STSCI 4060

Python Programming and Its Applications in Statistics

Install Python 2.7.15

- Go to https://www.python.org/downloads/release/python-2715.
- Download Python 2.7.15, which is the right version that fits your computer's hardware and OS. (Python 3.x is not recommended for this class due to many reasons.)
- The Windows OS (10 Professional) is recommended. You download the Windows X86-64 MSI Installer (for 64-bit systems).
- Install Python as directed by the installer (In Windows, normally C:\Python27\ by default. If you install elsewhere, make sure that there is no space between words of your directory)
- In Windows, verify Python installation by typing "python" (or "python –V") on a command line (type cmd in the search box and the command line interface will open).
- For Mac OS, download Mac OS X 64-bit installer or Mac OS X 64-bit/32-bit installer and install it based on your Version of Mac OS. Verify Python installation by typing "python" (or "python –V") on the Unix terminal.

Why Python?

- It is a general-purpose, high-level computer programming language with a huge and comprehensive standard library. Also, there a large number of field-specific Python packages for data science related tasks.
- It supports multiple programming paradigms, including object-oriented, imperative and functional, procedural and reflective programming styles.
- It is relatively fast and easy to learn.
- It is often used as a scripting language, but is also used in a wide range of non-scripting contexts.
- It is used for rapid prototyping of complex computing tasks.
- It's great for “gluing together” other programs and tasks into a custom workflow.
- It is portable (it runs on most computing platforms).
- It can be a time-saver for repetitive tasks.
- It is Free!

How to start/run your python program?

- Use the command line interface by typing "python" to run Python interactively.
- Run a Python program by providing a full path of your program, e.g., c:\mypython\myanalysis.py.
- Use the Python **IDLE** window (IDLE stands for integrated development environment, Python's GUI). To start, in Windows, run *C:\Python**27**\lib\idlelib\idle.bat (a Windows batch file identified with the file type), or create a shortcut on desktop and run from the desktop, idle-shortcut.* You are probably first brought to the **Python shell**. The term "shell" refers to the outermost layer of an operating system or a piece of software that communicates interactively with the user. Originally, the "shell" was the part of UNIX or DOS that accepted **commands** typed by the user. It is a place to run single Python statements (or blocks of Python code that you do not want to save as a file or Python module) and get the result **interactively**.
- Open your Python program from the Python shell, and you will see your Python program in IDLE (the edit window), where you can code, edit and run your programs.

Get your feet wet by using Python as an advanced calculator

- At the python prompt (>>>) type each line and hit Enter:
  - >>> 2+2
  - >>> 15/3
  - >>> 5*8
  - >>> a=10
  - >>> b=5
  - >>> a
  - >>> b
  - >>> a+b
  - >>> a/b
  - >>> c , d = 10, 15
  - >>> c
  - >>> d
  - >>> c+d
  - >>> 'Xiaolong ' + 'Yang’

- Use some math functions at the prompt
  - Use the built-in functions
    - >>> abs(-6/3)
    - >>> x=1000, 44, 456, 36,123
    - >>> sorted(x)
  - Use non built-in functions: At >>> you type “import math” first. Try the following.
    - >>> math.log(12)
    - >>> math.pow(2,3)
    - >>> math.sqrt(2)
    - >>> math.pi
    - >>> math.e
    - >>> math.sin(math.pi/2)
    - >>> math.hypot(3,4)

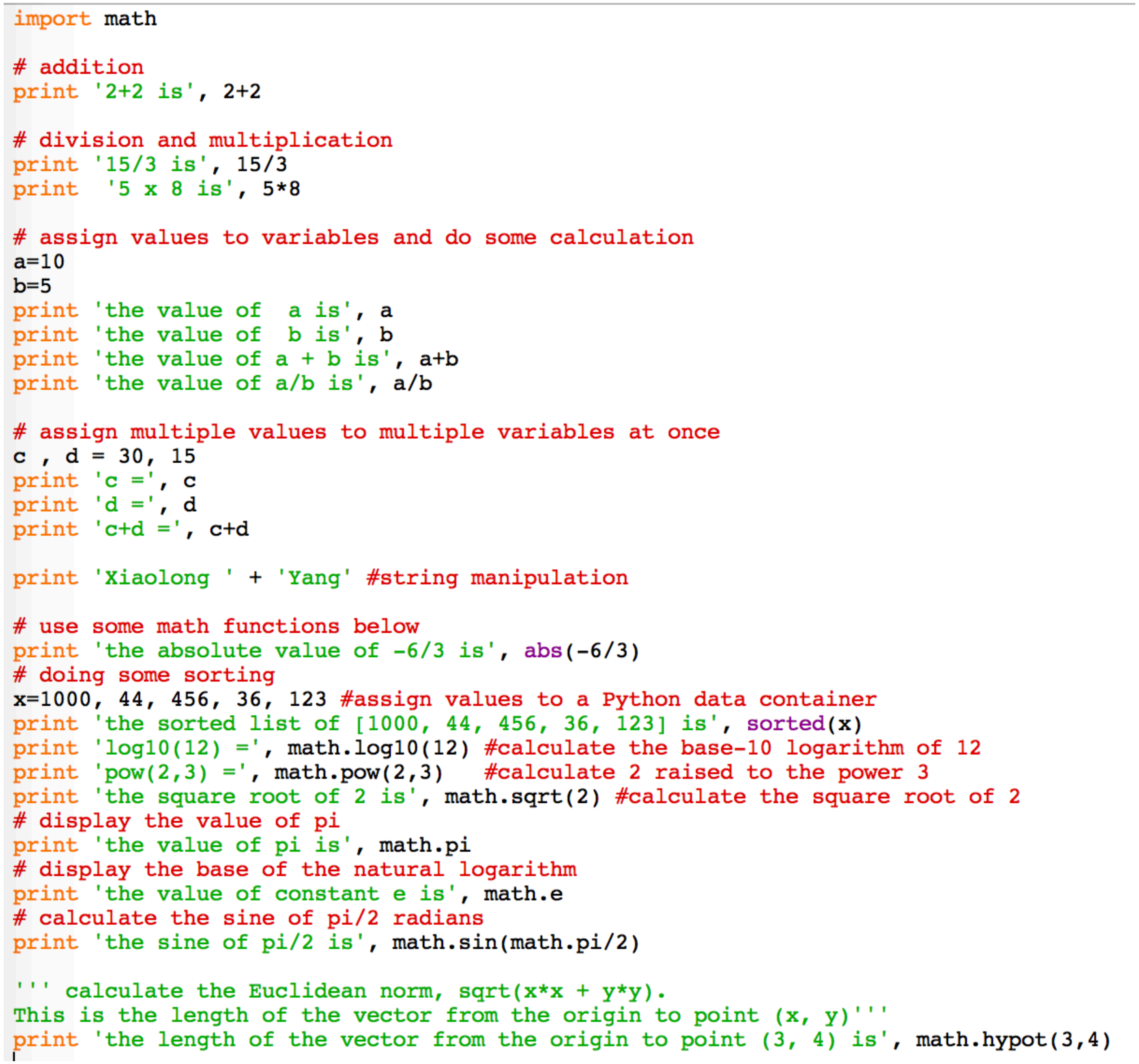
Experience your first Python program

- Python is not just a calculator; you can do all the calculator operations you just did with a Python program, and much more!
- We will be doing many hands-on programming in the class.
- I have put all the calculator steps in a Python file, named calculator_to_module.py (use the "py" extension to show this is a Python file and for color coding in IDLE).
- After you have written and saved your Python program, you run it in IDLE by clicking "Run Module" under the Run menu or pressing the F5 key (in Mac, use fn+F5).

Some Lexical Matters of a Python Program

- Python is **case sensitive**.
- Python uses **indentations** rather than parentheses to mark the beginning and end of a code block (e.g., a function). The convention is to use four spaces (and no hard tabs) for each level of indentation (actually, it's practically a requirement, which will help you merge code from others).
- You do not need a semicolon (;) at the end of a statement, but you do use a semicolon to separate statements only when there is more than one statement on a line (which is not common).
- Multi-line statements are denoted by a back-slash character (\) at the end of a physical line; however, statements contained within the [], {}, or () do not need to use the line continuation character.
- Python uses the hash sign (#) for single line comments and triple quotes (''') for multi-line comments.

