdefgh'
>>>
>>> x[3]
'd'
>>>
>>> x[3] = 'e'
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    x[3] = 'e'
TypeError: 'str' object does not support item assignment
>>>
>>> x[:3] + 'e' + x[4:]
'abceefgh'
>>>
>>> x[:3] + "xyz" + x[5:7]
'abcxyzfg'
>>>
```

strings are *immutable*, i.e., cannot be modified or updated

to "modify" a string, we have to create a copy of it with the appropriate part(s) replaced by the new values

these operations are called "slicing"

Basics of strings



```
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.
>>> x = input()
abcdefgh
>>>
>>> x
'abcdefgh'
>>>
>>> x[3]
'd'
>>>
>>> x[3] = 'e'
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    x[3] = 'e'
TypeError: 'str' object does not support item assignment
>>>
>>> x[:3] + 'e' + x[4:]
'abceefgh'
>>>
>>> x[:3] + "xyz" + x[5:7]
'abcxyzfg'
>>> 
```

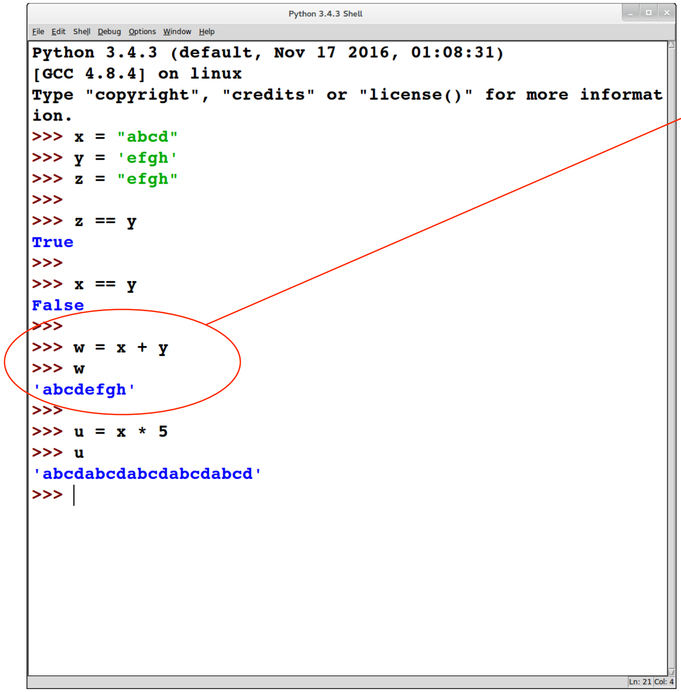
strings are *<u>immutable</u>*, i.e., cannot be modified or updated

to "modify" a string, we have to create a copy of it with the appropriate part(s) replaced by the new values

these operations are called "slicing"

+ applied to strings does concatenation

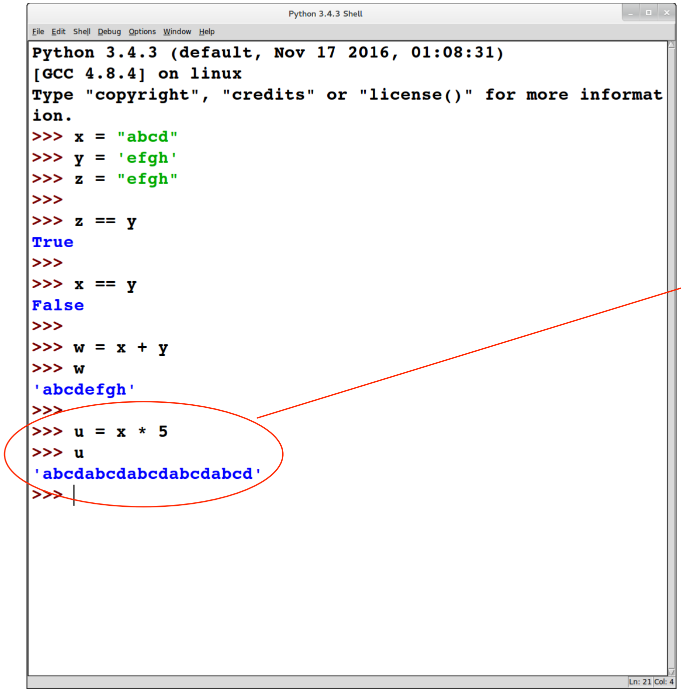
Basics of strings



```
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 = "abcd"
>>> y = 'efgh'
>>> z = "efgh"
>>>
>>> z == y
True
>>>
>>> x == y
False
>>>
>>> w = x + y
>>> w
'abcdefgh'
>>>
>>> u = x * 5
>>> u
'abcdabcdabcdabcdabcd'
>>> 
```

+ applied to strings does concatenation

Basics of strings



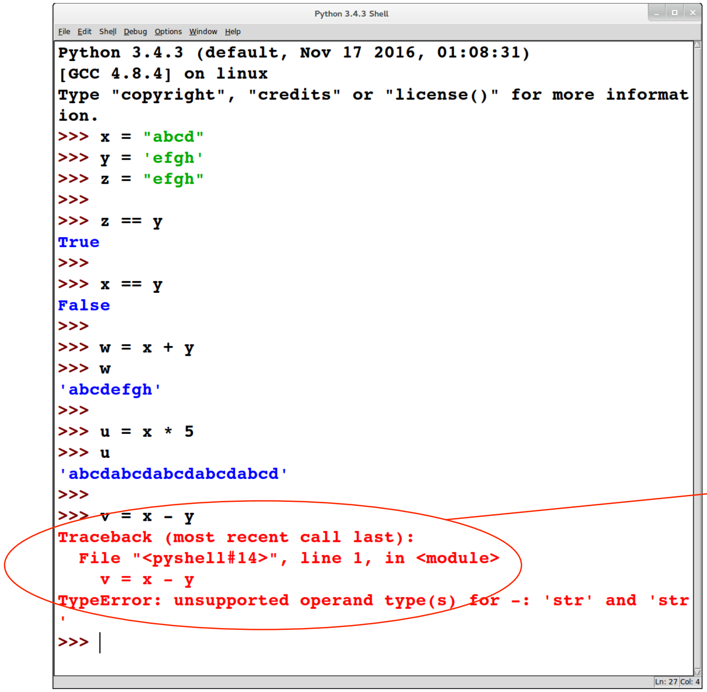
```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = "abcd"
>>> y = 'efgh'
>>> z = "efgh"
>>>
>>> z == y
True
>>>
>>> x == y
False
>>>
>>> w = x + y
>>> w
'abcdefgh'
>>>
>>> u = x * 5
>>> u
'abcdabcdabcdabcdabcd'
>>> 
```

+ applied to strings does concatenation

'*' applied to strings:

- does repeated concatenation *if one argument is a number*
- generates an error otherwise

Basics of strings



```
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 = "abcd"
>>> y = 'efgh'
>>> z = "efgh"
>>>
>>> z == y
True
>>>
>>> x == y
False
>>>
>>> w = x + y
>>> w
'abcdefgh'
>>>
>>> u = x * 5
>>> u
'abcdabcdabcdabcdabcd'
>>>
>>> v = x - y
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    v = x - y
TypeError: unsupported operand type(s) for -: 'str' and 'str
'
>>> 
```

+ applied to strings does concatenation

* applied to strings:

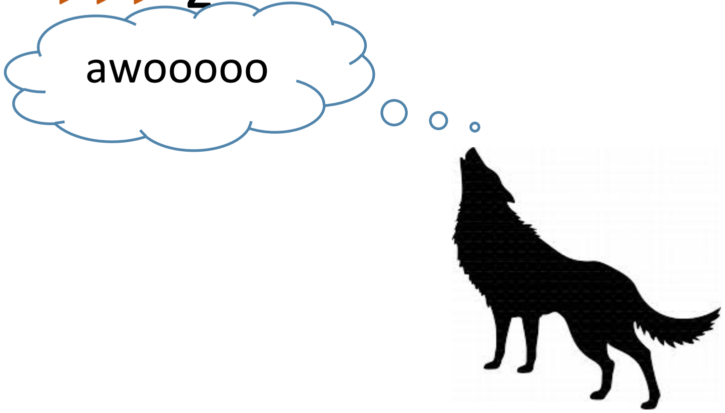
- does repeated concatenation *if one argument is a number*
- generates an error otherwise

not all arithmetic operators carry over to strings

EXERCISE

```
>>> x = "whoa!"
>>> y = x[2] * len(x)
>>> z = x[3] + x[0] + y
>>> z
```

what is printed here?



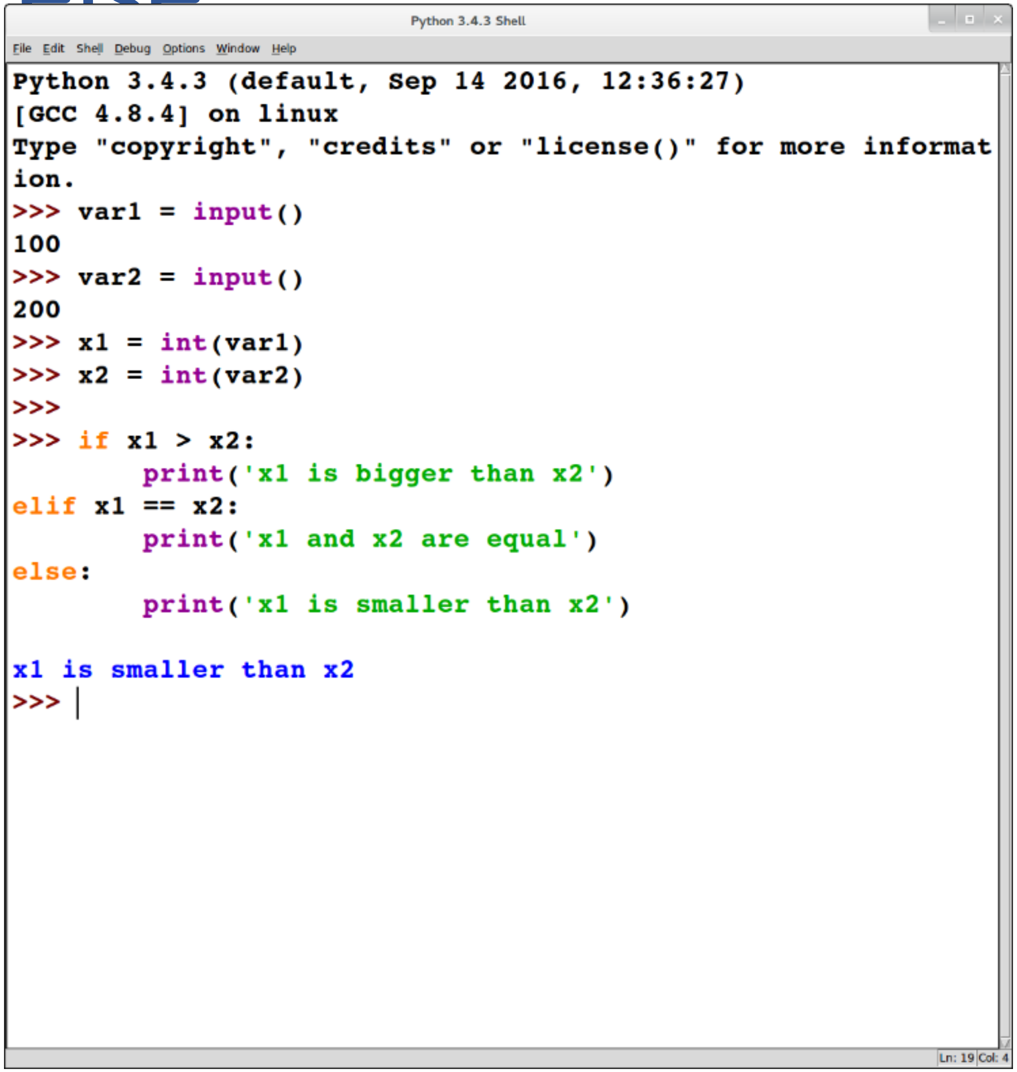
EXERCISE

```
>>> x = input()
>>> y = x + x
>>> int(x) == int(y)
True
```

what input value(s) will cause this to work as shown?

python review: conditionals

Conditional statements: **if/elif/else**



```
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.
>>> var1 = input()
100
>>> var2 = input()
200
>>> x1 = int(var1)
>>> x2 = int(var2)
>>>
>>> if x1 > x2:
        print('x1 is bigger than x2')
elif x1 == x2:
        print('x1 and x2 are equal')
else:
        print('x1 is smaller than x2')

x1 is smaller than x2
>>> 
```

Conditional statements: **if/elif/else**

```
>>> var1 = input()
100
>>> var2 = input()
200
>>> x1 = int(var1)
>>> x2 = int(var2)
>>>
>>> if x1 > x2:
            print('x1 is bigger than x2')
elif x1 == x2:
            print('x1 and x2 are equal')
else:
            print('x1 is smaller than x2')
x1 is smaller than x2
>>>
```

- **if**-statement syntax:

```
if  BooleanExpr :
        stmt
        …
elif BooleanExpr :
        stmt
        …
elif …
        …
else:
        stmt
        …
```

elifs are optional (use as needed)

Conditional statements: **if/elif/else**

```
>>> var1 = input()
100
>>> var2 = input()
200
>>> x1 = int(var1)
>>> x2 = int(var2)
>>>
>>> if x1 > x2:
        print('x1 is bigger than x2')
elif x1 == x2:
        print('x1 and x2 are equal')
else:
        print('x1 is smaller than
x2')
x1 is smaller than x2
>>>
```

- **if**-statement syntax:

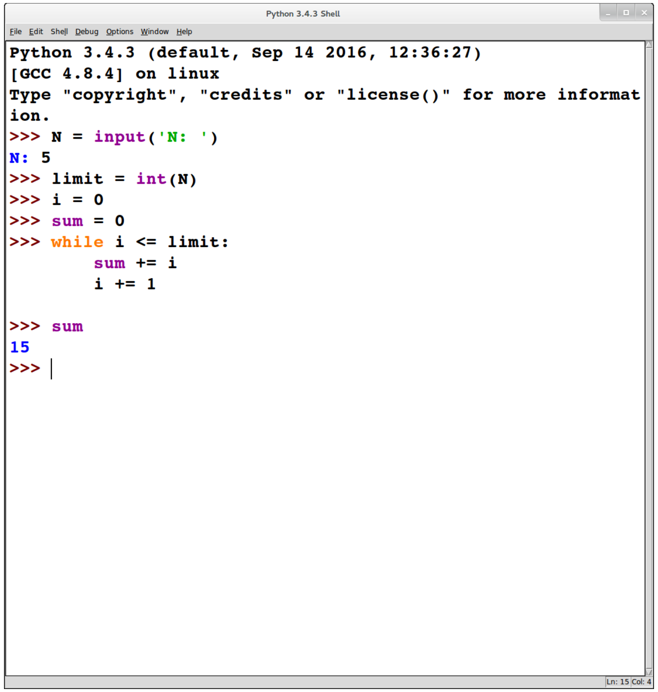
```
if  BooleanExpr :
        stmt
        …
elif BooleanExpr :
        stmt
        …
elif …
        …
else:
        stmt
        …
```

elifs are optional (use as needed)

else is optional

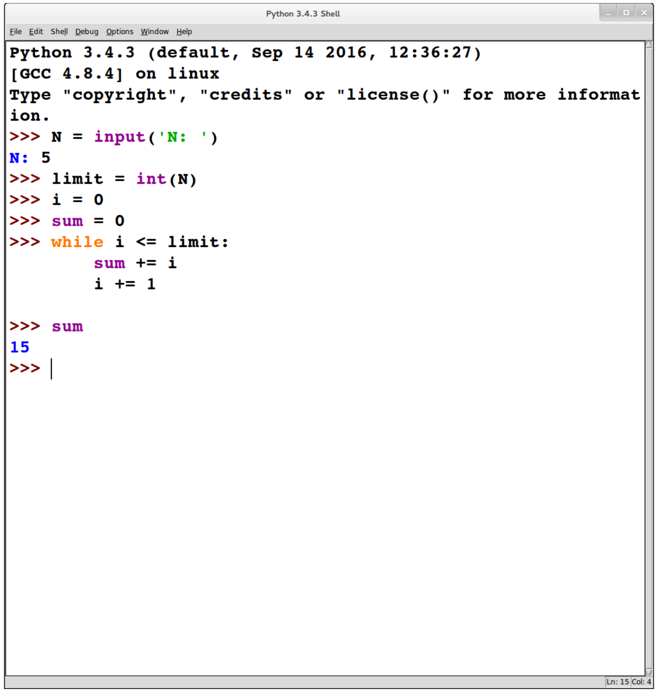
python review: while loops

Loops I: **while**



```
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.
>>> N = input('N: ')
N: 5
>>> limit = int(N)
>>> i = 0
>>> sum = 0
>>> while i <= limit:
	sum += i
	i += 1

>>> sum
15
>>> 
```

Loops I: **while**



```
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.
>>> N = input('N: ')
N: 5
>>> limit = int(N)
>>> i = 0
>>> sum = 0
>>> while i <= limit:
        sum += i
        i += 1

>>> sum
15
>>> 
```

- **while**-statement syntax:

  while *BooleanExpr* **:**

  $stmt_1$

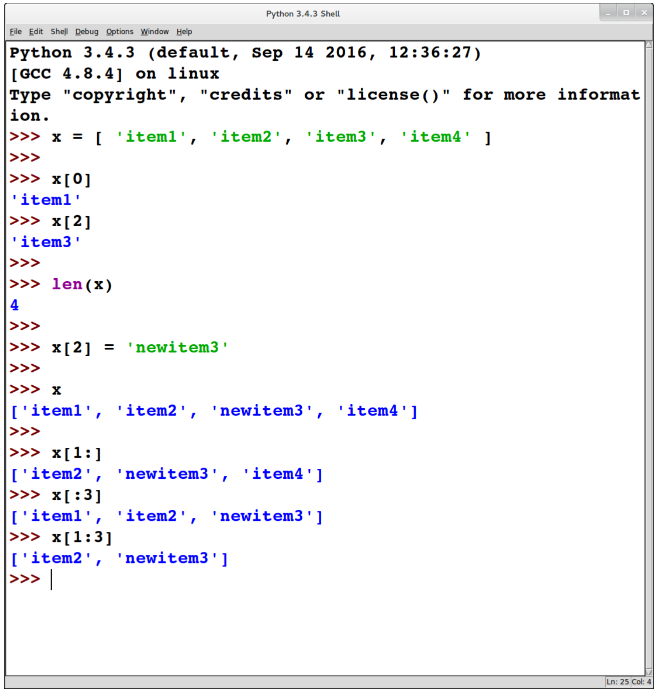
  …

  $stmt_n$

- $stmt_1$ … $stmt_n$ are executed repeatedly as long as *BooleanExpr* is True

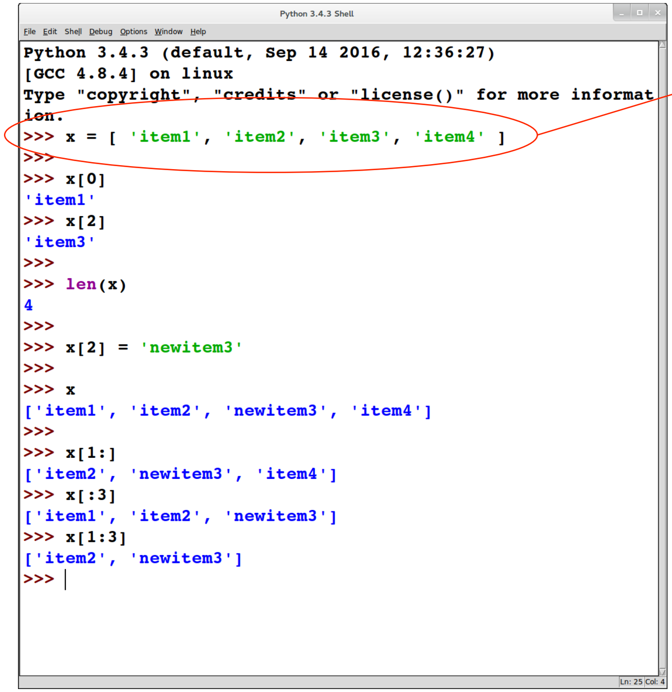
python review:
lists (aka arrays)

Lists



```
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.
>>> x = [ 'item1', 'item2', 'item3', 'item4' ]
>>>
>>> x[0]
'item1'
>>> x[2]
'item3'
>>>
>>> len(x)
4
>>>
>>> x[2] = 'newitem3'
>>>
>>> x
['item1', 'item2', 'newitem3', 'item4']
>>>
>>> x[1:]
['item2', 'newitem3', 'item4']
>>> x[:3]
['item1', 'item2', 'newitem3']
>>> x[1:3]
['item2', 'newitem3']
>>> 
```

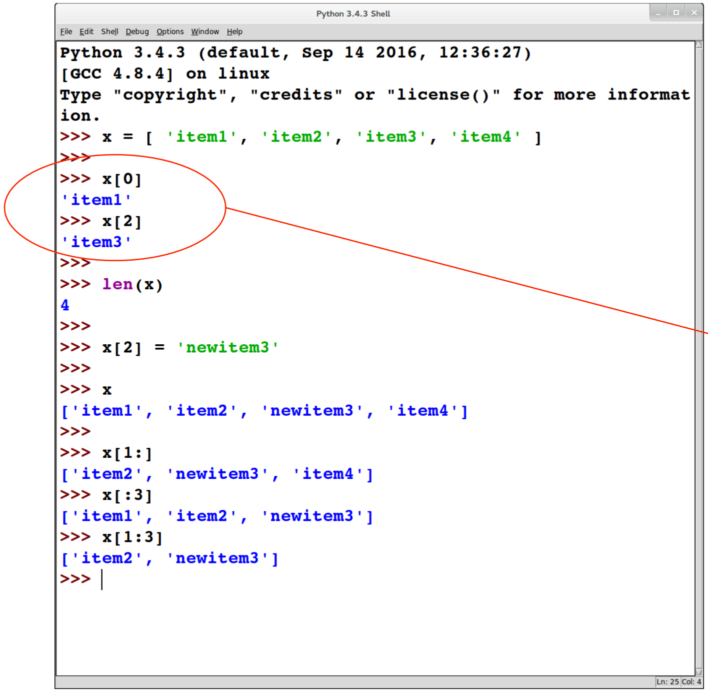
Lists



```
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.
>>> x = [ 'item1', 'item2', 'item3', 'item4' ]
>>>
>>> x[0]
'item1'
>>> x[2]
'item3'
>>>
>>> len(x)
4
>>>
>>> x[2] = 'newitem3'
>>>
>>> x
['item1', 'item2', 'newitem3', 'item4']
>>>
>>> x[1:]
['item2', 'newitem3', 'item4']
>>> x[:3]
['item1', 'item2', 'newitem3']
>>> x[1:3]
['item2', 'newitem3']
>>> 
```

a list (or array) is a sequence of values

Lists



```
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.
>>> x = [ 'item1', 'item2', 'item3', 'item4' ]
>>>
>>> x[0]
'item1'
>>> x[2]
'item3'
>>>
>>> len(x)
4
>>>
>>> x[2] = 'newitem3'
>>>
>>> x
['item1', 'item2', 'newitem3', 'item4']
>>>
>>> x[1:]
['item2', 'newitem3', 'item4']
>>> x[:3]
['item1', 'item2', 'newitem3']
>>> x[1:3]
['item2', 'newitem3']
>>> 
```

a list (or array) is a sequence of values

accessing list elements (i.e., indexing), computing length: similar to strings

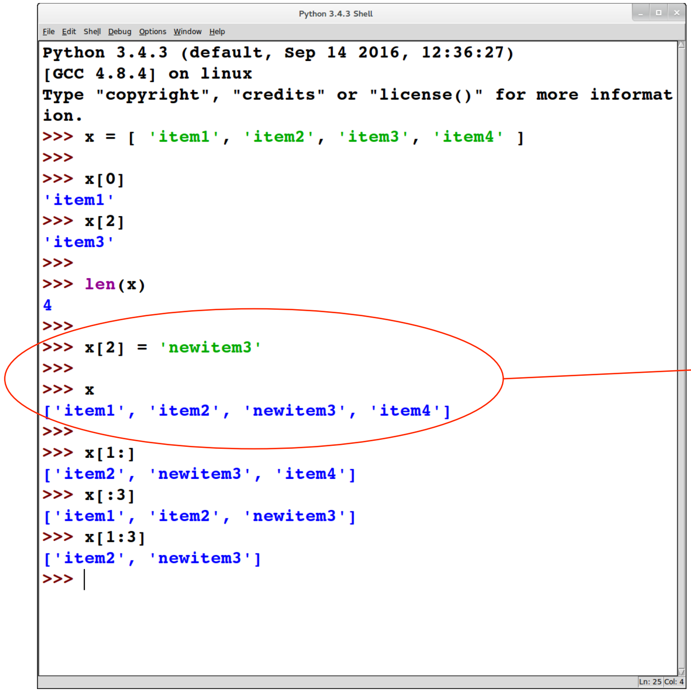
- non-negative index values (≥ 0) index from the front of the list
  - the first element has index 0
- negative index values index from the end of the list
  - the last element has index -1

EXERCISE

```
>>> x = [ “abc”, “def”, “ghi”, “jkl” ]
>>> x[1] + x[-1]
```

what do you think will be printed here?

Lists



```
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.
>>> x = [ 'item1', 'item2', 'item3', 'item4' ]
>>>
>>> x[0]
'item1'
>>> x[2]
'item3'
>>>
>>> len(x)
4
>>>
>>> x[2] = 'newitem3'
>>>
>>> x
['item1', 'item2', 'newitem3', 'item4']
>>>
>>> x[1:]
['item2', 'newitem3', 'item4']
>>> x[:3]
['item1', 'item2', 'newitem3']
>>> x[1:3]
['item2', 'newitem3']
>>> 
```

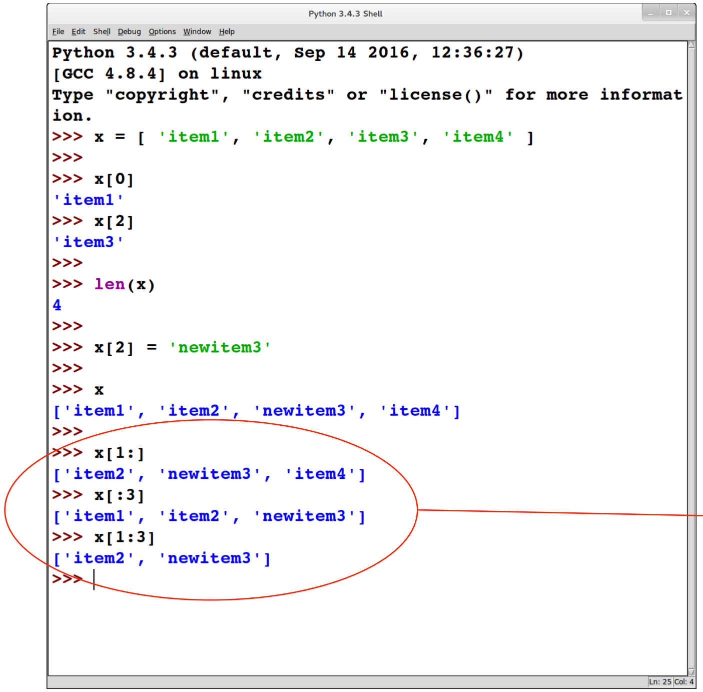
a list (or array) is a sequence of values

accessing list elements (i.e., indexing), computing length: similar to strings

lists are ***mutable***, i.e., can be modified or updated

- different from strings

Lists



```
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.
>>> x = [ 'item1', 'item2', 'item3', 'item4' ]
>>>
>>> x[0]
'item1'
>>> x[2]
'item3'
>>>
>>> len(x)
4
>>>
>>> x[2] = 'newitem3'
>>>
>>> x
['item1', 'item2', 'newitem3', 'item4']
>>>
>>> x[1:]
['item2', 'newitem3', 'item4']
>>> x[:3]
['item1', 'item2', 'newitem3']
>>> x[1:3]
['item2', 'newitem3']
>>> 
```

a list (or array) is a sequence of values

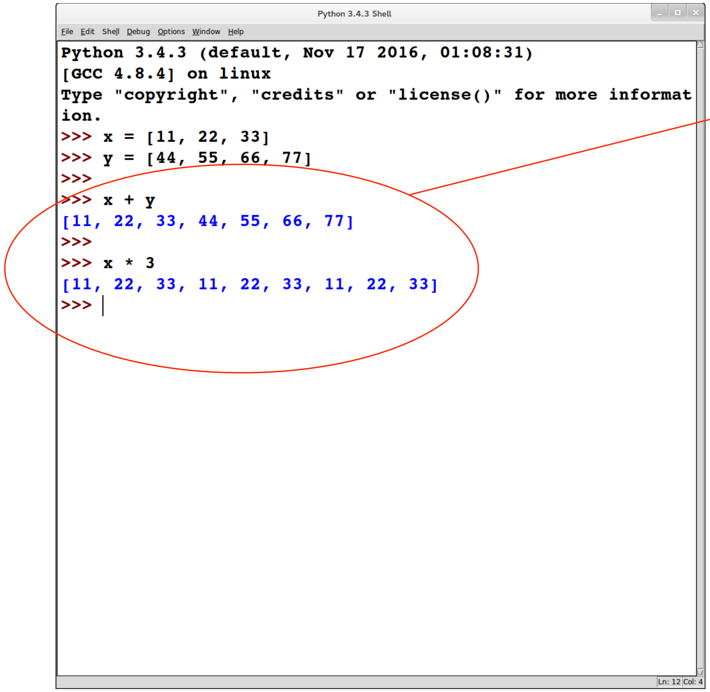
accessing list elements (i.e., indexing), computing length: similar to strings

lists are ***mutable***, i.e., can be modified or updated

- different from strings

slicing : similar to strings

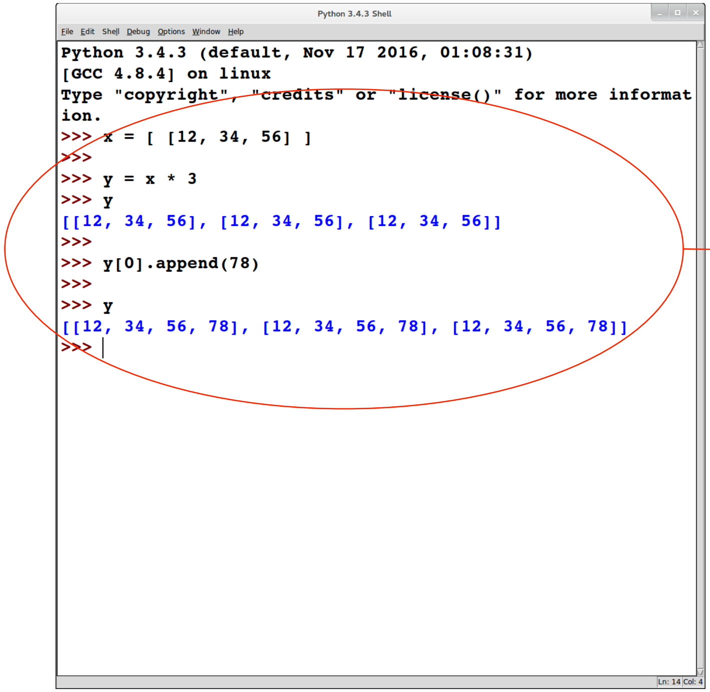
Lists