The calculator_to_molule program



```
import math

# addition
print '2+2 is', 2+2

# division and multiplication
print '15/3 is', 15/3
print  '5 x 8 is', 5*8

# assign values to variables and do some calculation
a=10
b=5
print 'the value of  a is', a
print 'the value of  b is', b
print 'the value of a + b is', a+b
print 'the value of a/b is', a/b

# assign multiple values to multiple variables at once
c , d = 30, 15
print 'c =', c
print 'd =', d
print 'c+d =', c+d

print 'Xiaolong ' + 'Yang' #string manipulation

# use some math functions below
print 'the absolute value of -6/3 is', abs(-6/3)
# doing some sorting
x=1000, 44, 456, 36, 123 #assign values to a Python data container
print 'the sorted list of [1000, 44, 456, 36, 123] is', sorted(x)
print 'log10(12) =', math.log10(12) #calculate the base-10 logarithm of 12
print 'pow(2,3) =', math.pow(2,3)   #calculate 2 raised to the power 3
print 'the square root of 2 is', math.sqrt(2) #calculate the square root of 2
# display the value of pi
print 'the value of pi is', math.pi
# display the base of the natural logarithm
print 'the value of constant e is', math.e
# calculate the sine of pi/2 radians
print 'the sine of pi/2 is', math.sin(math.pi/2)

''' calculate the Euclidean norm, sqrt(x*x + y*y).
This is the length of the vector from the origin to point (x, y)'''
print 'the length of the vector from the origin to point (3, 4) is', math.hypot(3,4)
```

Code breakdown, part 1

```
import math

# addition
print '2+2 is', 2+2

# division and multiplication
print '15/3 is', 15/3
print  '5 x 8 is', 5*8

# assign values to variables and do some calculation
a=10
b=5
print 'the value of  a is', a
print 'the value of  b is', b
print 'the value of a + b is', a+b
print 'the value of a/b is', a/b
```

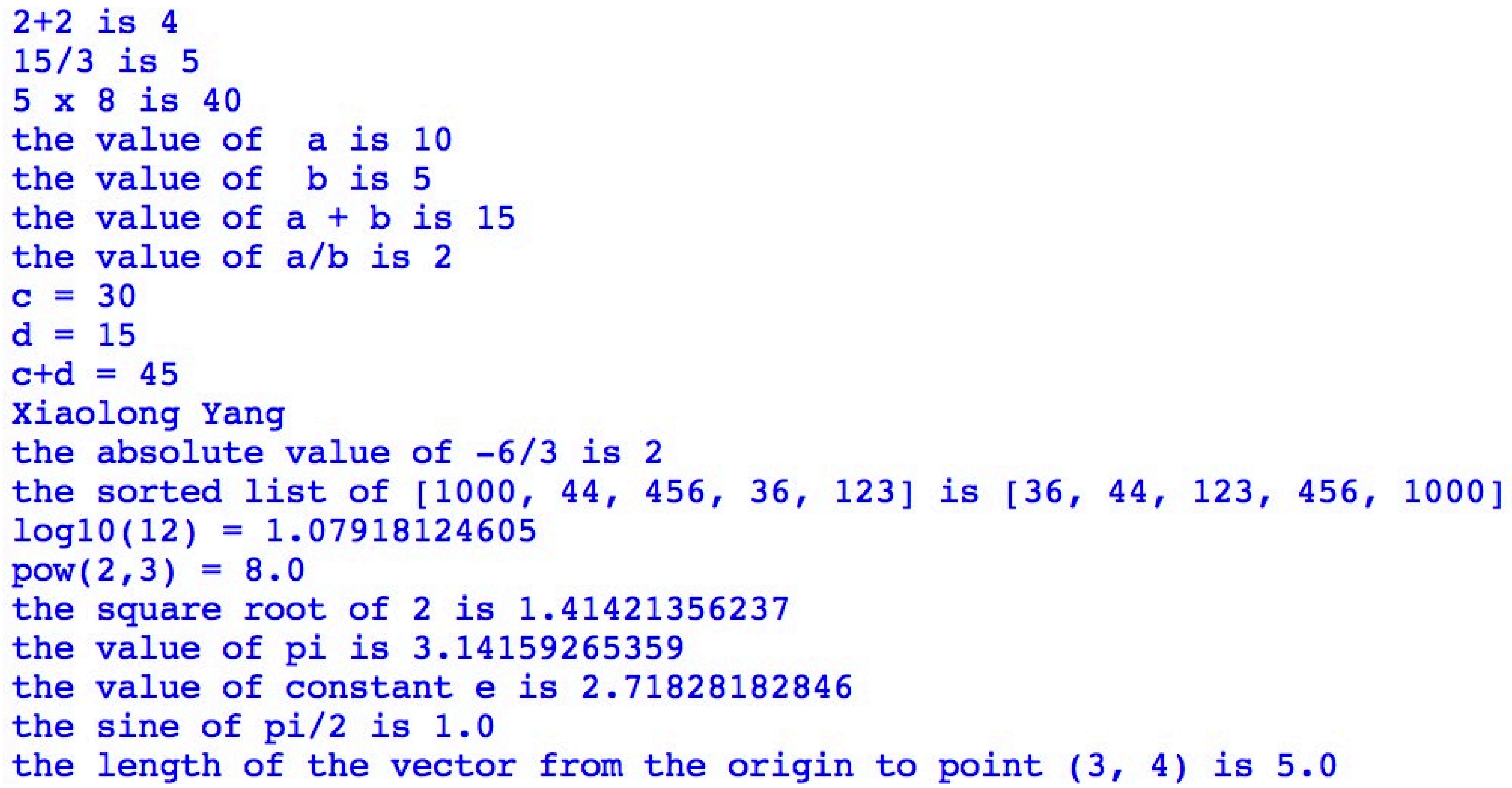
Code breakdown, part 2

```
# assign multiple values to multiple variables at once
c , d = 30, 15
print 'c =', c
print 'd =', d
print 'c+d =', c+d

print 'Xiaolong ' + 'Yang' #string manipulation

# use some math functions below
print 'the absolute value of -6/3 is', abs(-6/3)
# doing some sorting
x=1000, 44, 456, 36, 123 #assign values to a Python data container
print 'the sorted list of [1000, 44, 456, 36, 123] is', sorted(x)
```

Code breakdown, part 3

```python
print 'log10(12) =', math.log10(12) #calculate the base-10 logarithm of 12
print 'pow(2,3) =', math.pow(2,3)   #calculate 2 raised to the power 3
print 'the square root of 2 is', math.sqrt(2) #calculate the square root of 2
# display the value of pi
print 'the value of pi is', math.pi
# display the base of the natural logarithm
print 'the value of constant e is', math.e
# calculate the sine of pi/2 radians
print 'the sine of pi/2 is', math.sin(math.pi/2)

''' calculate the Euclidean norm, sqrt(x*x + y*y).
This is the length of the vector from the origin to point (x, y)'''
print 'the length of the vector from the origin to point (3, 4) is', math.hypot(3,4)
```

The output of the calculator_to_module program



```
2+2 is 4
15/3 is 5
5 x 8 is 40
the value of  a is 10
the value of  b is 5
the value of a + b is 15
the value of a/b is 2
c = 30
d = 15
c+d = 45
Xiaolong Yang
the absolute value of -6/3 is 2
the sorted list of [1000, 44, 456, 36, 123] is [36, 44, 123, 456, 1000]
log10(12) = 1.07918124605
pow(2,3) = 8.0
the square root of 2 is 1.41421356237
the value of pi is 3.14159265359
the value of constant e is 2.71828182846
the sine of pi/2 is 1.0
the length of the vector from the origin to point (3, 4) is 5.0
```

Python Basics

Variables, Names (Identifiers) in Python

- A variable is a name that refers to a value.
- Allowed characters in a name must be letters (a-z, A-Z), digits (0-9), and underscores (_).
- It must begin with a **letter** or **underscore**. Special names usually begin and end in single/double underscores.
- Names are case sensitive.
- Identifiers can be of unlimited length.
- Names should be meaningful. Multi-word names can be underscore separated (e.g., cost_of_event) or camelcase (e.g., costOfEvent).
- The Python keywords cannot be used as names, these include the following (in IDEL these keywords are automatically colored orange):

and	def	exec	if	not	return
assert	del	finally	import	or	try
break	elif	for	in	pass	while
class	else	from	is	print	yield
continue	except	global	lambda	raise	

Basic (built-in) Python Data Types

- Numbers
  - * int (single integers)
  - * long (long integers)
  - * float (floating point real numbers)
  - * complex (complex numbers)
- String
- List
- Tuple
- Dictionary
- Set

Lists, tuples and dictionaries are Python's general purpose built-in data containers. These are high performance containers, which are one of the reasons why Python becomes popular rapidly.

Python Number Types

- **int (single integers):** positive or negative whole numbers with no decimal point.
- **long (long integers):** integers of unlimited size, written like integers and followed by a lowercase or uppercase L.
- **float (floating point real numbers):** real numbers written with a decimal point dividing the integer and fractional parts. Floats may also be in scientific notation, e.g., 10.5e2 = 10.5 x 10^2).
- **complex (complex numbers):** they are of the form a + bJ, where a (the real part) and b (the imaginary part) are floats.

Python Strings

- Strings are immutable sequences of characters. If you want to modify a string, you must produce a new string.
- String are enclosed in between quotation marks. Python allows for pairs of single, double, or triple quotes (e.g., 'Cornell', "Cornell" and '''Cornell''').
- Python indexes the characters in a string, starting from 0 if you count from the left end, but starting from -1 if you start from the right end.

Character	C	o	r	n	e	l	l
Index from left	0	1	2	3	4	5	6
Index from right	-7	-6	-5	-4	-3	-2	-1

- Python strings have many operators:
  - The slice operator ([] and [:])
  - The concatenation operator (+)
  - The repetition operator (*)
  - The membership operator (in and not in)
  - The format operator (%)

Python Strings

The slice operator ([] and [:])

- Use operator [] to extract a single character

```
>>> a = 'Cornell'
>>> b = 'Statistics'
>>> a[0]
>>> a[2]
>>> a[-5]
>>> b[-1]
>>> a[100]
>>> b[-100]
```

- Use operator [:] (or [startIndex:pastIndex]) $\xrightarrow{\hspace{4em}}$ to extract a substring

```
>>> a[0:4]
>>> b[0:4]
>>> a[-4: -1]
>>> b[:4]
>>> b[5:]
>>> a[:]
>>> a[3:0]
>>> a[3:100]
>>> a[:-1]
```

Python Strings

The concatenation operator (+)

The concatenation operator connect two strings together without a space in between. See the following example.

```
>>> a = 'Master of Professional Studies'
>>> b = 'in Applied Statistics'
>>> a + b
'Master of Professional Studiesin Applied Statistics'
```

no space in between

You can add one as follows:

```
>>> a + ' ' + b
'Master of Professional Studies in Applied Statistics'
```

the space added

Python Strings

The repetition operator (*)

The repetition operator repeats the same string for the specified number of times without a space in between. See the following example.

```
>>> c = 'MPS'
>>> 5*c
'MPSMPSMPSMPSMPS'
```

no space in between

You can add one as follows:

```
>>> d = c + ' '
>>> 5*d
'MPS MPS MPS MPS MPS '
```

the spaces added

You can add any character, e.g., *:

```
>>> d = c + '*'
>>> 5*d
'MPS*MPS*MPS*MPS*MPS*'
```

the * characters added

Python Strings

The membership operator (**in** and **not in**)

- The membership operator **in** returns True if a character exists in the given string.
- The membership operator **not in** returns True if a character does not exist in the given string.

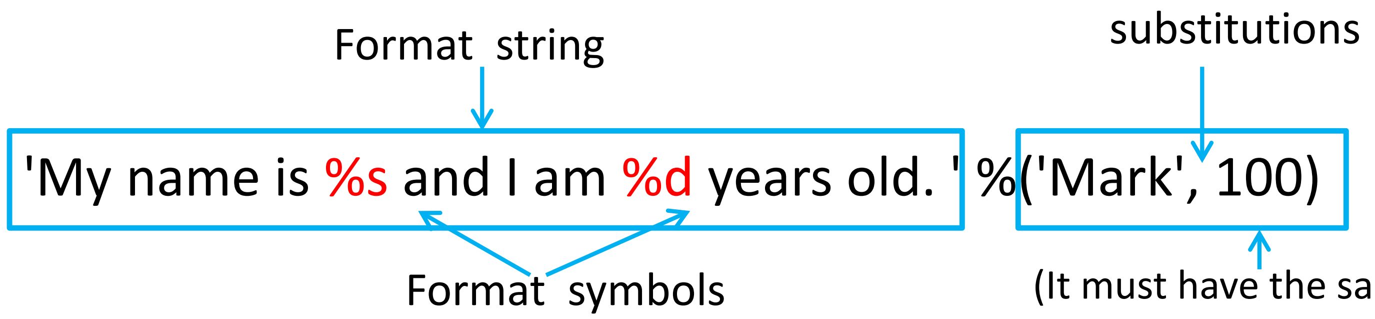
```
>>> e = 'Go, Big Red Go!'
>>> 'Big' in e
True
>>> 'Red' not in e
False
>>> '!' in e
True
>>> 'Cornell' not in e
True
>>> 'big' in e
```

Python Strings

The format operator (%)

The format operator % performs string formatting. This is one of the best features, making Python an excellent string handing language. Try the following:

```
>>> print 'My name is %s and I am %d years old. ' %('Mark', 100)
My name is Mark and I am 100 years old.
```



```
'My name is %s and I am %d years old. ' %('Mark', 100)
```

(It must have the same number of values as the format symbols, in the same order and with the same data types.)

Common string format symbols:

- %s (convert a string via str() prior to formatting)
- %d (for decimal integers)
- %e (for exponential notation)
- %f (for floating point real number)

Built-in String Methods/Functions

The following methods/functions can be used to handle strings effectively

- capitalize()
- center(width, fillchar)
- count(str[, start[, end]])
- decode([encoding[, errors]])
- encode([encoding[, errors]])
- endswith(suffix[, start[, end]])
- expandtabs([tabsize])
- find(str[, start[, end]])
- index(str[, start[, end]])
- isalnum()
- isalpha()
- isdigit()
- islower()
- isnumeric()
- isspace()
- istitle()
- isupper()
- join(seq)
- len(string)
- ljust(width[, fillchar])
- lower()
- lstrip()
- maketrans()
- max(str)
- min(str)
- replace(old, new [, max])
- rfind(str[, start[, end]])
- rindex(str[, start[, end]])
- rjust(width,[, fillchar])
- rstrip()
- split([str [, maxsplit]])
- splitlines([num])
- startswith(str[, start[, end]])
- strip([chars])
- swapcase()
- title()
- translate(table[, deletechars])
- upper()
- zfill (width)
- isdecimal()

Examples: Built-in String Methods/Functions

- **capitalize()**: Capitalizes first letter of the string.

```
>>> a = 'computer'
>>> a.capitalize()
'Computer'
```

- **len(string)**: Returns the length of the string.

```
>>> len(a)
8
>>> len('computer')
8
```

- **count(str[, start[, end]])**: Counts how many times str occurs in the string, or in a substring of the string if starting index **start** and ending index **end** are given.

```
>>> s='Master of Professional Studies'
>>> s.count('s')
4
>>> s.count('s', 0, 6)
1
>>> s. count('s', 6)
```

- **find(str[, start[, end]])**: Find the 1st occurrence of str from the left end of the string, and return the index at which the substring begins or -1 if not found.

```
>>> s. find ('fess')
13
>>> s.find('fess', 6, 12)
-1
```

Examples: Built-in string methods/functions (cont'd)

- **join(collection)**:Merges the string elements of the collection into a string, with the separator string.

```
>>> ' '.join(('1', '2', '3', '4'))
'1 2 3 4'
>>> '-'. join(('1', '2', '3', '4'))
'1-2-3-4'
```

- **replace(old, new [, max])**: Replace all or up to max number of occurrences of an old substring with the new string.

```
>>> s.replace('s', 'S')
'MaSter of ProfeSSional StudieS'
```

- **split([str[, maxsplit]])**: Splits the string into substrings using space or a user-specified str as a delimiter. If *maxsplit* is given, at most *maxsplit* splits are done (thus, the result will have at most *maxsplit+1* string elements).

```
>>> s1 = 'Cornell University Department of Statistical Science'
>>> s1.slpit()
['Cornell', 'University', 'Department', 'of', 'Statistical', 'Science']
>>> s1.split('a')
['Cornell University Dep', 'rtment of St', 'tistic', 'l Science']
>>> s1.split('a',2)
['Cornell University Dep', 'rtment of St', 'tistical Science']
```

Extra Notes About Python Strings

- Use triple quotes (''' or """) to define a string with formatting info, e.g., defining a string of a book.

```
>>> ss='''Winter is not over soon
	Because the Groundhog said so yesterday

	Let's wait and see

'''
>>> ss
"Winter is not over soon\n\tBecause the Groundhog said so yesterday\n\n\tLet's wait and see\n\n"
>>> print ss
Winter is not over soon
	Because the Groundhog said so yesterday

	Let's wait and see
```

Python Lists

Definition:

It is a named, square-bracket-enclosed, comma-separated collection of ordered arbitrary data values (or elements), where each value (or element) is identified by an index. For example:

```
list1 = ['Country', 'University', 'Department', 'Student']
list2 = ['road', 'path', 'driveway', 'avenue', 100]
list3 = [1, 2, 3, 4, 5,[1.5, 6.8, 10.5]]
courses = ['Math', 'Statistics', 'Python Programming', 'Database']
list4 =[] (an empty list)
```

Features of Python lists:

- You can include **different types of data** in the same list, which makes Python lists very versatile.
- Lists are dynamic arrays since you can index elements in a list, select sub-ranges and add/remove list elements.
- The elements of a list is **mutable**, meaning that you can change the value of a list element or add/remover an element(s).