```
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 = [11, 22, 33]
>>> y = [44, 55, 66, 77]
>>>
>>> x + y
[11, 22, 33, 44, 55, 66, 77]
>>>
>>> x * 3
[11, 22, 33, 11, 22, 33, 11, 22, 33]
>>> 
```

concatenation (+ and *) : similar to strings

Lists



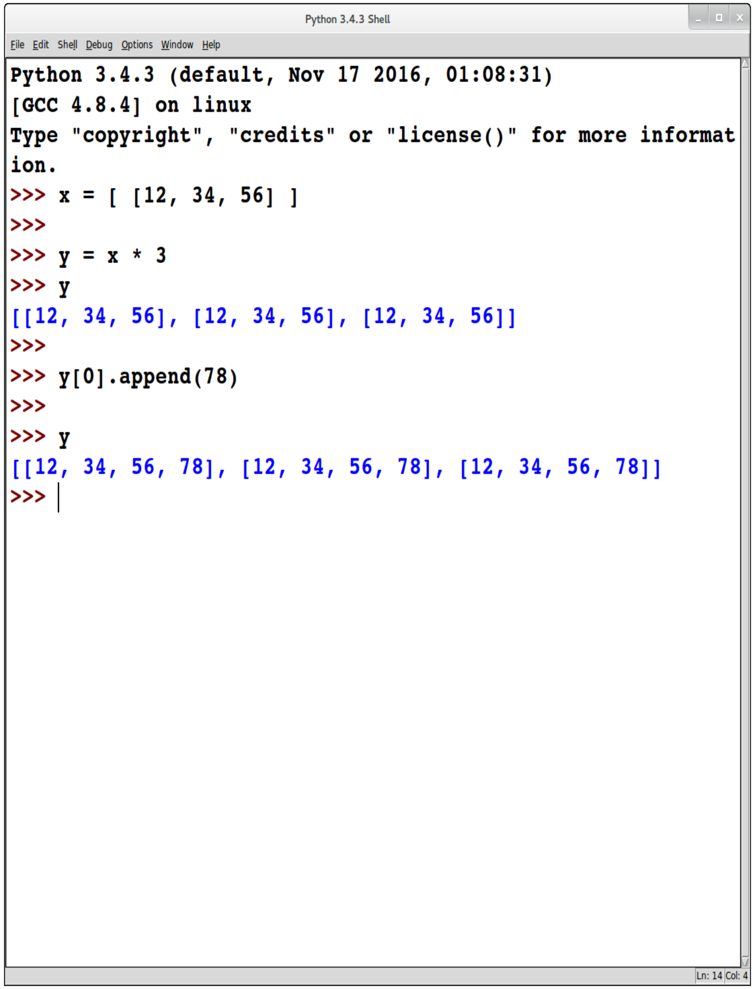
```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = [ [12, 34, 56] ]
>>>
>>> y = x * 3
>>> y
[[12, 34, 56], [12, 34, 56], [12, 34, 56]]
>>>
>>> y[0].append(78)
>>>
>>> y
[[12, 34, 56, 78], [12, 34, 56, 78], [12,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

concatenation (+ and *) : similar to strings

these operators create “shallow” copies

- due to list mutability, this can cause unexpected behavior

Lists

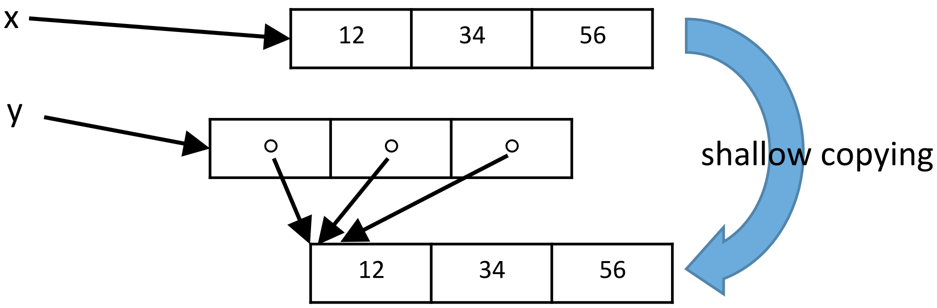


```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = [ [12, 34, 56] ]
>>>
>>> y = x * 3
>>> y
[[12, 34, 56], [12, 34, 56], [12, 34, 56]]
>>>
>>> y[0].append(78)
>>>
>>> y
[[12, 34, 56, 78], [12, 34, 56, 78], [12,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

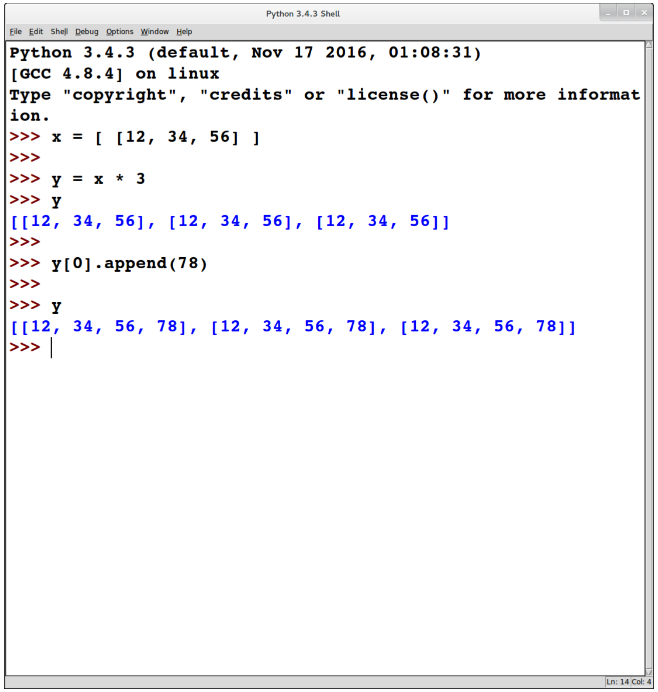
concatenation (+ and *) : similar to strings

these operators create “shallow” copies

- due to list mutability, this can cause unexpected behavior



Lists

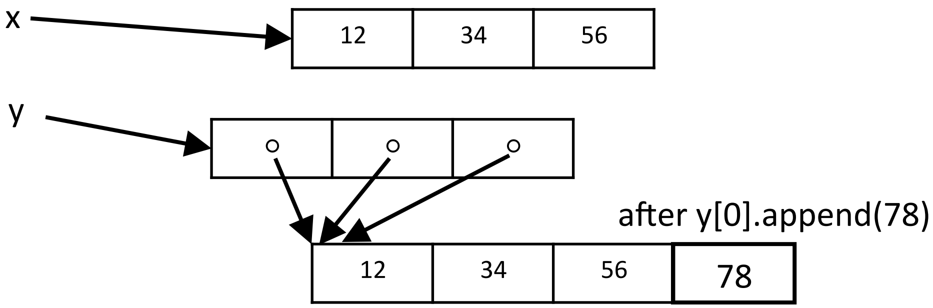


```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = [ [12, 34, 56] ]
>>>
>>> y = x * 3
>>> y
[[12, 34, 56], [12, 34, 56], [12, 34, 56]]
>>>
>>> y[0].append(78)
>>>
>>> y
[[12, 34, 56, 78], [12, 34, 56, 78], [12,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

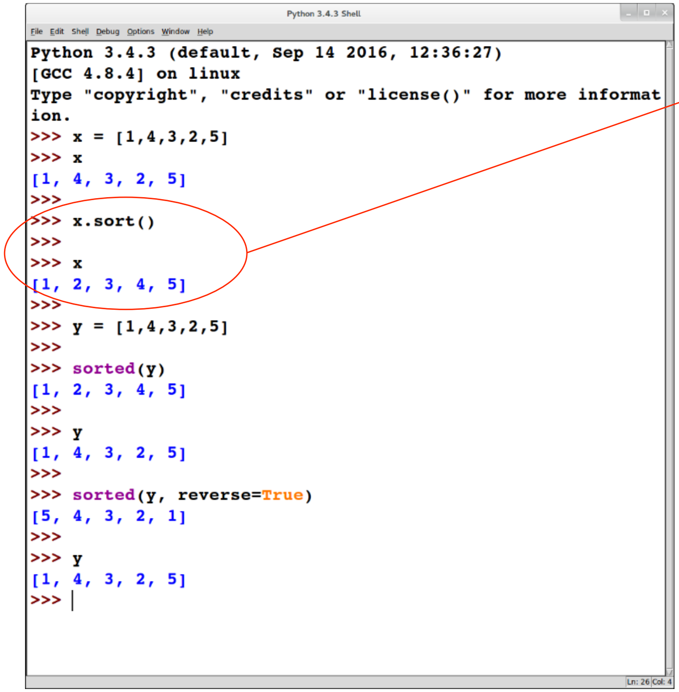
concatenation (+ and *) : similar to strings

these operators create “shallow” copies

- due to list mutability, this can cause unexpected behavior



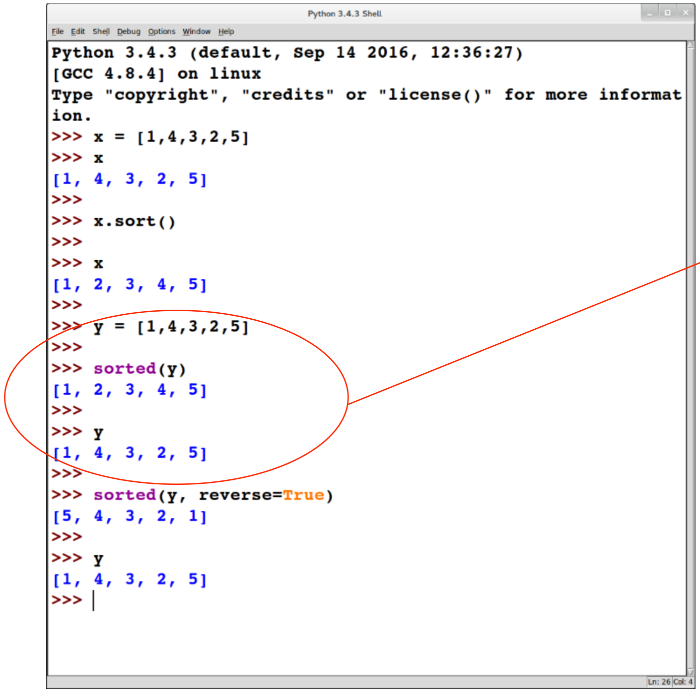
Lists: sorting



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

sort() : sorts a list

Lists: sorting

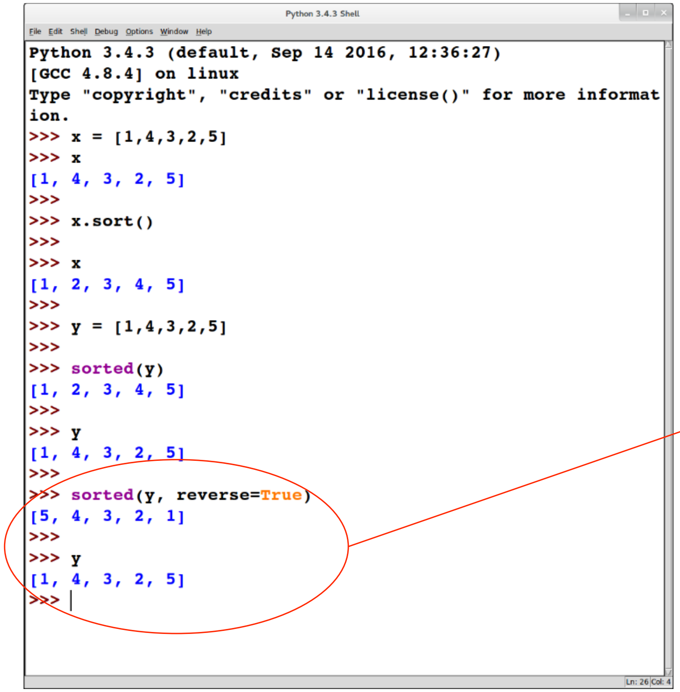


```
Python 3.4.3 (default, Sep 14 2016, 12:36:27)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = [1,4,3,2,5]
>>> x
[1, 4, 3, 2, 5]
>>>
>>> x.sort()
>>>
>>> x
[1, 2, 3, 4, 5]
>>>
>>> y = [1,4,3,2,5]
>>>
>>> sorted(y)
[1, 2, 3, 4, 5]
>>>
>>> y
[1, 4, 3, 2, 5]
>>>
>>> sorted(y, reverse=True)
[5, 4, 3, 2, 1]
>>>
>>> y
[1, 4, 3, 2, 5]
>>> 
```

sort() : sorts a list

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

Lists: sorting



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

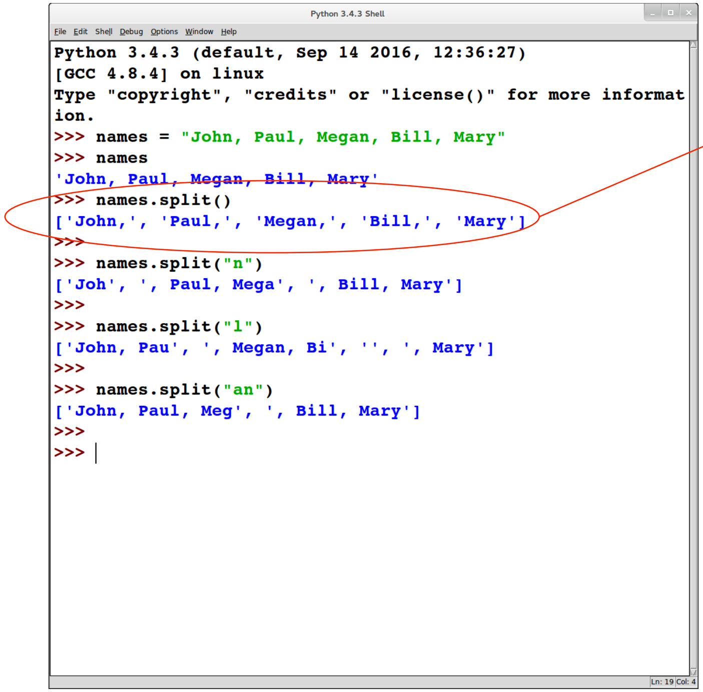
sort() : sorts a list

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

the optional argument reverse specifies ascending or descending order

python review:
lists ↔ strings

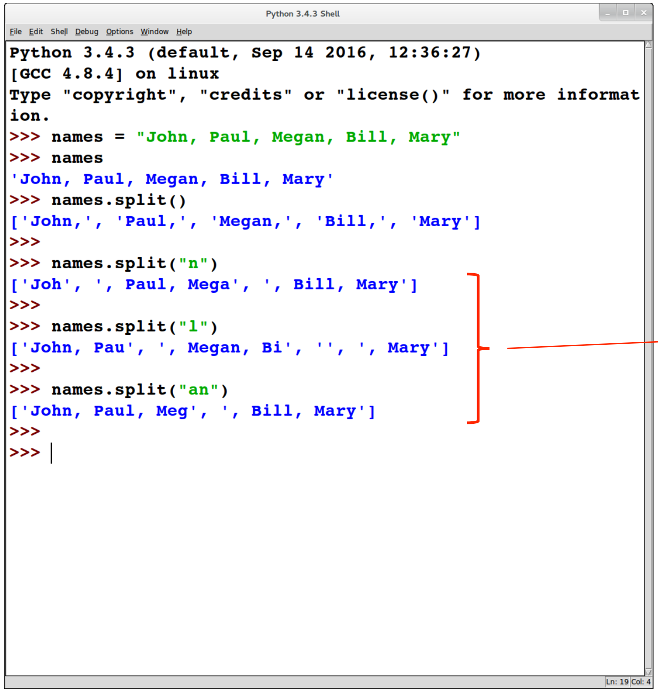
Strings → lists



```
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.
>>> 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("l")
['John, Pau', ', Megan, Bi', '', ', Mary']
>>>
>>> names.split("an")
['John, Paul, Meg', ', Bill, Mary']
>>>
>>> 
```

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

Strings → lists



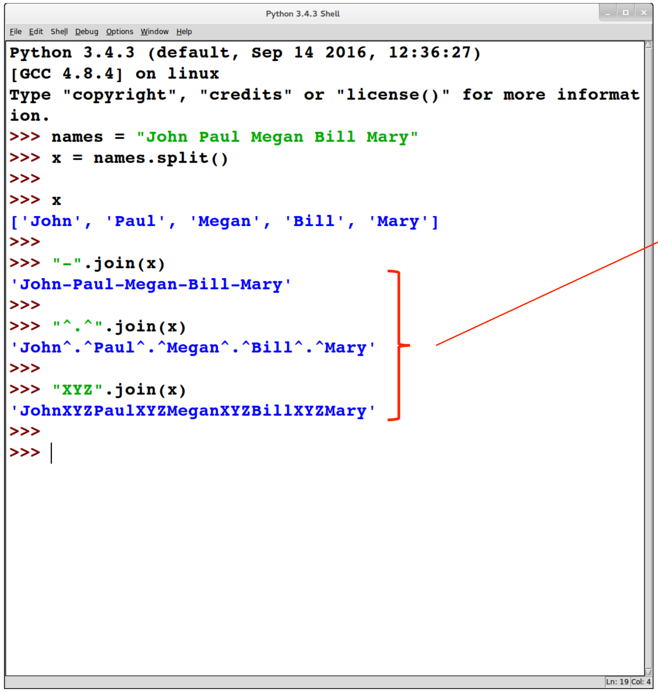
```
Python 3.4.3 (default, Sep 14 2016, 12:36:27)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> names = "John, Paul, Megan, Bill, Mary"
>>> names
'John, Paul, Megan, Bill, Mary'
>>> names.split()
['John,', 'Paul,', 'Megan,', 'Bill,', 'Mary']
>>>
>>> names.split("n")
['Joh', ', Paul, Mega', ', Bill, Mary']
>>>
>>> names.split("l")
['John, Pau', ', Megan, Bi', '', ', Mary']
>>>
>>> names.split("an")
['John, Paul, Meg', ', Bill, Mary']
>>>
>>> |
```

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

split(*delim*) : given an optional argument

delim, splits the string on *delim*

Lists → strings

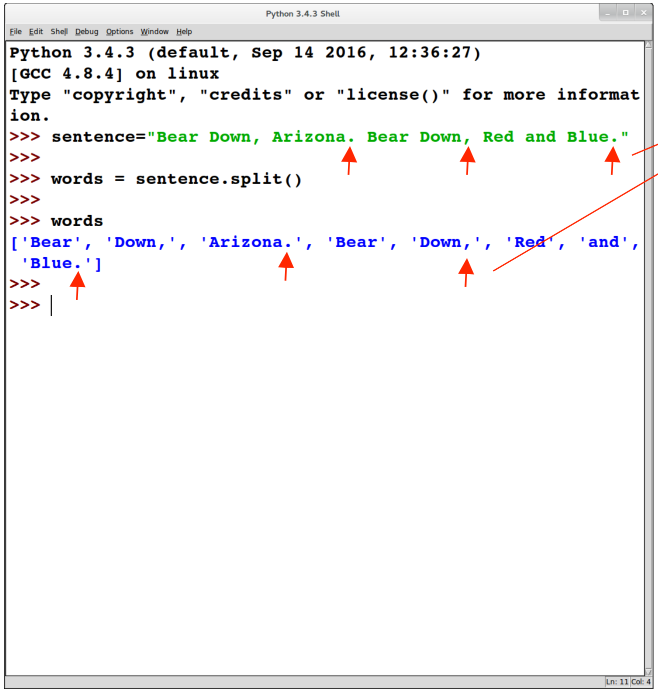


```
Python 3.4.3 (default, Sep 14 2016, 12:36:27)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> names = "John Paul Megan Bill Mary"
>>> x = names.split()
>>>
>>> x
['John', 'Paul', 'Megan', 'Bill', 'Mary']
>>>
>>> "-".join(x)
'John-Paul-Megan-Bill-Mary'
>>>
>>> "^.^".join(x)
'John^.^Paul^.^Megan^.^Bill^.^Mary'
>>>
>>> "XYZ".join(x)
'JohnXYZPaulXYZMeganXYZBillXYZMary'
>>>
>>> 
```

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

returns a string

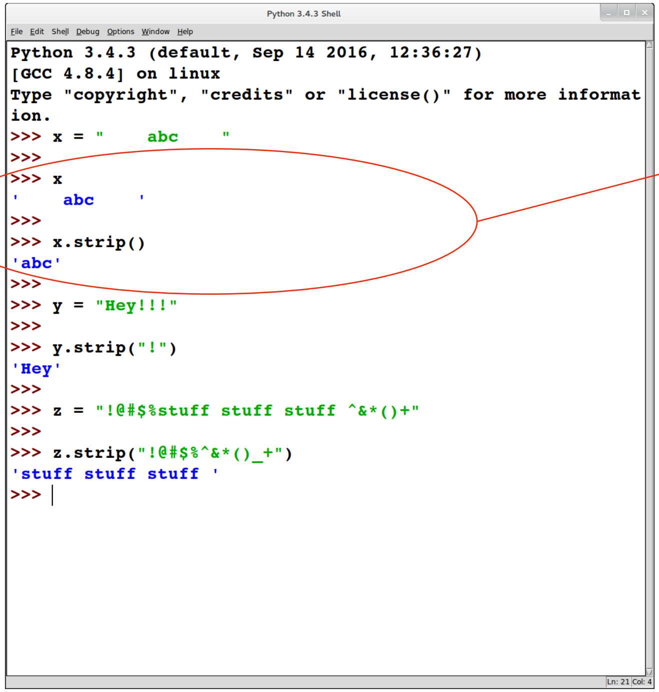
String trimming



```
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.
>>> sentence="Bear Down, Arizona. Bear Down, Red and Blue."
>>>
>>> words = sentence.split()
>>>
>>> words
['Bear', 'Down,', 'Arizona.', 'Bear', 'Down,', 'Red', 'and',
 'Blue.']
>>>
>>> 
```

what if we wanted to get rid of the punctuation?

String trimming

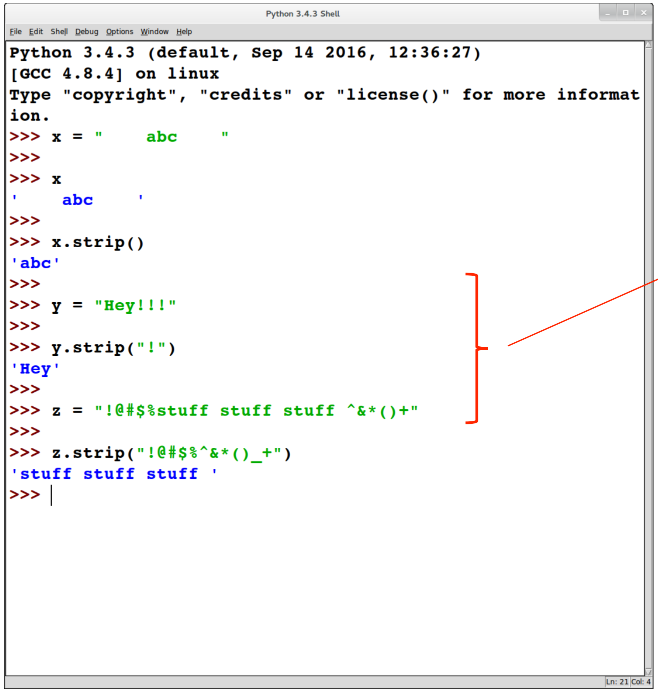


```
Python 3.4.3 (default, Sep 14 2016, 12:36:27)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = "    abc    "
>>>
>>> x
'    abc    '
>>>
>>> x.strip()
'abc'
>>>
>>> y = "Hey!!!"
>>>
>>> y.strip("!")
'Hey'
>>>
>>> z = "!@#$%stuff stuff stuff ^&*()+"
>>>
>>> z.strip("!@#$%^&*()_+")
'stuff stuff stuff '
>>> 
```

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

returns a string

String trimming

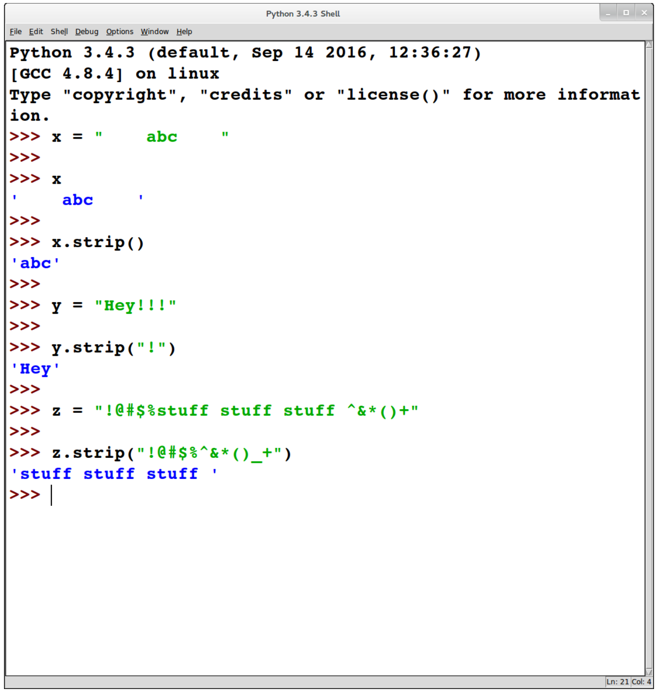


```
Python 3.4.3 (default, Sep 14 2016, 12:36:27)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = "     abc     "
>>>
>>> x
'     abc     '
>>>
>>> x.strip()
'abc'
>>>
>>> y = "Hey!!!"
>>>
>>> y.strip("!")
'Hey'
>>>
>>> z = "!@#$%stuff stuff stuff ^&*()+"
>>>
>>> z.strip("!@#$%^&*()_+")
'stuff stuff stuff '
>>> 
```

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

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

String trimming



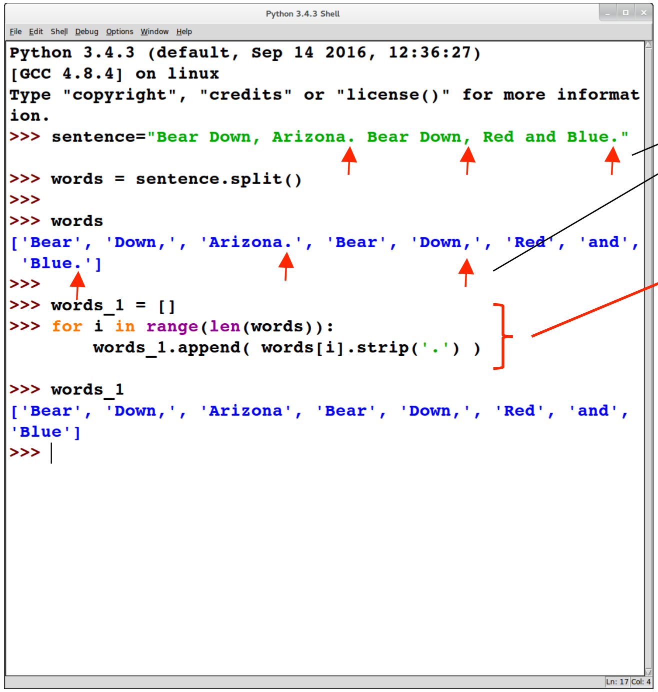
```
Python 3.4.3 (default, Sep 14 2016, 12:36:27)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = "    abc    "
>>> 
>>> x
'    abc    '
>>> 
>>> x.strip()
'abc'
>>> 
>>> y = "Hey!!!"
>>> 
>>> y.strip("!")
'Hey'
>>> 
>>> z = "!@#$%stuff stuff stuff ^&*()+"
>>> 
>>> z.strip("!@#$%^&*()_+")
'stuff stuff stuff '
>>> 
```

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

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

String trimming

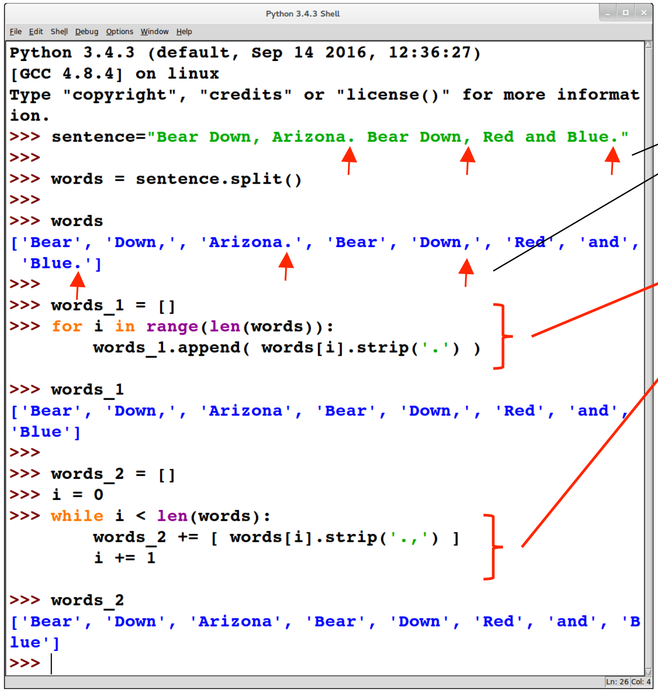


```
Python 3.4.3 (default, Sep 14 2016, 12:36:27)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> sentence="Bear Down, Arizona. Bear Down, Red and Blue."

>>> words = sentence.split()
>>>
>>> words
['Bear', 'Down,', 'Arizona.', 'Bear', 'Down,', 'Red', 'and',
 'Blue.']
>>>
>>> words_1 = []
>>> for i in range(len(words)):
        words_1.append( words[i].strip('.') )

>>> words_1
['Bear', 'Down,', 'Arizona', 'Bear', 'Down,', 'Red', 'and',
'Blue']
>>> 
```

what if we wanted to get rid of the punctuation?

- iterate over the list
- use strip() to trim each word in the list
- reassemble the trimmed words into a list

String trimming



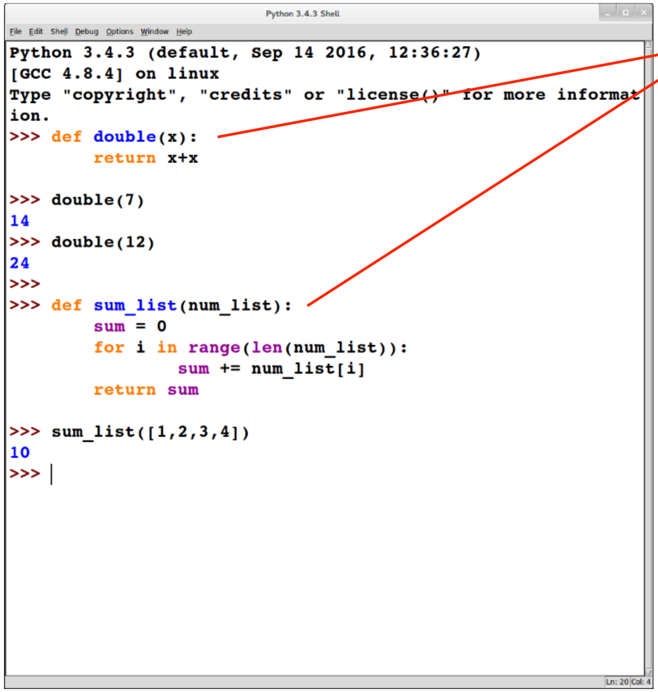
```
Python 3.4.3 (default, Sep 14 2016, 12:36:27)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> sentence="Bear Down, Arizona. Bear Down, Red and Blue."
>>>
>>> words = sentence.split()
>>>
>>> words
['Bear', 'Down,', 'Arizona.', 'Bear', 'Down,', 'Red', 'and',
 'Blue.']
>>>
>>> words_1 = []
>>> for i in range(len(words)):
        words_1.append( words[i].strip('.') )

>>> words_1
['Bear', 'Down,', 'Arizona', 'Bear', 'Down,', 'Red', 'and',
'Blue']
>>>
>>> words_2 = []
>>> i = 0
>>> while i < len(words):
        words_2 += [ words[i].strip('.,') ]
        i += 1

>>> words_2
['Bear', 'Down', 'Arizona', 'Bear', 'Down', 'Red', 'and', 'B
lue']
>>> 
```

what if we wanted to get rid of the punctuation?