List Indexing

(the same as in string indexing)

list1 = ['Country', 'University', 'Department', 'Student']

List element	Country	University	Department	Student
Index from left	0	1	2	3
Index from right	-4	-3	-2	-1

list3 = [1, 2, 3, 4, [1.5, 6.8, 10.5]]

List element	1	2	3	4	5	[1.5, 6.8, 10.5]
Index from left	0	1	2	3	4	5
Index from right	-6	-5	-4	-3	-2	-1

List Operators

(the same as all the string operators except the format operator %)

- The slice operator ([] and [:])

```
>>> list1[0]
'Country'
>>> list1[1:3]
['University', 'Department']
```

- The concatenation operator (+)

```
>>> list1 + list 3
['Country', 'University', 'Department', 'Student', 1, 2, 3, 4, 5, [1.5,
6.8, 10.5]]
```

- The repetition operator (*)

```
>>> list1[3]*3
'StudentStudentStudent'
```

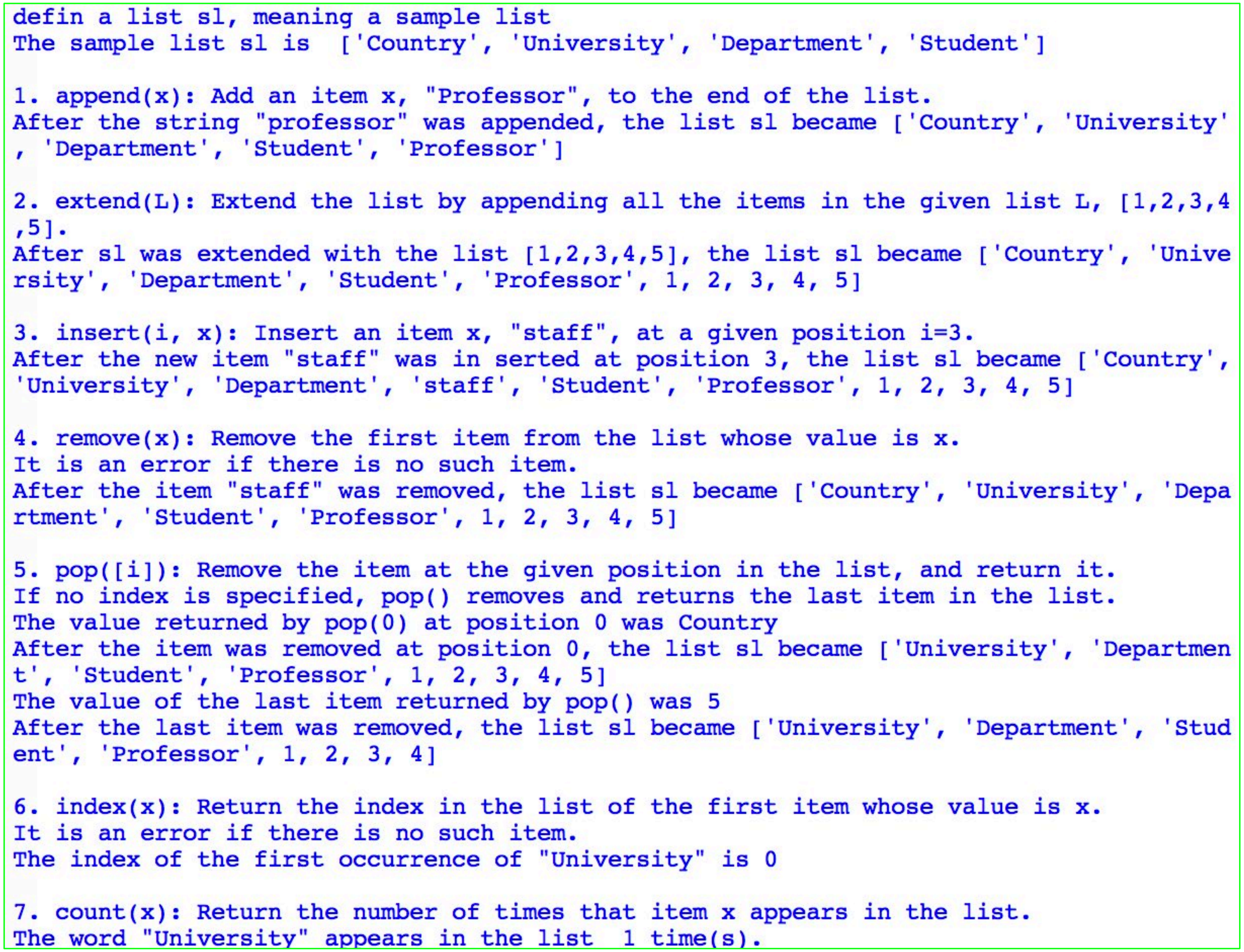
- The membership operator (in and not in)

```
>>> 'University' in list1
True
>>> 100 not in list 3
True
```

Built-in List Methods and Functions

- **append(x)**: Adds an item x to the end of the list.
- **extend(L)**: Extends the list by appending all the items in the given list L.
- **insert(i, x)**: Inserts an item x at a given position i.
- **remove(x)**: Removes the first item from the list whose value is *x*. It is an error if there is no such item.
- **pop([i])**: Removes the item at the given position i in the list, and returns it. If no index is specified, pop() removes and returns the last item in the list.
- **index(x)**: Returns the index in the list of the first item whose value is *x*. It is an error if there is no such item.
- **count(x)**: Returns the number of times *x* appears in the list.
- **sort()**: Sorts the items of the list, in place.
- **reverse()**: Reverses the elements of the list, in place.
- **len(list)**: Gives the total length of the list.
- **max(list)**: Returns element from the list with max value.
- **min(list)**: Returns element from the list with min value.
- **list(seq)**: Converts seq into a list.

The output (partial) of a Program Using Python Built-in List Methods & Functions

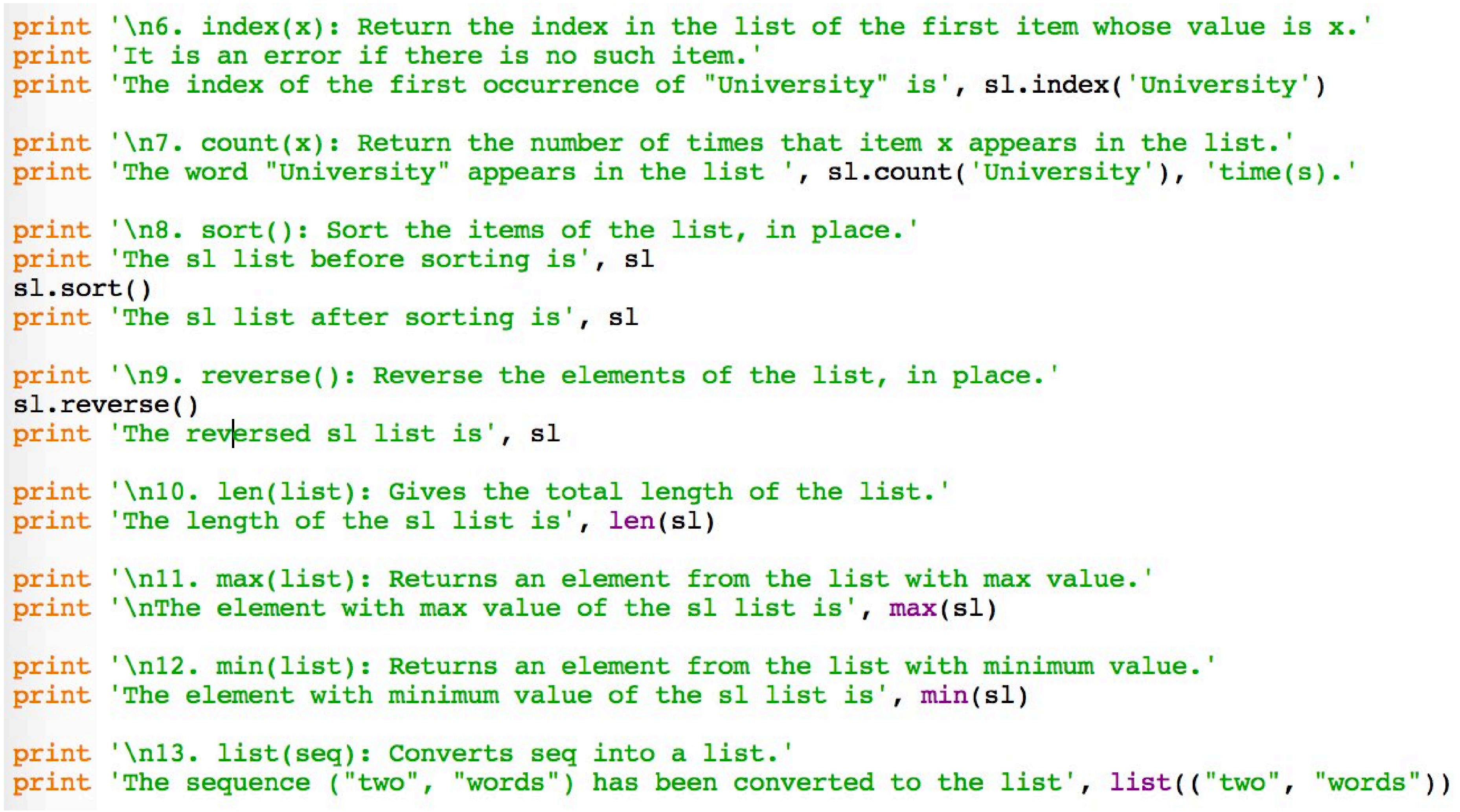


```
defin a list sl, meaning a sample list
The sample list sl is  ['Country', 'University', 'Department', 'Student']

1. append(x): Add an item x, "Professor", to the end of the list.
After the string "professor" was appended, the list sl became ['Country', 'University'
, 'Department', 'Student', 'Professor']

2. extend(L): Extend the list by appending all the items in the given list L, [1,2,3,4
,5].
After sl was extended with the list [1,2,3,4,5], the list sl became ['Country', 'Unive
rsity', 'Department', 'Student', 'Professor', 1, 2, 3, 4, 5]

3. insert(i, x): Insert an item x, "staff", at a given position i=3.
After the new item "staff" was in serted at position 3, the list sl became ['Country',
'University', 'Department', 'staff', 'Student', 'Professor', 1, 2, 3, 4, 5]

4. remove(x): Remove the first item from the list whose value is x.
It is an error if there is no such item.
After the item "staff" was removed, the list sl became ['Country', 'University', 'Depa
rtment', 'Student', 'Professor', 1, 2, 3, 4, 5]

5. pop([i]): Remove the item at the given position in the list, and return it.
If no index is specified, pop() removes and returns the last item in the list.
The value returned by pop(0) at position 0 was Country
After the item was removed at position 0, the list sl became ['University', 'Departmen
t', 'Student', 'Professor', 1, 2, 3, 4, 5]
The value of the last item returned by pop() was 5
After the last item was removed, the list sl became ['University', 'Department', 'Stud
ent', 'Professor', 1, 2, 3, 4]

6. index(x): Return the index in the list of the first item whose value is x.
It is an error if there is no such item.
The index of the first occurrence of "University" is 0

7. count(x): Return the number of times that item x appears in the list.
The word "University" appears in the list  1 time(s).
```