- iterate over the list
- use strip() to trim each word in the list
- reassemble the trimmed words into a list

python review: functions

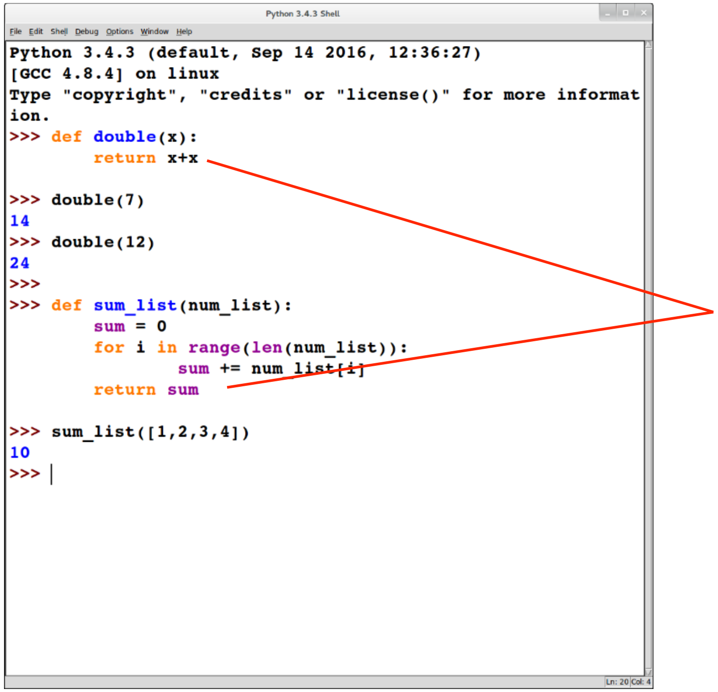
Functions



```
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.
>>> def double(x):
        return x+x

>>> double(7)
14
>>> double(12)
24
>>>
>>> def sum_list(num_list):
        sum = 0
        for i in range(len(num_list)):
                sum += num_list[i]
        return sum

>>> sum_list([1,2,3,4])
10
>>> 
```

- **`def`** *fn_name* (arg_1 , …, arg_n)
  - defines a function *fn_name* with n arguments arg_1 , …, arg_n

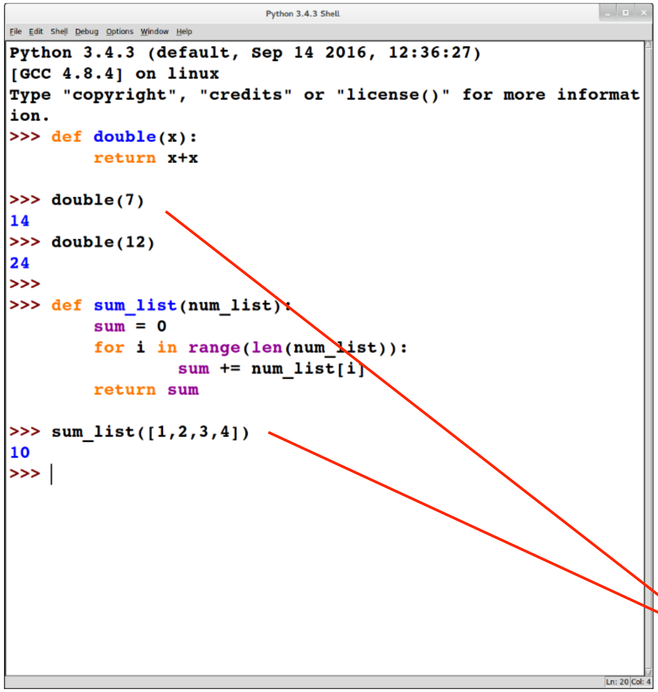
Functions



```
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.
>>> def double(x):
        return x+x

>>> double(7)
14
>>> double(12)
24
>>>
>>> def sum_list(num_list):
        sum = 0
        for i in range(len(num_list)):
                sum += num_list[i]
        return sum

>>> sum_list([1,2,3,4])
10
>>> |
```

- **`def`** *fn_name* (arg_1 , ..., arg_n)
  - defines a function *fn_name* with n arguments arg_1 , ..., arg_n
- **`return`** *expr*
  - optional
  - returns the value of the expression *expr* to the caller

Functions



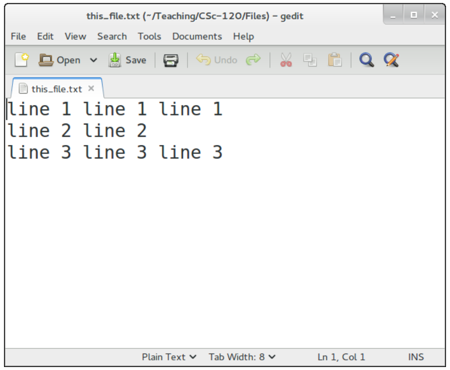
```
Python 3.4.3 (default, Sep 14 2016, 12:36:27)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> def double(x):
        return x+x

>>> double(7)
14
>>> double(12)
24
>>>
>>> def sum_list(num_list):
        sum = 0
        for i in range(len(num_list)):
                sum += num_list[i]
        return sum

>>> sum_list([1,2,3,4])
10
>>> |
```

- **`def`** *fn_name* (arg_1 , …, arg_n)
  - defines a function *fn_name* with n arguments arg_1 , …, arg_n
- **`return`** *expr*
  - optional
  - returns the value of the expression *expr* to the caller
- *fn_name*($expr_1$, …, $expr_n$)
  - calls *fn_name* with arguments expr_1, …, expr_n

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

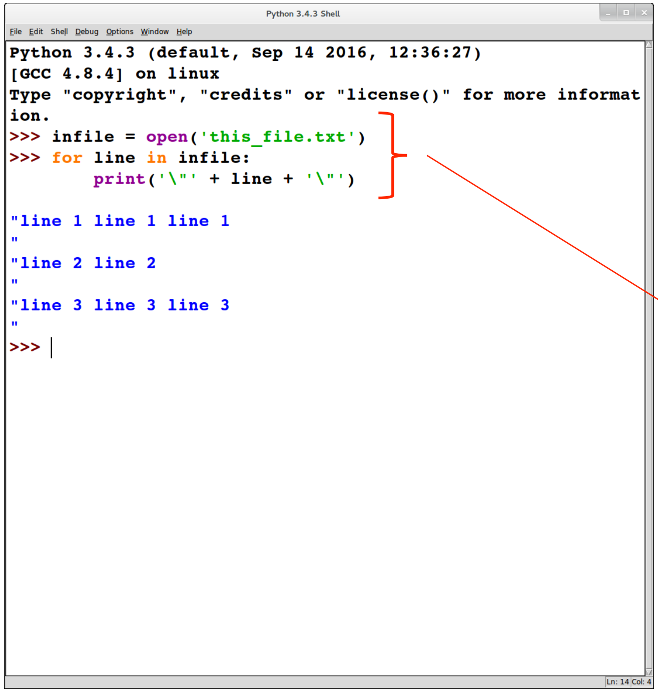
Reading user input II: file I/O

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

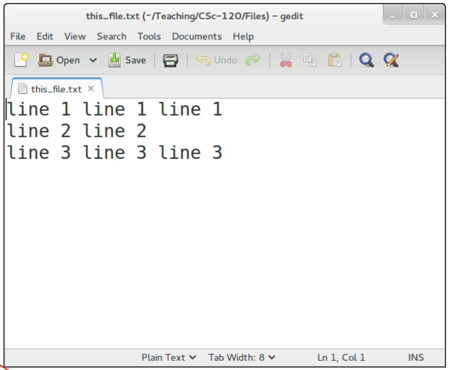

this_file.txt (~/Teaching/CSc-120/Files) - gedit
File Edit View Search Tools Documents Help
Open
Save
Undo
this_file.txt
Plain Text
Tab Width: 8
Ln 1, Col 1
INS

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

Reading user input II: file I/O



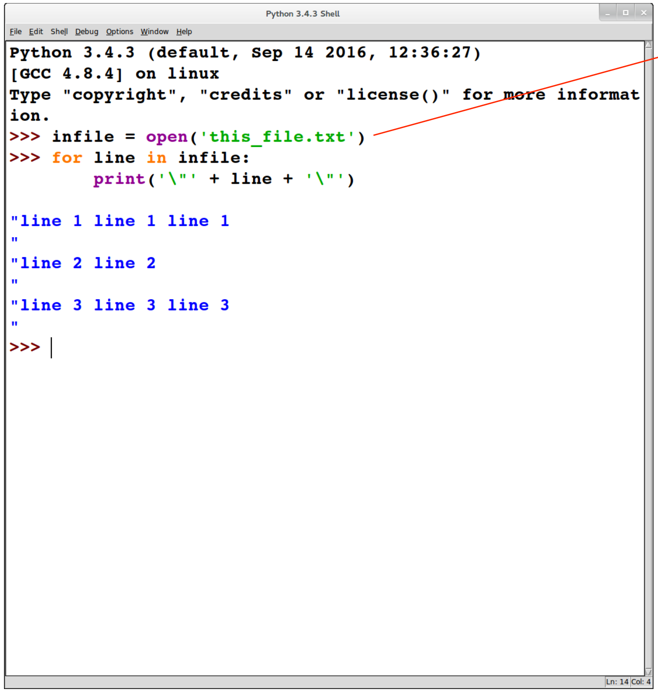
```
Python 3.4.3 (default, Sep 14 2016, 12:36:27)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> infile = open('this_file.txt')
>>> for line in infile:
        print('\"' + line + '\"')

"line 1 line 1 line 1
"
"line 2 line 2
"
"line 3 line 3 line 3
"
>>> 
```


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
```

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

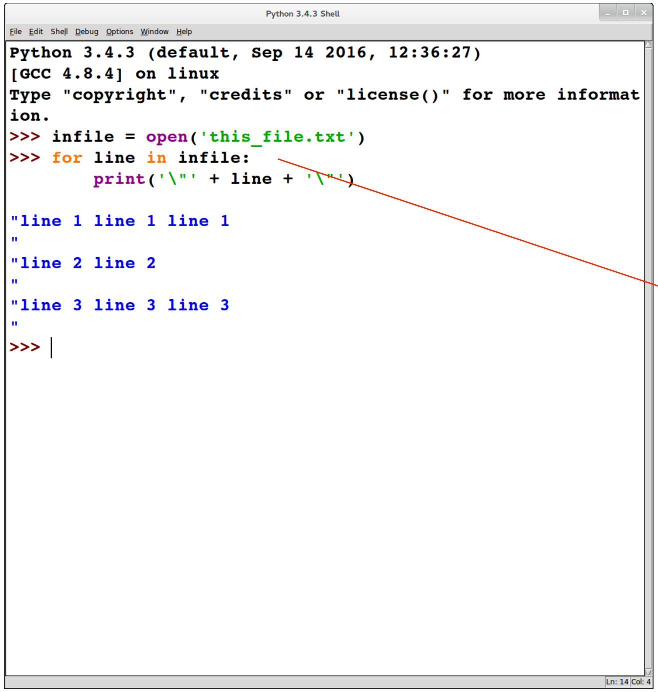
Reading user input II: file I/O



```
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.
>>> 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 3
"
>>> 
```

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

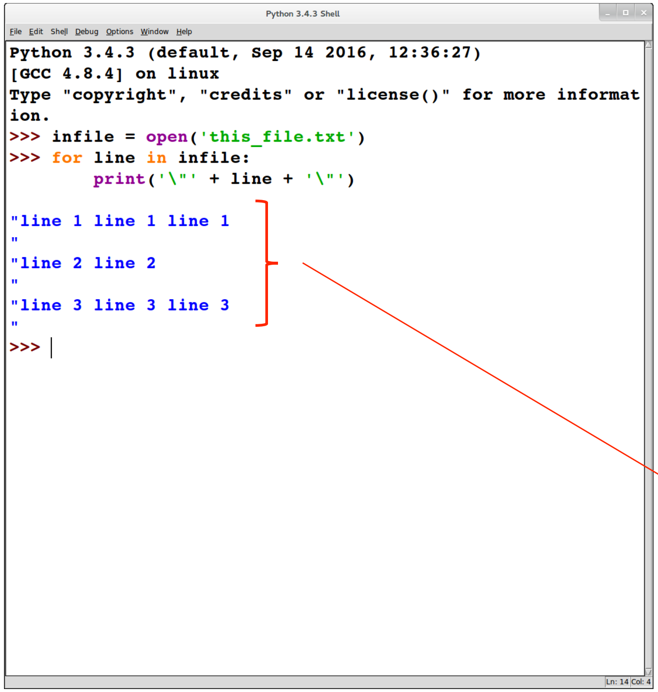
Reading user input II: file I/O



```
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.
>>> 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 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*

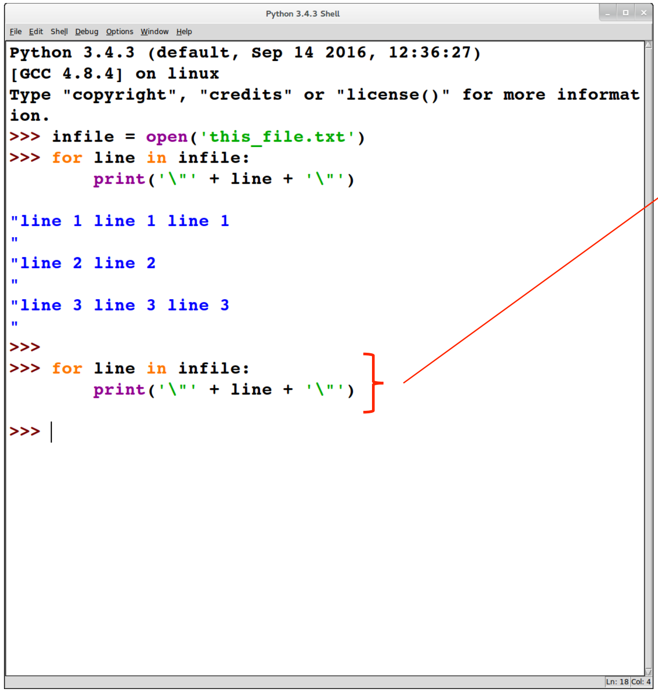
Reading user input II: file I/O



```
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.
>>> 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 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

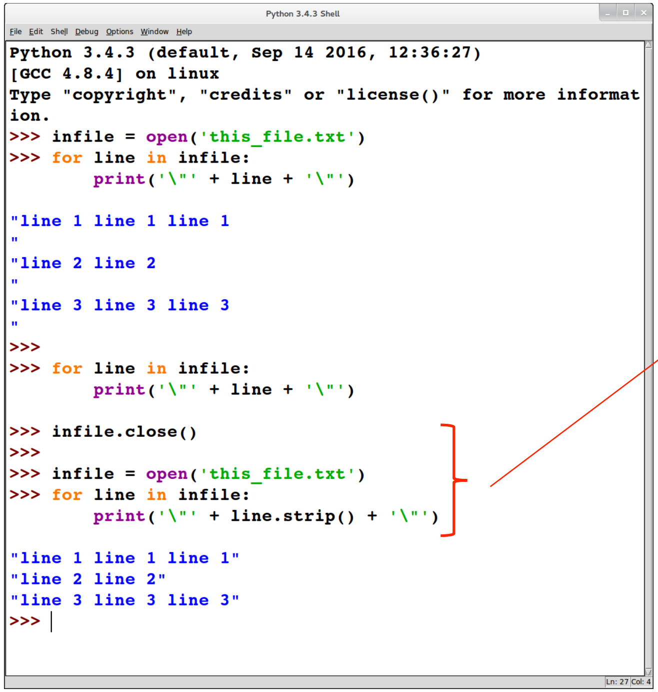
Reading user input II: file I/O



```
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.
>>> 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 3
"
>>>
>>> for line in infile:
        print('\"' + line + '\"')

>>> 
```

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

Reading user input II: file I/O

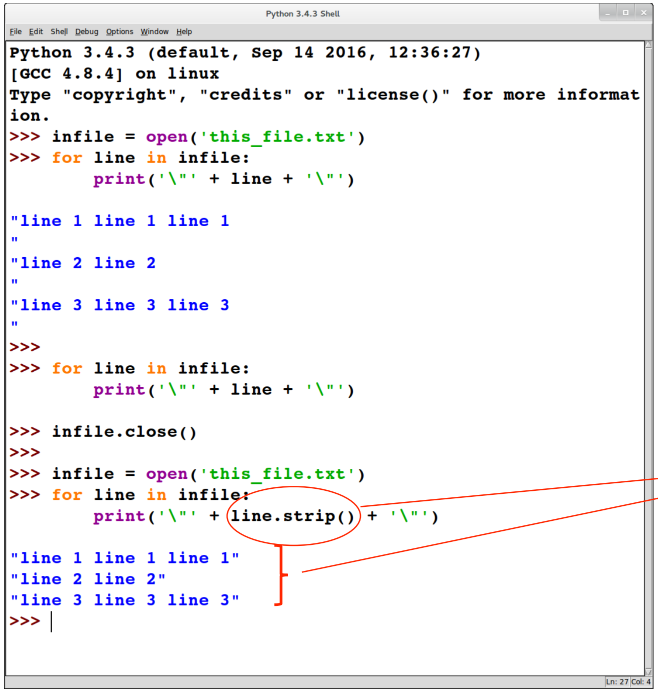


```
Python 3.4.3 (default, Sep 14 2016, 12:36:27)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> infile = open('this_file.txt')
>>> for line in infile:
        print('\"' + line + '\"')

"line 1 line 1 line 1
"
"line 2 line 2
"
"line 3 line 3 line 3
"
>>> 
>>> for line in infile:
        print('\"' + line + '\"')

>>> infile.close()
>>> 
>>> 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 line 3"
>>> 
```

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



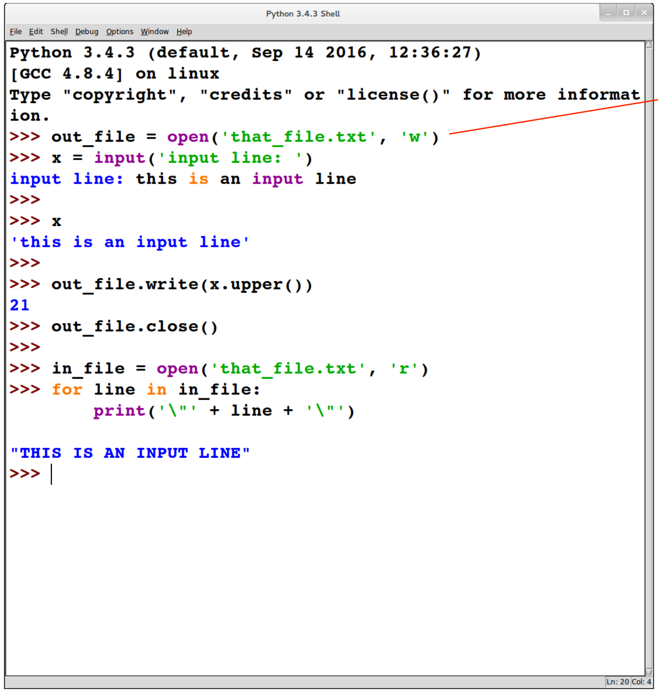
```
Python 3.4.3 (default, Sep 14 2016, 12:36:27)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> infile = open('this_file.txt')
>>> for line in infile:
	print('\"' + line + '\"')

"line 1 line 1 line 1
"
"line 2 line 2
"
"line 3 line 3 line 3
"
>>> 
>>> for line in infile:
	print('\"' + line + '\"')

>>> infile.close()
>>> 
>>> 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 line 3"
>>> |
```

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

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

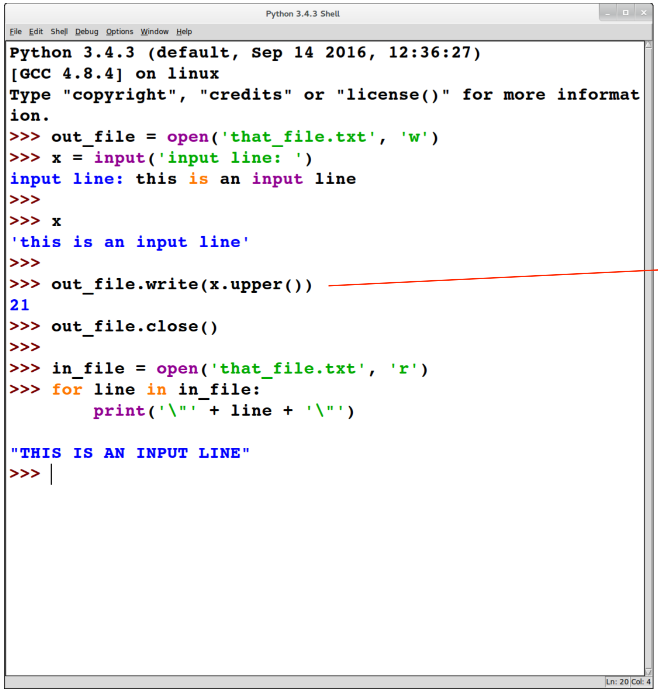
Writing output to a file



```
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

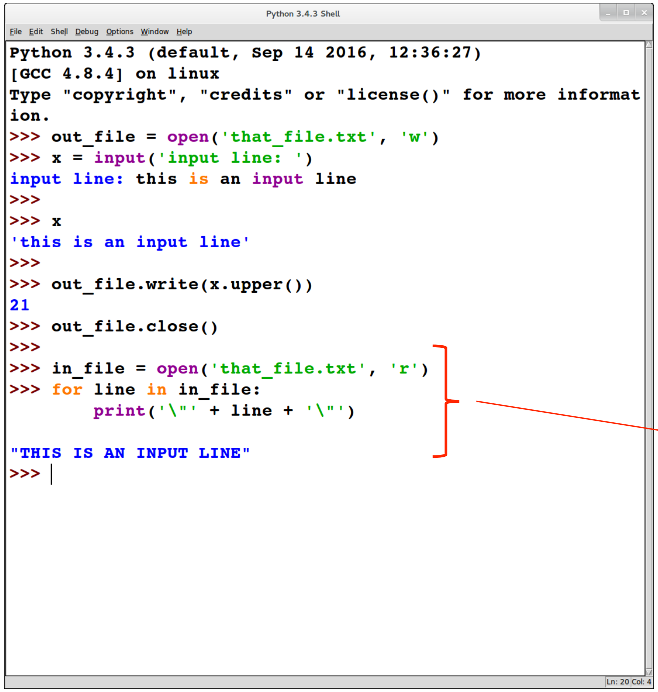


```
Python 3.4.3 (default, Sep 14 2016, 12:36:27)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> out_file = open('that_file.txt', 'w')
>>> x = input('input line: ')
input line: this is an input line
>>>
>>> x
'this is an input line'
>>>
>>> out_file.write(x.upper())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

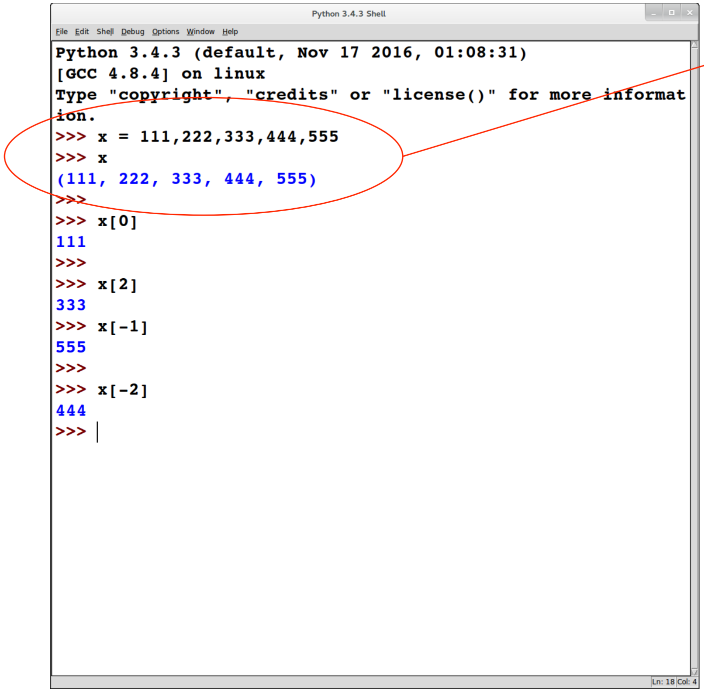
open(*filename*, **"w")** : opens *filename* in write mode, i.e., for output

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

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

python review: tuples

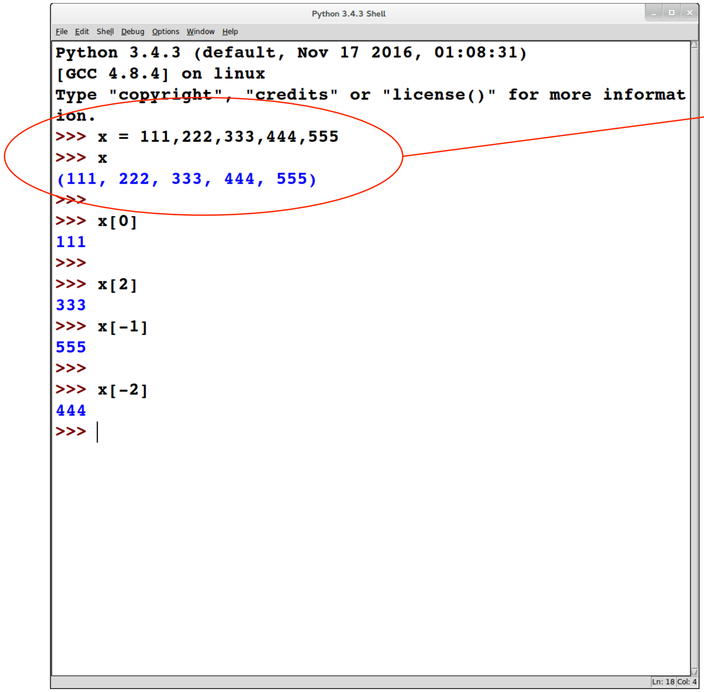
Tuples



```
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
>>> 
```

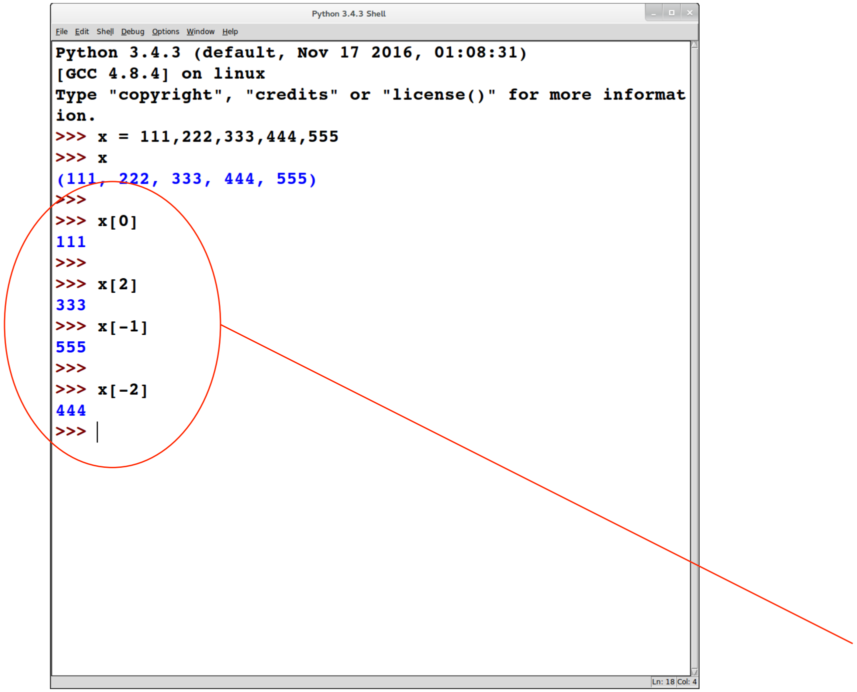
a tuple is a sequence of values (like lists)

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)

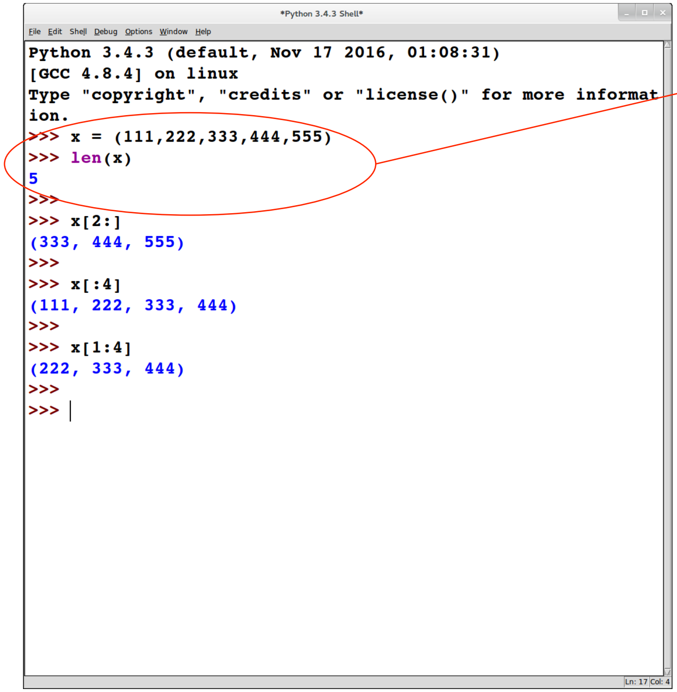
tuples use parens ()

- 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

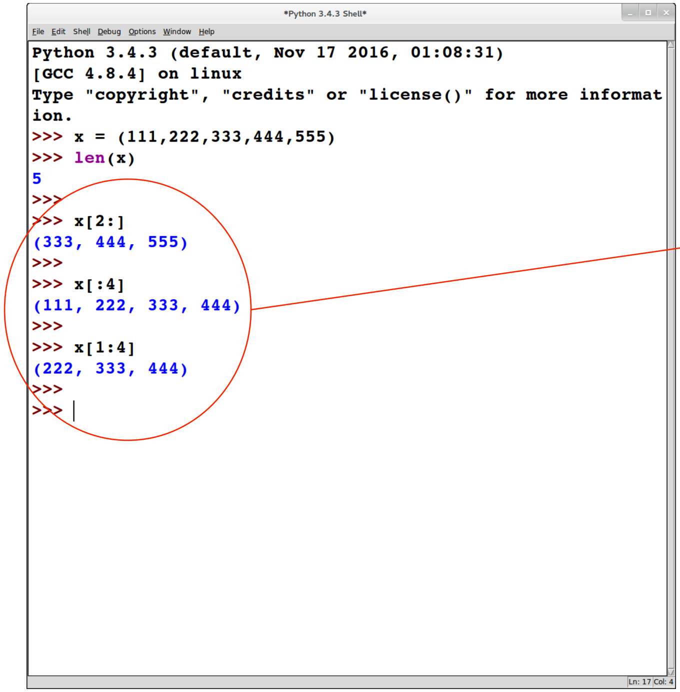
Tuples



```
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

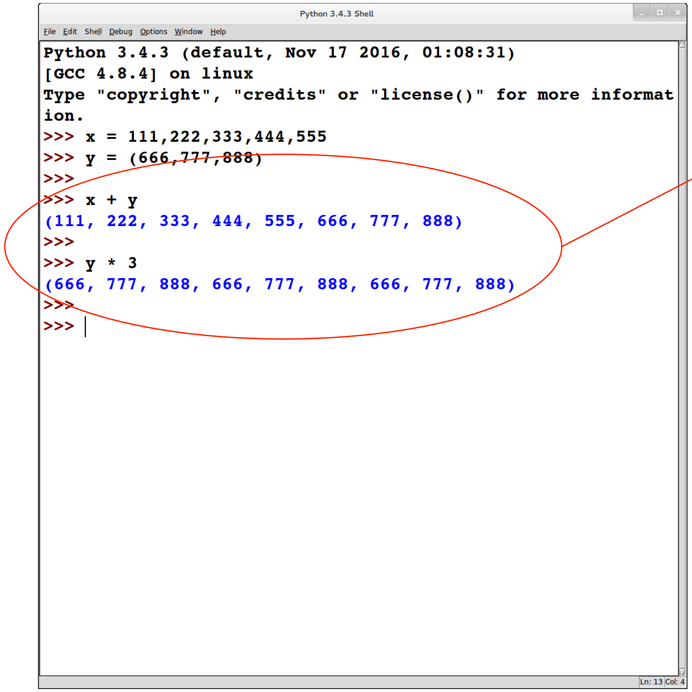


```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = (111,222,333,444,555)
>>> len(x)
5
>>>
>>> x[2:]
(333, 444, 555)
>>>
>>> x[:4]
(111, 222, 333, 444)
>>>
>>> x[1:4]
(222, 333, 444)
>>>
>>> 
```