A Program Using Python Built-in List Methods & Functions (Part 1)

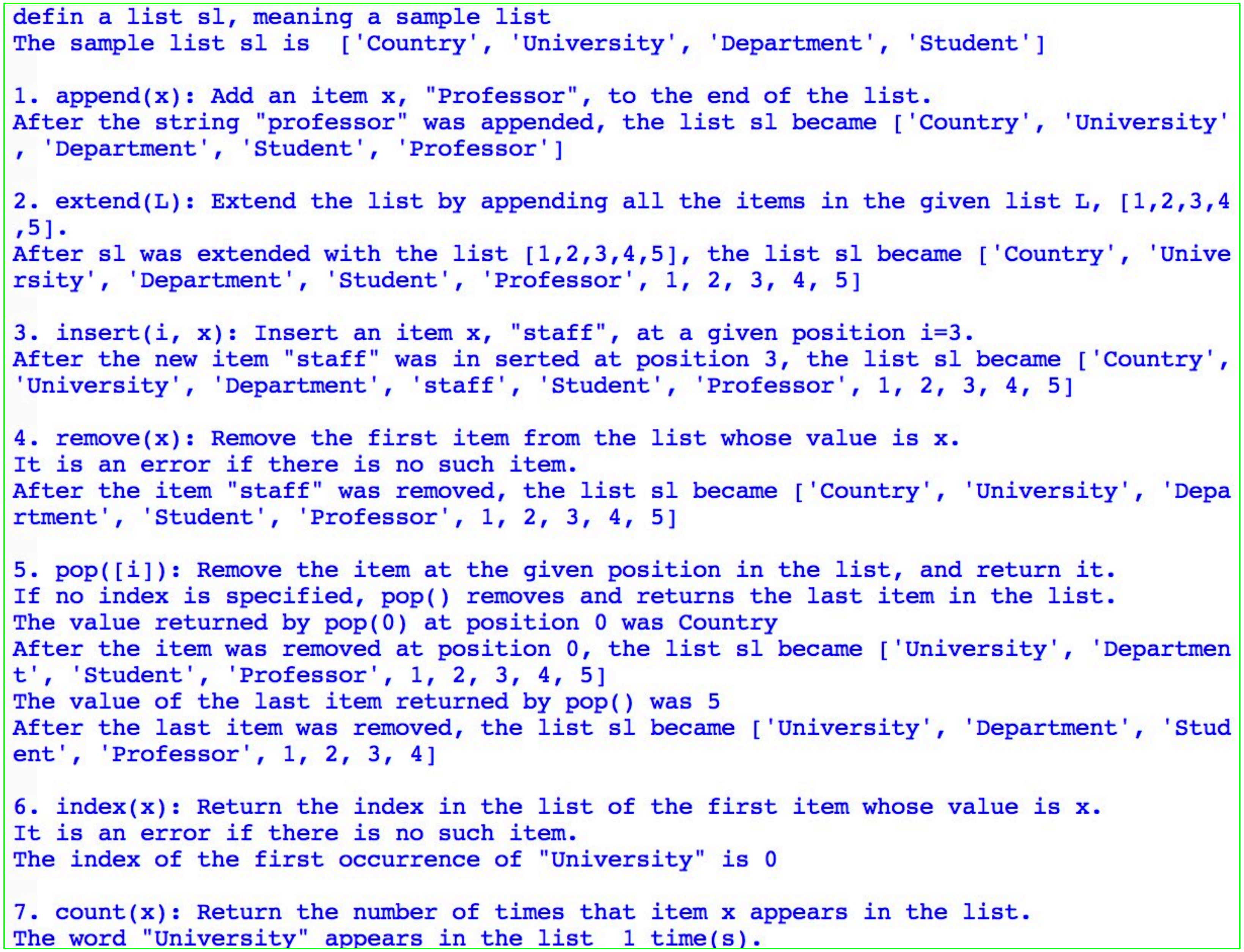
```
'''This program is to practice Python list methods and functions.
The functionality of each method/function is explained first, then
an example follows. --XY'''

print 'defin a list sl, meaning a sample list'
sl = ['Country', 'University', 'Department', 'Student']
print 'The sample list sl is ', sl

print '\n1. append(x): Add an item x, "Professor", to the end of the list.'
sl.append('Professor')
print 'After the string "professor" was appended, the list sl became', sl

print '\n2. extend(L): Extend the list by appending all the items in the given list L, [1,2,3,4,5].'
sl.extend([1,2,3,4,5])
print 'After sl was extended with the list [1,2,3,4,5], the list sl became', sl

print '\n3. insert(i, x): Insert an item x, "staff", at a given position i=3.'
sl.insert(3, 'staff')
print 'After the new item "staff" was in serted at position 3, the list sl became', sl

print '\n4. remove(x): Remove the first item from the list whose value is x.'
print 'It is an error if there is no such item.'
sl.remove('staff')
print 'After the item "staff" was removed, the list sl became', sl

print '\n5. pop([i]): Remove the item at the given position in the list, and return it.'
print 'If no index is specified, pop() removes and returns the last item in the list.'
print 'The value returned by pop(0) at position 0 was', sl.pop(0)
print 'After the item was removed at position 0, the list sl became', sl

print 'The value of the last item returned by pop() was', sl.pop() #removes the last item of sl
print 'After the last item was removed, the list sl became', sl
```

A Program Using Python Built-in List Methods & Functions (Part 2)



```
print '\n6. index(x): Return the index in the list of the first item whose value is x.'
print 'It is an error if there is no such item.'
print 'The index of the first occurrence of "University" is', sl.index('University')

print '\n7. count(x): Return the number of times that item x appears in the list.'
print 'The word "University" appears in the list ', sl.count('University'), 'time(s).'

print '\n8. sort(): Sort the items of the list, in place.'
print 'The sl list before sorting is', sl
sl.sort()
print 'The sl list after sorting is', sl

print '\n9. reverse(): Reverse the elements of the list, in place.'
sl.reverse()
print 'The reversed sl list is', sl

print '\n10. len(list): Gives the total length of the list.'
print 'The length of the sl list is', len(sl)

print '\n11. max(list): Returns an element from the list with max value.'
print '\nThe element with max value of the sl list is', max(sl)

print '\n12. min(list): Returns an element from the list with minimum value.'
print 'The element with minimum value of the sl list is', min(sl)

print '\n13. list(seq): Converts seq into a list.'
print 'The sequence ("two", "words") has been converted to the list', list(("two", "words"))
```

The output (partial) of a Program Using Python Built-in List Methods & Functions



```
defin a list sl, meaning a sample list
The sample list sl is  ['Country', 'University', 'Department', 'Student']

1. append(x): Add an item x, "Professor", to the end of the list.
After the string "professor" was appended, the list sl became ['Country', 'University'
, 'Department', 'Student', 'Professor']

2. extend(L): Extend the list by appending all the items in the given list L, [1,2,3,4
,5].
After sl was extended with the list [1,2,3,4,5], the list sl became ['Country', 'Unive
rsity', 'Department', 'Student', 'Professor', 1, 2, 3, 4, 5]

3. insert(i, x): Insert an item x, "staff", at a given position i=3.
After the new item "staff" was in serted at position 3, the list sl became ['Country',
'University', 'Department', 'staff', 'Student', 'Professor', 1, 2, 3, 4, 5]

4. remove(x): Remove the first item from the list whose value is x.
It is an error if there is no such item.
After the item "staff" was removed, the list sl became ['Country', 'University', 'Depa
rtment', 'Student', 'Professor', 1, 2, 3, 4, 5]

5. pop([i]): Remove the item at the given position in the list, and return it.
If no index is specified, pop() removes and returns the last item in the list.
The value returned by pop(0) at position 0 was Country
After the item was removed at position 0, the list sl became ['University', 'Departmen
t', 'Student', 'Professor', 1, 2, 3, 4, 5]
The value of the last item returned by pop() was 5
After the last item was removed, the list sl became ['University', 'Department', 'Stud
ent', 'Professor', 1, 2, 3, 4]

6. index(x): Return the index in the list of the first item whose value is x.
It is an error if there is no such item.
The index of the first occurrence of "University" is 0

7. count(x): Return the number of times that item x appears in the list.
The word "University" appears in the list  1 time(s).
```