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

computing slices of a tuple: similar to strings and lists

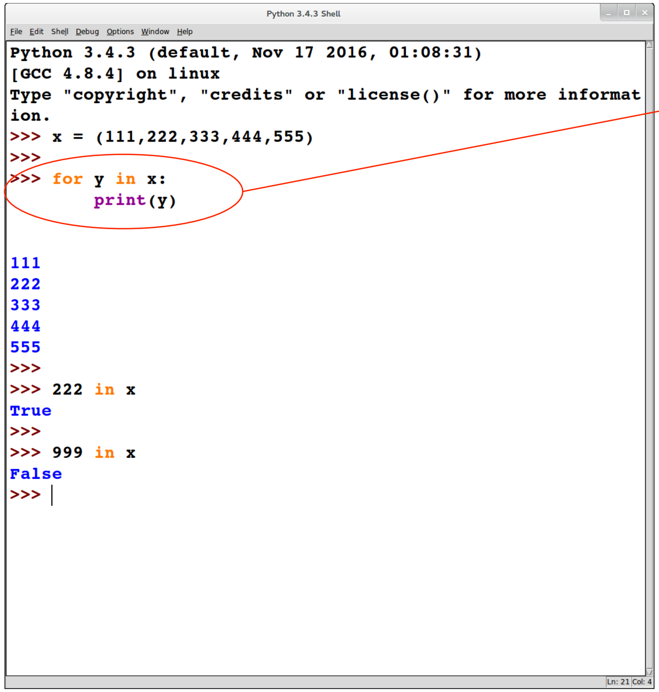
Tuples



```
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

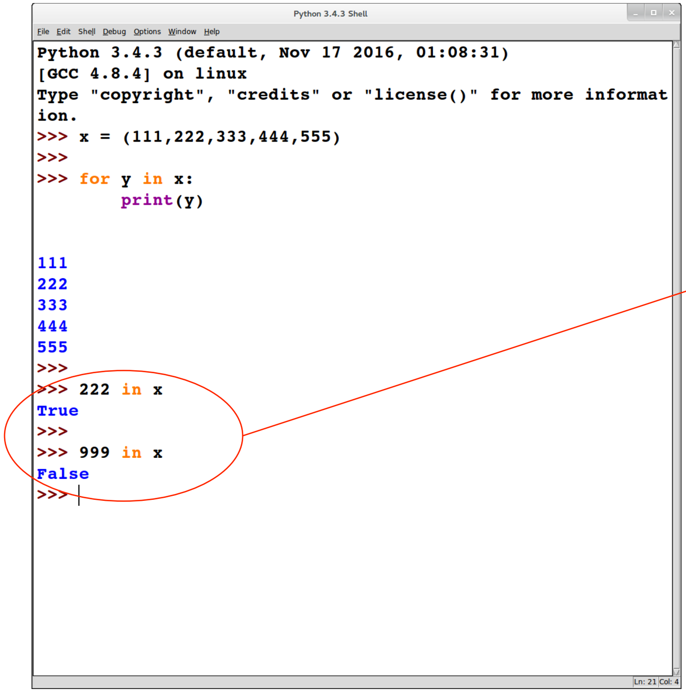
Tuples



```
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

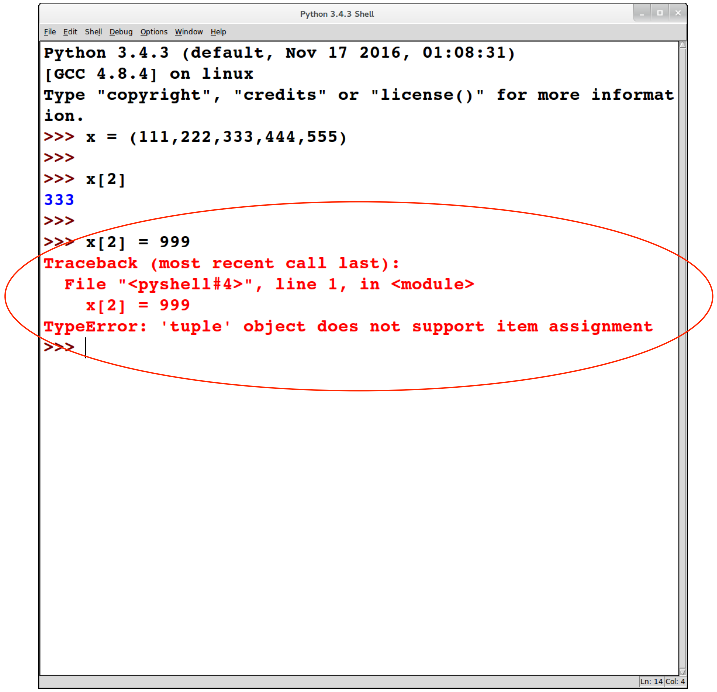


```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = (111,222,333,444,555)
>>>
>>> for y in x:
        print(y)


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

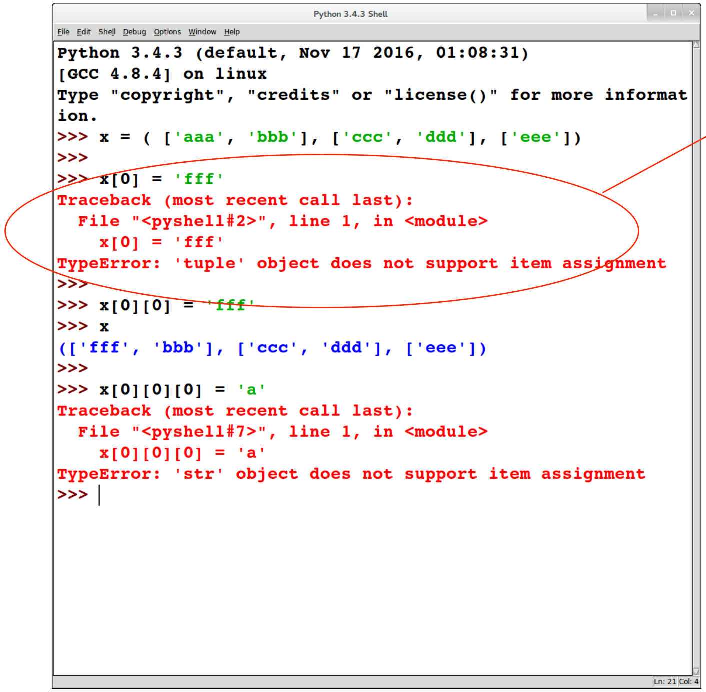
Tuples



```
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

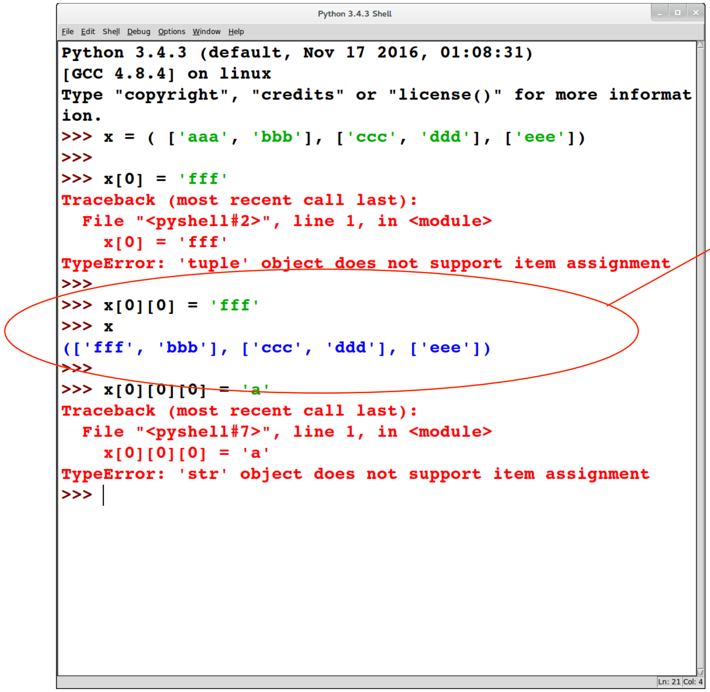
Sequence types: mutability



```
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

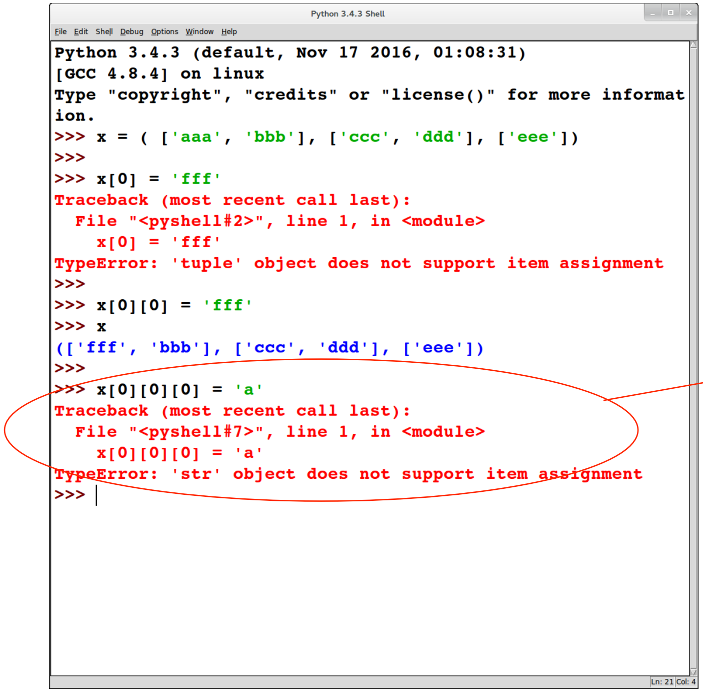


```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = ( ['aaa', 'bbb'], ['ccc', 'ddd'], ['eee'])
>>>
>>> x[0] = 'fff'
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    x[0] = 'fff'
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



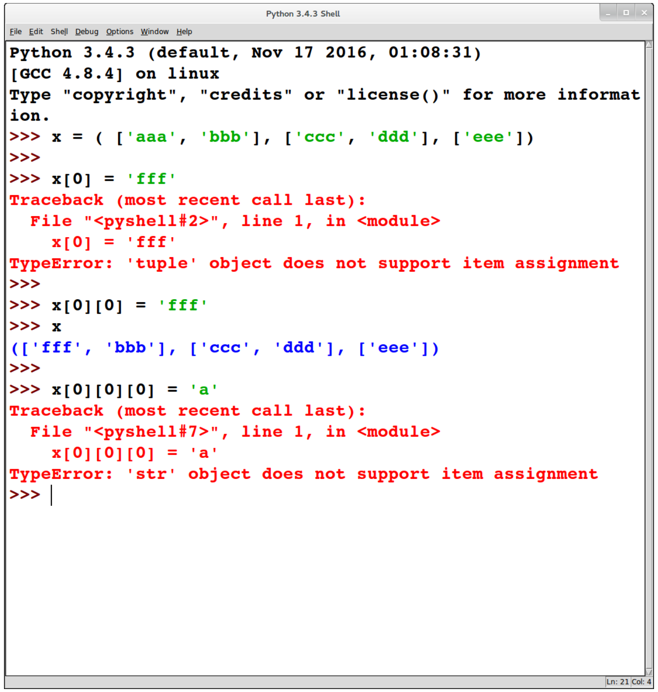
```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x = ( ['aaa', 'bbb'], ['ccc', 'ddd'], ['eee'])
>>>
>>> x[0] = 'fff'
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    x[0] = 'fff'
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

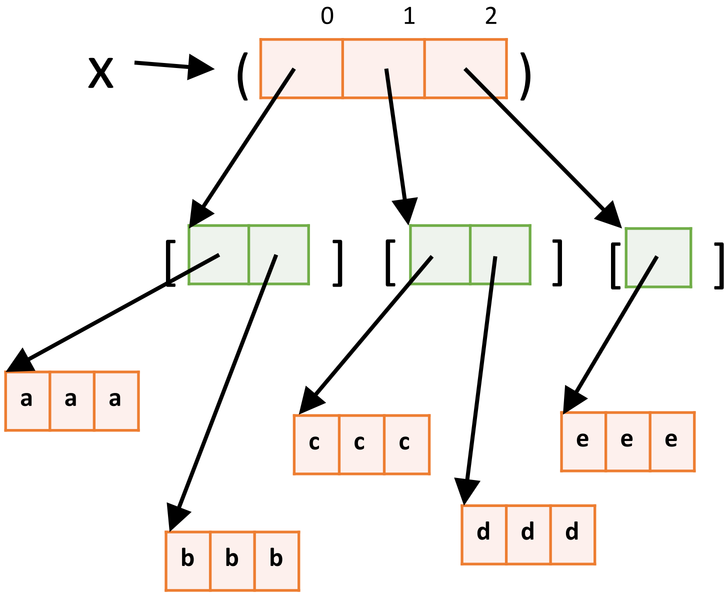
lists are mutable (even if the list is an element of a [immutable] tuple)

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
>>> |
```

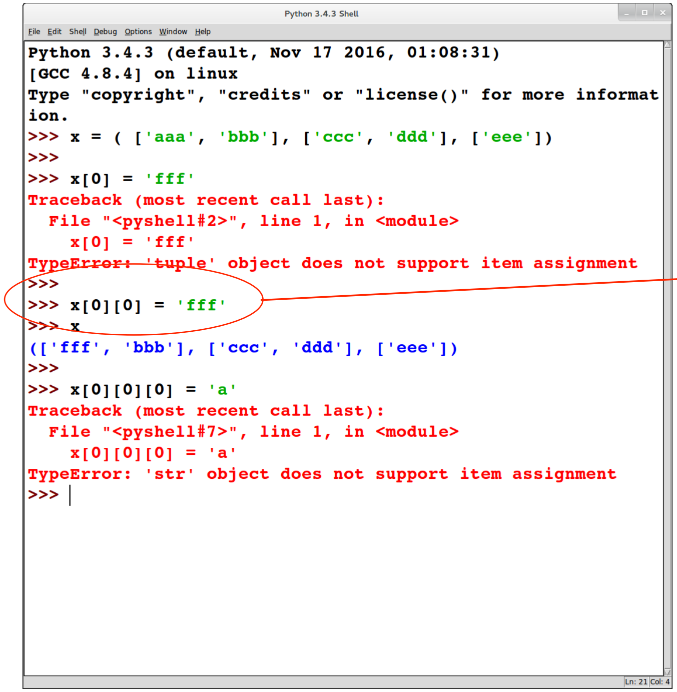


tuple
(immutable)

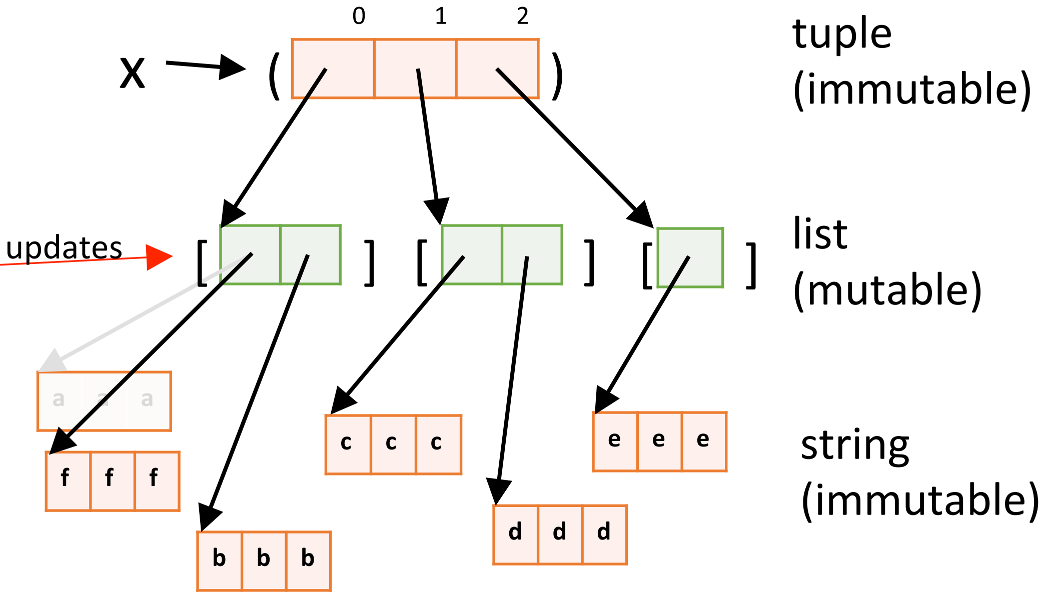
list
(mutable)

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?