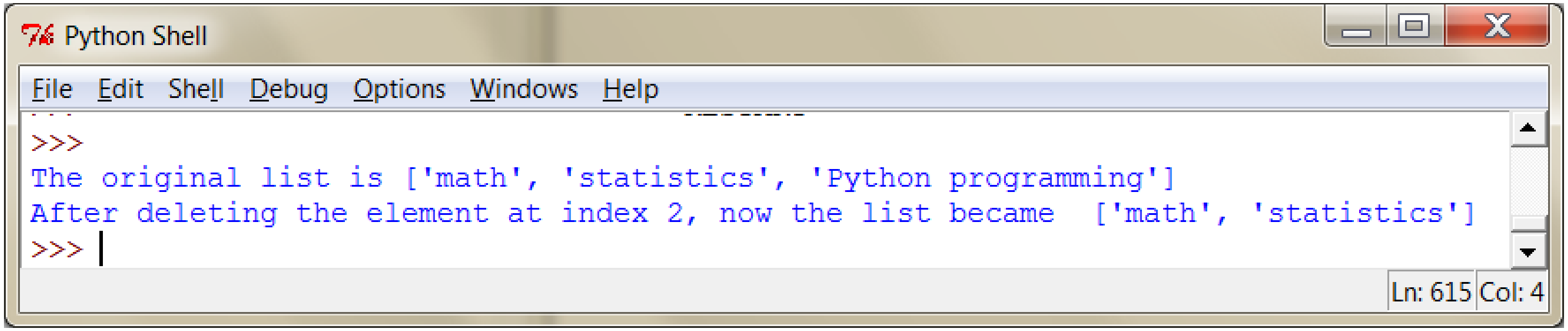
Updating the Values of Mutable List Elements

```
>>> courses = ['math', 'statistics', 'Python programming',
'database', 4.3, 3.9]
>>> course[5] = 4.0
>>> courses
['math', 'statistics', 'Python programming', 'database',
4.3, 4.0]
>>> courses[0] = 'Linear Algebra'
>>> courses
['Linear Algebra', 'statistics', 'Python programming',
'database', 4.3, 4.0]
```

Deleting List Elements With the del Statement

In addition to the remove() method, you can also use the del statement to remove list elements. Write the following script in IDEL and run it:

```
list1 = ['math', 'statistics', 'Python programming']
print 'The original list is', list1
del list1[2];
print "After deleting the element at index 2, now the list \
became ", list1
```

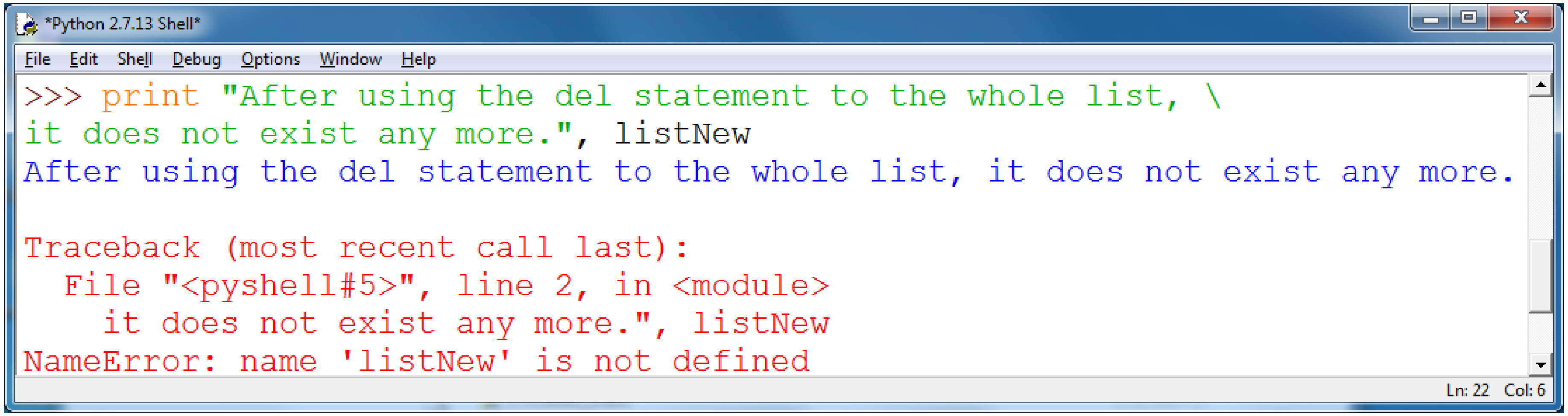


```
>>>
The original list is ['math', 'statistics', 'Python programming']
After deleting the element at index 2, now the list became  ['math', 'statistics']
>>> 
```

Deleting the Whole List With the del Statement

The del statement can also be used to remove the whole list:

```
listNew = [1,2,3,4,5]
print 'The original list is', listNew
del listNew;
print "After using the del statement to the whole list, \
it does not exist any more.", listNew
```



```
>>> print "After using the del statement to the whole list, \
it does not exist any more.", listNew
After using the del statement to the whole list, it does not exist any more.

Traceback (most recent call last):
  File "<pyshell#5>", line 2, in <module>
    it does not exist any more.", listNew
NameError: name 'listNew' is not defined
```

Lists and Matrixes

Nested lists are often used to represent matrixes. The matrix:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

can be represented as:

```
>>> matrix = [[1, 2, 3, 4], [5, 6, 7, 8],[9, 10, 11, 12], [13, 14, 15, 16]]
```

where matrix is a list with four elements, each of which is a row of the matrix.

```
>>> matrix[0]
[1, 2, 3, 4]
>>> matrix[3]
[13, 14, 15, 16]
```

A single element can be extracted using the double-index form:

```
>>> matrix[3][3]        # The first index selects the row and the second index selects the column.
16
```

Think about how to represent the matrix with a list of columns.

Python Tuples

Definition:

It is a named, parenthesis-enclosed, comma-separated collection of ordered values (or elements), where each value is identified by an index. For example:

t1 = ('Country', 'University', 'Department', 'Student')

t2 = ('road', 'path', 'driveway', 'avenue', 100)

t3 = (1, 2, 3, 4, (1.5, 6.8, 10.5))

courses = ('math', 'statistics', 'Python programming', 'database')

t4 = () (an empty tuple)

t5 = ('singleton',) (one item tuple; the trailing ',' assures that this is a tuple, not an expression.)

Features of tuples:

- Tuples are light-weight lists, so they share many list features, operators, all the functions and two methods, index(x) and count(x).
- The main difference is that tuples are **immutable**, i.e., the values of the tuple elements cannot be changed, any list methods that make changes cannot be used with tuples, e.g., append(x), insert(i,x), remove (x), sort(), and reverse().

Some Special Notes About Tuples

- When you define a tuple, you may omit the parentheses.

```
>>> tuple1 = 'Country', 'University', 'Department', 'Student'
>>> tuple1
('Country', 'University', 'Department', 'Student')
```

- When you access values in tuple, you use the square brackets for slicing along with the index or indices to obtain the value at that index.

```
>>> tuple1[0]
'Country'
>>> tuple1[1:3]
('University', 'Department')
```

- You will get an error when trying to update a tuple since a tuple is immutable (same as a string), but you can remove the entire tuple with a del statement.

```
>>> tuple1[0] ='Nation'
Traceback (most recent call last):
  File "<pyshell#158>", line 1, in <module>
    tuple1[0] = 'Nation'
TypeError: 'tuple' object does not support item assignment
>>> del tuple1
>>> tuple1
Traceback (most recent call last):
  File "<pyshell#162>", line 1, in <module>
    tuple1
NameError: name 'tuple1' is not defined
```

Python Dictionaries (Associative Arrays or Hash Tables)

Definition: A dictionary is an unordered set of comma-separated key:value pairs placed within the braces, with the requirement that the keys are immutable and unique within one dictionary. The values can be any Python object.

```
>>> dictID = {'Elise':'2038', 'Beth':'9999', 'Mary':'8888'}  # all strings
>>> dictInventory = {'apples':12 , 'eggs':23, 'pears':78}  # strings and integers
>>> dictEmpty = {}  # an empty dictionary
```

Features of dictionaries:

- Ability to look up a value by key

```
>>> dictID['Elise']  # use square brackets only
'2038'
>>> dictInventory['pears']
78
```

- Ability to add a new key:value pair dynamically, so it is mutable.

```
>>> dictID['David'] = '1234'   # again, use square brackets only
>>>  dictID
{'Elise': '2038', 'Beth': '9999', 'Mary': '8888', 'David': '1234'}
```

- Ability to update and delete dictionary elements.

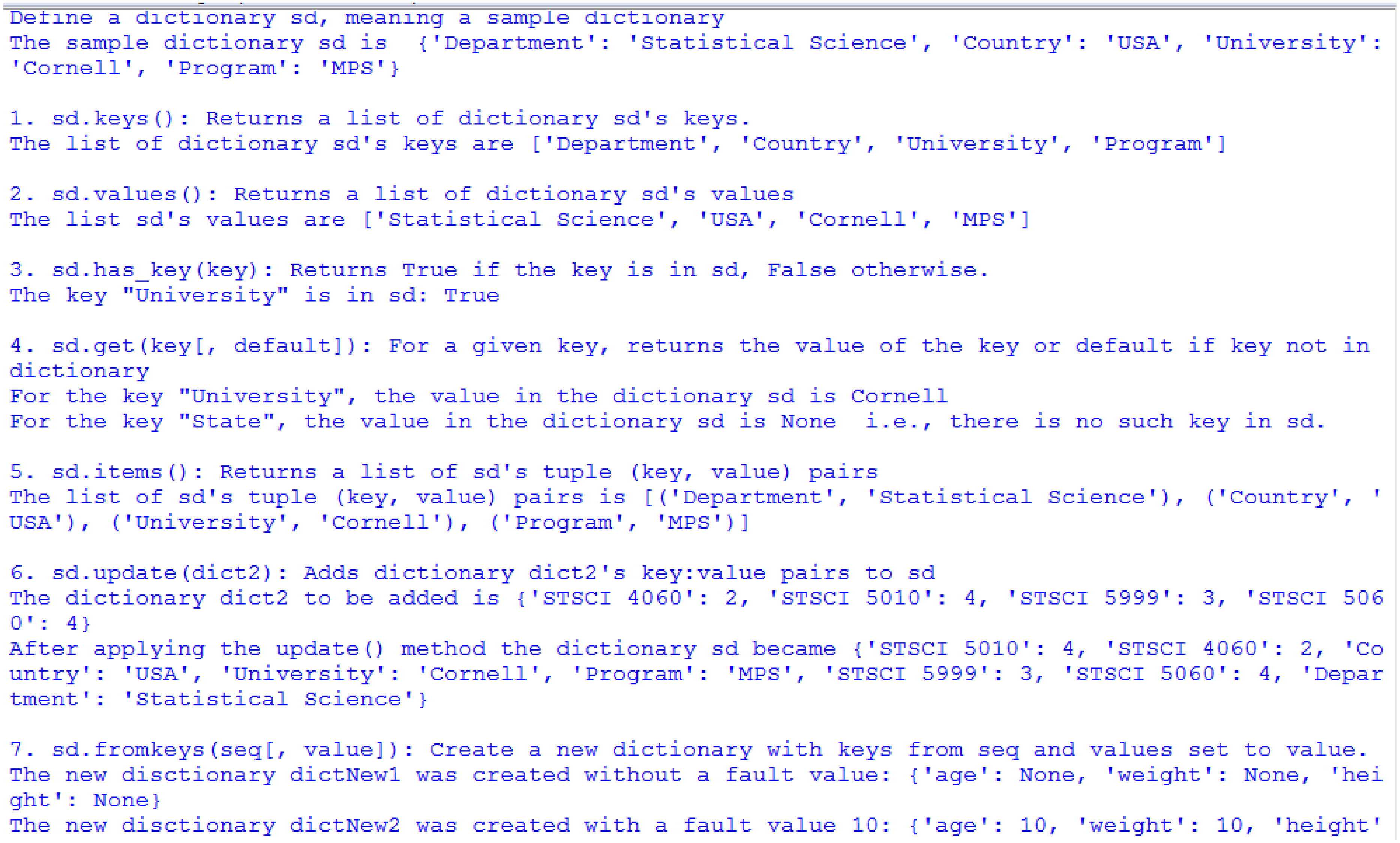
```
>>> dictInventory['apple'] = 100
>>> del dictID['Mary']  # remove  the entry with key 'Mary'
```

Python Dictionary (Dict) Methods and Functions

- **Dict.keys()**: Returns a list of dictionary keys
- **Dict.values()**: Returns a list of dictionary values
- **Dict.has_key(key)**: Returns *true* if key is in dictionary *Dict*, *false* otherwise
- **Dict.get(key[, default])**: For a given *key*, returns the value of the key or default if key not in dictionary
- **Dict.items()**: Returns a list of *Dict*'s tuple pairs (key, value)
- **Dict.update(dict2)**: Adds dictionary *dict2*'s key:value pairs to *Dict*
- **dict.fromkeys(seq[, value])**: Create a new dictionary with keys from seq and values *set* to *value*
- **Dict.setdefault(key[, default])**: Similar to get(), but will set Dict[key]=default if *key* is not already in Dict
- **Dict.copy()**: Returns a shallow copy of dictionary *Dict* (the content of the dictionary is not copied by value, but just creates a new reference)
- **Dict.clear()**: Removes all elements of dictionary *Dict*
- **len(Dict)**: Returns the total length of the dictionary
- **str(Dict)**: Produces a printable string representation of a dictionary

A Program Using Python Built-in Dictionary Methods & Functions

(Output, partial)



```
Define a dictionary sd, meaning a sample dictionary
The sample dictionary sd is  {'Department': 'Statistical Science', 'Country': 'USA', 'University':
'Cornell', 'Program': 'MPS'}

1. sd.keys(): Returns a list of dictionary sd's keys.
The list of dictionary sd's keys are ['Department', 'Country', 'University', 'Program']

2. sd.values(): Returns a list of dictionary sd's values
The list sd's values are ['Statistical Science', 'USA', 'Cornell', 'MPS']

3. sd.has_key(key): Returns True if the key is in sd, False otherwise.
The key "University" is in sd: True

4. sd.get(key[, default]): For a given key, returns the value of the key or default if key not in
dictionary
For the key "University", the value in the dictionary sd is Cornell
For the key "State", the value in the dictionary sd is None  i.e., there is no such key in sd.

5. sd.items(): Returns a list of sd's tuple (key, value) pairs
The list of sd's tuple (key, value) pairs is [('Department', 'Statistical Science'), ('Country', '
USA'), ('University', 'Cornell'), ('Program', 'MPS')]

6. sd.update(dict2): Adds dictionary dict2's key:value pairs to sd
The dictionary dict2 to be added is {'STSCI 4060': 2, 'STSCI 5010': 4, 'STSCI 5999': 3, 'STSCI 506
0': 4}
After applying the update() method the dictionary sd became {'STSCI 5010': 4, 'STSCI 4060': 2, 'Co
untry': 'USA', 'University': 'Cornell', 'Program': 'MPS', 'STSCI 5999': 3, 'STSCI 5060': 4, 'Depar
tment': 'Statistical Science'}

7. sd.fromkeys(seq[, value]): Create a new dictionary with keys from seq and values set to value.
The new disctionary dictNew1 was created without a fault value: {'age': None, 'weight': None, 'hei
ght': None}
The new disctionary dictNew2 was created with a fault value 10: {'age': 10, 'weight': 10, 'height'
```