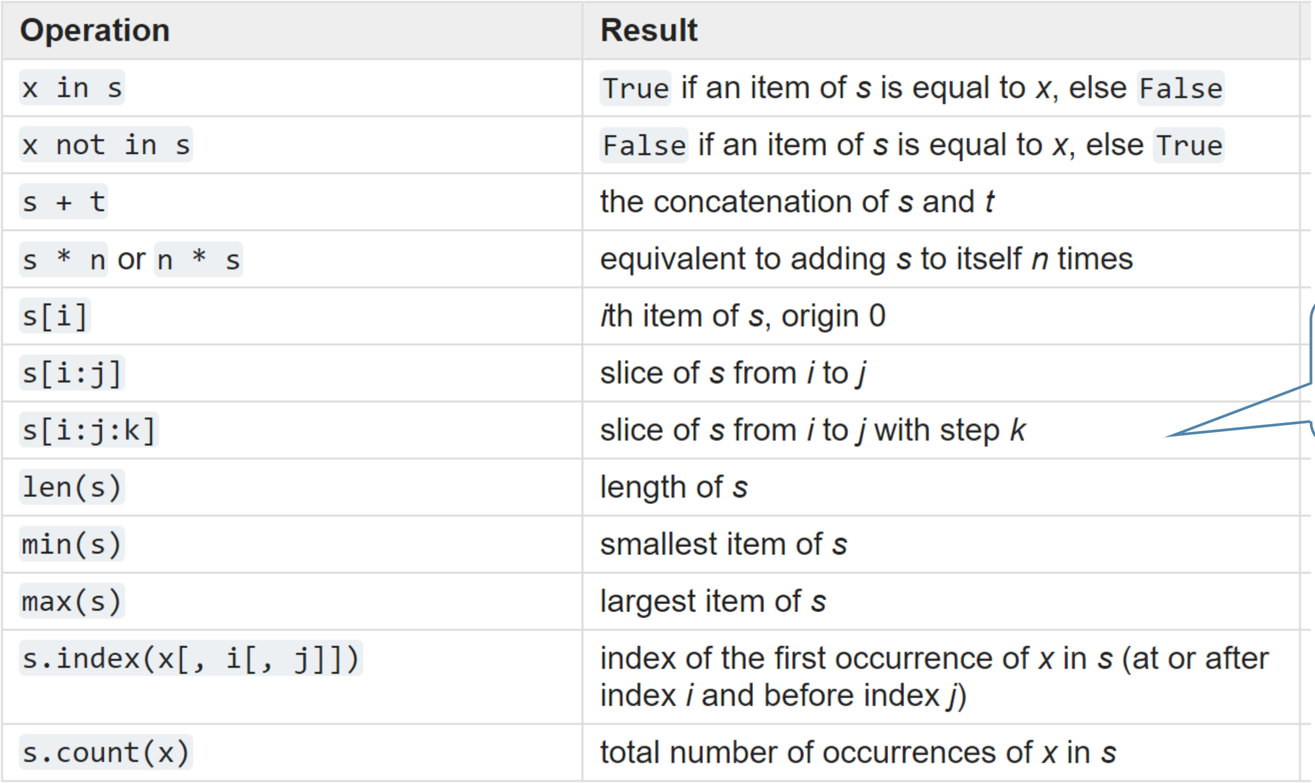
At the implementation level, tuples are much simpler than lists:

- 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 ]
```

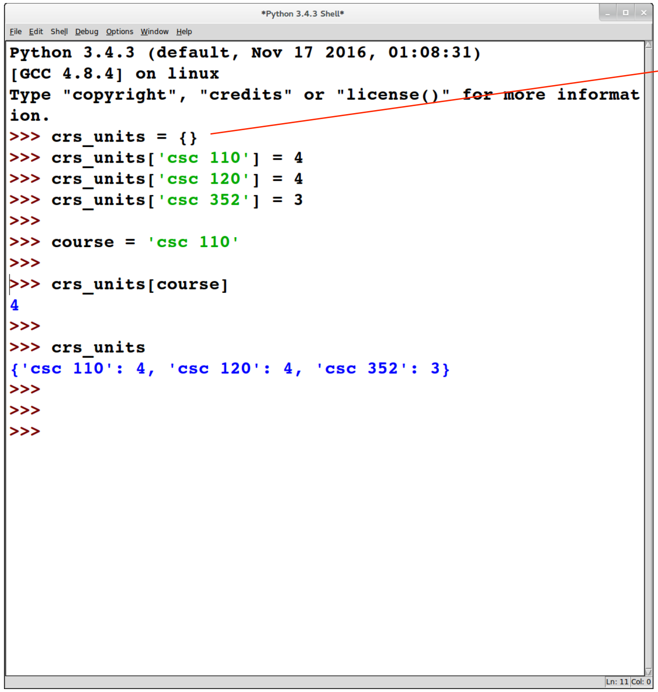
what do you think will be printed out?

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

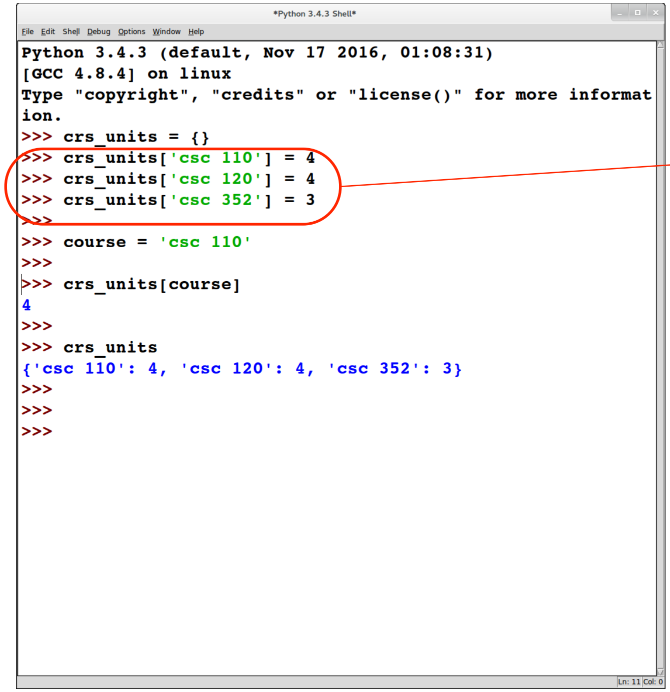
Dictionaries



```
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



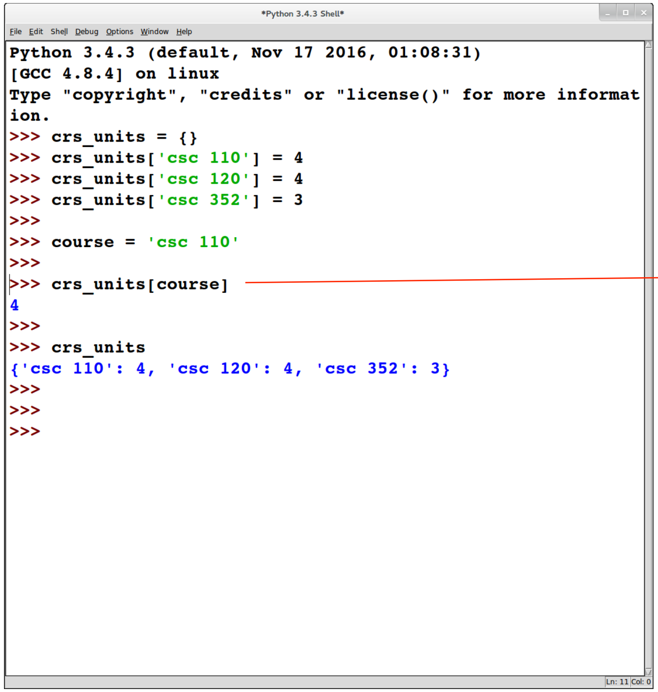
```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> crs_units = {}
>>> crs_units['csc 110'] = 4
>>> crs_units['csc 120'] = 4
>>> crs_units['csc 352'] = 3
>>>
>>> course = 'csc 110'
>>>
>>> crs_units[course]
4
>>>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

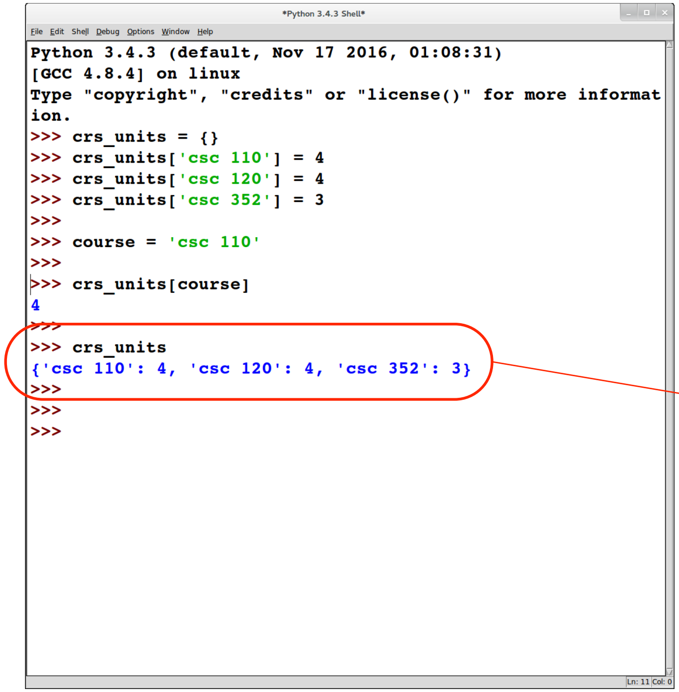
empty dictionary

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

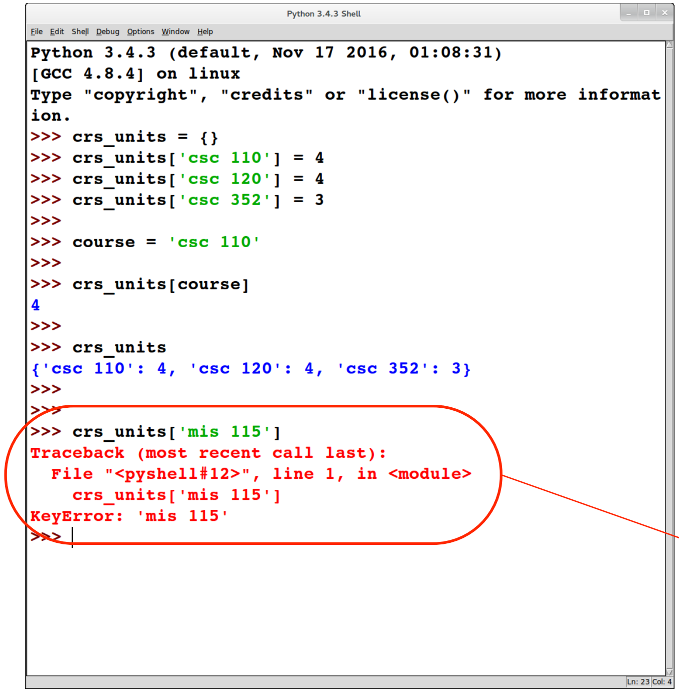
- in this example, one item at a time

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

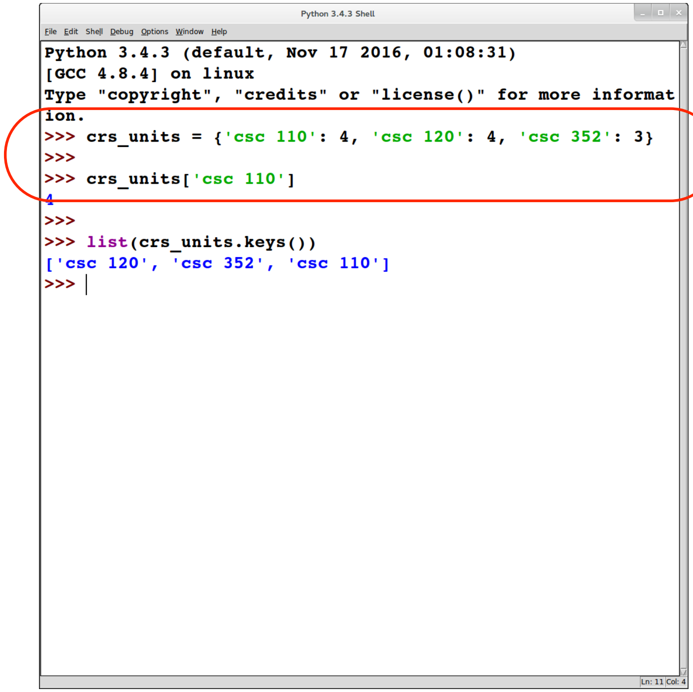
looking up the dictionary (indexing)

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

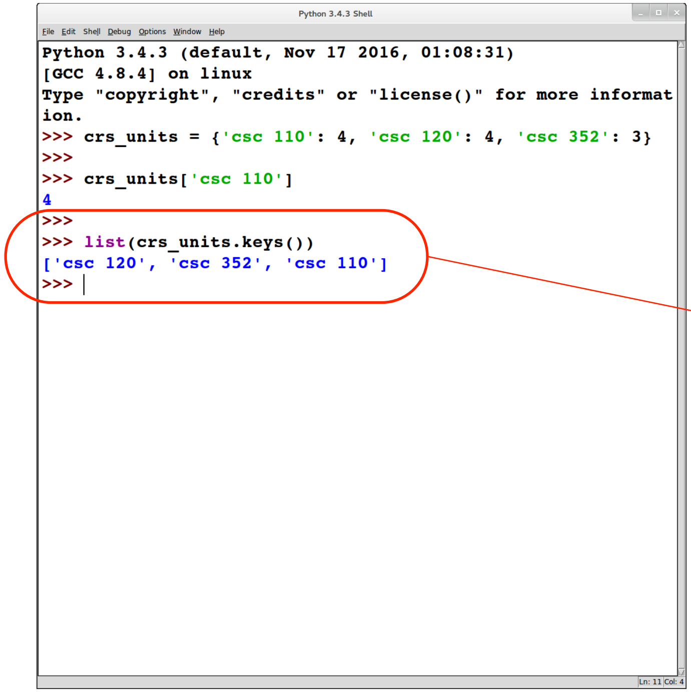


```
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more informat
ion.
>>> crs_units = {'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
>>>
>>> crs_units['csc 110']
4
>>>
>>> list(crs_units.keys())
['csc 120', 'csc 352', 'csc 110']
>>> 
```

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']
>>> 
```

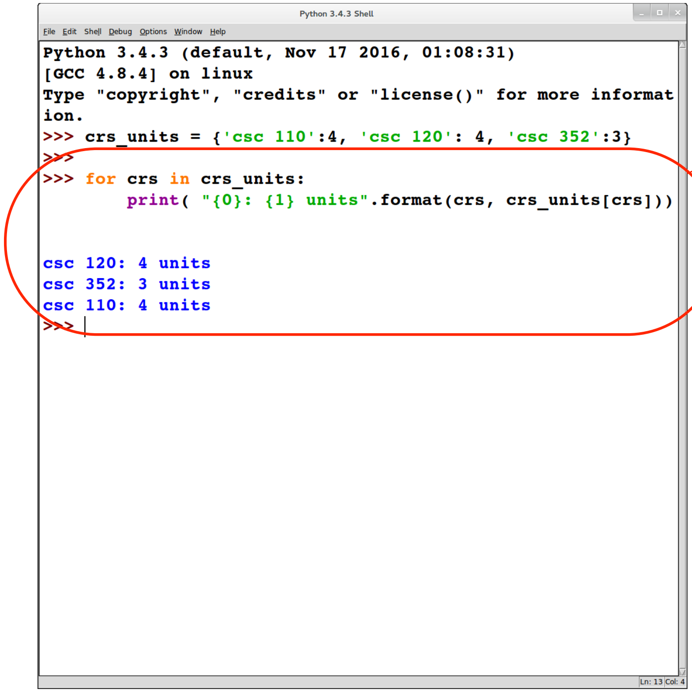
initializing the dictionary

- 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

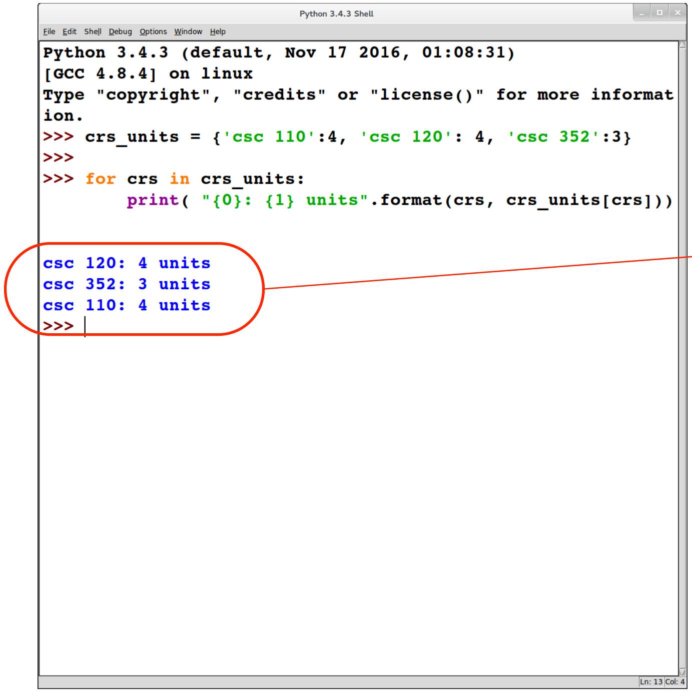
Dictionaries



```
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?)