A Program Using Python Built-in Dictionary Methods & Functions Part 1

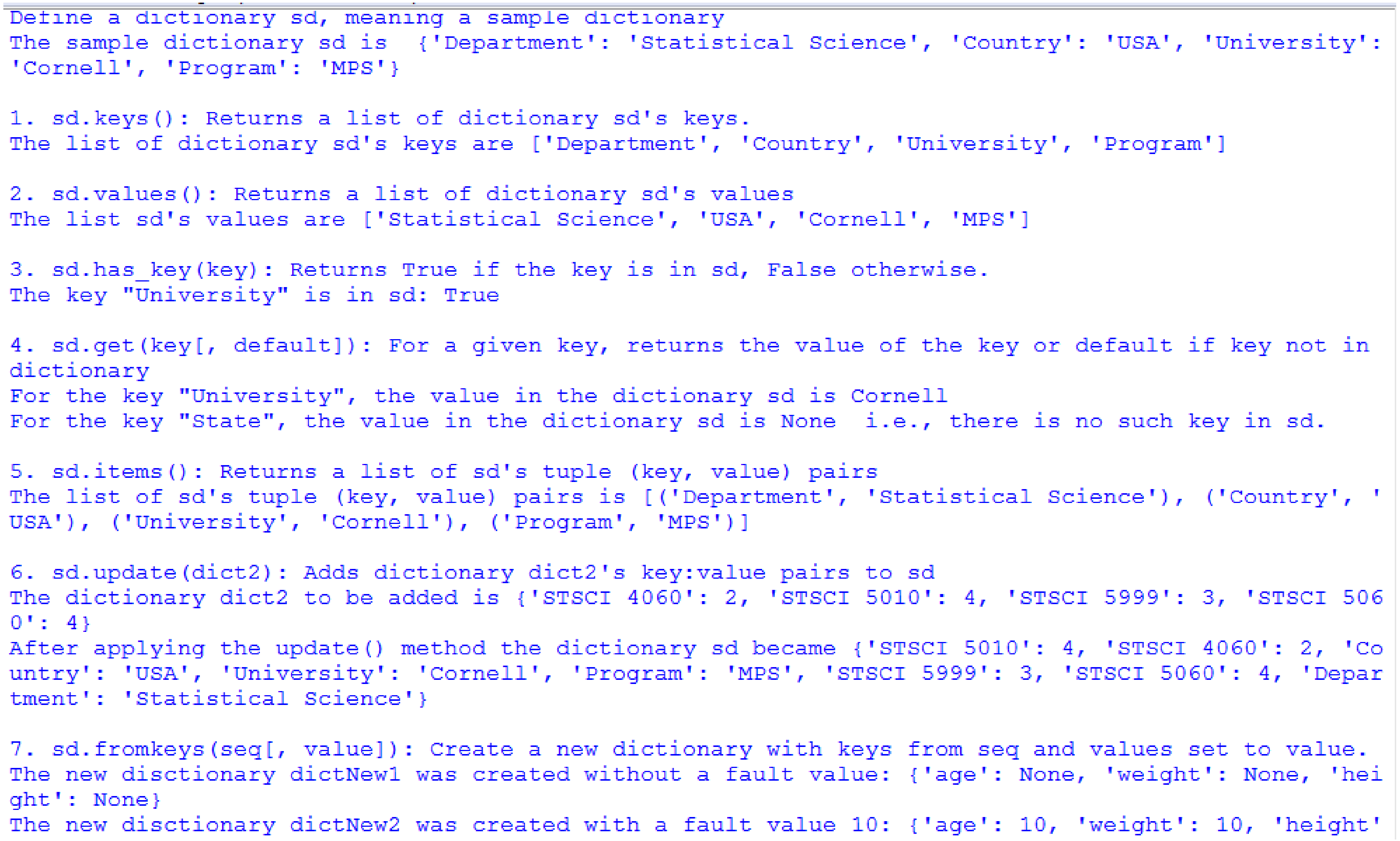
```
'''This program is to practice Python dictionary methods and functions.
The functionality of each method/function is explained first, then
an example follows. --XY'''

print 'Define a dictionary sd, meaning a sample dictionary'
sd = {'Country':'USA', 'University':'Cornell', 'Department':'Statistical Science', 'Program':'MPS'}
print 'The sample dictionary sd is ', sd

print "\n1. sd.keys(): Returns a list of dictionary sd's keys."
print "The list of dictionary sd's keys are", sd.keys()

print "\n2. sd.values(): Returns a list of dictionary sd's values"
print "The list sd's values are", sd.values()

print '\n3. sd.has_key(key): Returns True if the key is in sd, False otherwise.'
print 'The key "University" is in sd:', sd.has_key('University')

print '\n4. sd.get(key[, default]): For a given key, returns the value of the key or \
default if key not in dictionary'
print 'For the key "University", the value in the dictionary sd is', sd.get('University')
print 'For the key "State", the value in the dictionary sd is', sd.get('State', 'None'), ' \
i.e., there is no such key in sd.'

print "\n5. sd.items(): Returns a list of sd's tuple (key, value) pairs"
print "The list of sd's tuple (key, value) pairs is", sd.items()

print "\n6. sd.update(dict2): Adds dictionary dict2's key:value pairs to sd"
dict2 = {'STSCI 5010':4, 'STSCI 4060':2, 'STSCI 5999':3, 'STSCI 5060':4}
print 'The dictionary dict2 to be added is', dict2
sd.update(dict2)
print 'After applying the update() method the dictionary sd became', sd
```

A Program Using Python Built-in Dictionary Methods & Functions Part 2

```
print '\n7. sd.fromkeys(seq[, value]): Create a new dictionary with keys from seq and \
values set to value.'
seq = ('age', 'weight', 'height')
dictNew1 = sd.fromkeys(seq)
dictNew2 = sd.fromkeys(seq, 10)
print 'The new disctionary dictNew1 was created without a fault value:', dictNew1
print 'The new disctionary dictNew2 was created with a fault value 10:', dictNew2

print '\n8. sd.setdefault(key[, default]): Similar to get(), but will set sd[key]=default \
if key is not already in sd'
print 'When a key "Department" is in the dictionary sd, the value, ', \
      sd.setdefault('Department', 'None'), ' is returned.'
print 'When a key "State" is in the dictionary sd, the default value, ', \
      sd.setdefault('State', 'None'), ' is returned.'

print '\n9. sd.copy(): Returns a shallow copy of dictionary sd.'
sdNew = sd.copy()
print 'sdNew, a shallow copy of dictionary sd is', sdNew

print '\n10. sdNew.clear(): Removes all the elements of dictionary sdNew'
sdNew.clear()
print 'After being applied the clear() method, the sdNew became', sdNew

print '\n11. len(sd): Returns the total length of the dictionary.'
print 'The total length of the dictionary sd is', len(sd)

print '\n12. str(sd): Produces a printable string representation of the dictionary sd.'
print 'The printable string representation of the dictionary sd is:', str(sd)
```

A Program Using Python Built-in Dictionary Methods & Functions

(Output, partial)



```
Define a dictionary sd, meaning a sample dictionary
The sample dictionary sd is  {'Department': 'Statistical Science', 'Country': 'USA', 'University':
'Cornell', 'Program': 'MPS'}

1. sd.keys(): Returns a list of dictionary sd's keys.
The list of dictionary sd's keys are ['Department', 'Country', 'University', 'Program']

2. sd.values(): Returns a list of dictionary sd's values
The list sd's values are ['Statistical Science', 'USA', 'Cornell', 'MPS']

3. sd.has_key(key): Returns True if the key is in sd, False otherwise.
The key "University" is in sd: True

4. sd.get(key[, default]): For a given key, returns the value of the key or default if key not in
dictionary
For the key "University", the value in the dictionary sd is Cornell
For the key "State", the value in the dictionary sd is None  i.e., there is no such key in sd.

5. sd.items(): Returns a list of sd's tuple (key, value) pairs
The list of sd's tuple (key, value) pairs is [('Department', 'Statistical Science'), ('Country', '
USA'), ('University', 'Cornell'), ('Program', 'MPS')]

6. sd.update(dict2): Adds dictionary dict2's key:value pairs to sd
The dictionary dict2 to be added is {'STSCI 4060': 2, 'STSCI 5010': 4, 'STSCI 5999': 3, 'STSCI 506
0': 4}
After applying the update() method the dictionary sd became {'STSCI 5010': 4, 'STSCI 4060': 2, 'Co
untry': 'USA', 'University': 'Cornell', 'Program': 'MPS', 'STSCI 5999': 3, 'STSCI 5060': 4, 'Depar
tment': 'Statistical Science'}

7. sd.fromkeys(seq[, value]): Create a new dictionary with keys from seq and values set to value.
The new disctionary dictNew1 was created without a fault value: {'age': None, 'weight': None, 'hei
ght': None}
The new disctionary dictNew2 was created with a fault value 10: {'age': 10, 'weight': 10, 'height'
```

Python Sets

Definition:

A set is an unordered collection of unique elements. Curly braces or the set() function can be used to create sets. For example:

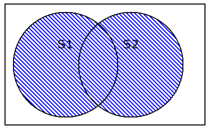
```
>>> s1 = {'SAS', 'SPSS', 'R', 'JMP', 'MINITAB'}
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> fruit = set (basket)
>>> emptySet = set()   # it's not {}
>>> s1
set(['SAS', 'JMP', 'R', 'MINITAB', 'SPSS'])
>>> fruit
set(['orange', 'pear', 'apple', 'banana'])
>>> emptySet
set([])
```

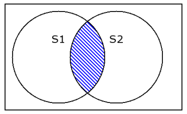
Features of sets:

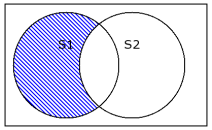
- Ability to test membership. (>>> 'SAS' in s1 → True)
- Ability to eliminate duplicate entries. (see above)
- Ability to support set operations, including union (|), intersection (&), difference (-), and symmetric difference (^).
- It is mutable.
- It can contain numbers, strings and tuples but not a list, dictionary and set.

Examples of Set Operations

```
>>> engineers  = set(['John', 'Jane', 'Jack', 'Janice'])
>>> programmers  = set(['Jack', 'Sam', 'Susan', 'Janice'])
>>> managers  = set(['Jane', 'Jack', 'Susan', 'Zack'])
>>> employees  = engineers | programmers | managers        # union
>>> employees
set(['Jack', 'Sam', 'Susan', 'Jane', 'Janice', 'John', 'Zack'])
>>> engineering_management = engineers & managers        # intersection
>>> engineering_management
set(['Jane', 'Jack'])
>>> fulltime_management = managers - engineers – programmers  # difference
>>> fulltime_management
set(['Zack'])
```

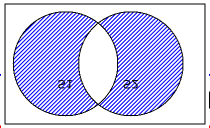






```
>>> fib = set( (1,1,2,3,5,8,13) )
>>> prime = set( (2,3,5,7,11,13) )
>>> fib | prime
set([1, 2, 3, 5, 7, 8, 11, 13])
>>> fib & prime
set([2, 3, 5, 13])
```

```
>>> fib - prime
set([8, 1])
>>> prime - fib
set([11, 7])
>>> fib ^ prime   # symmetric difference
set([1, 7, 8, 11])
```



Python Set Methods and Functions

- set.clear()
- set.pop() → object
- set.add(*new*)
- set.remove(*old*)
- set.discard()   (the same as set.remove())
- set.update(*new*)
- set.intersection_update(*new*)
- set.difference_update(*new*)
- set.symmetric_difference_update(*new*)
- set.copy() → set
- set.union(*new*) → set
- set.intersection(*new*) → set

- set.difference(*new*) → set
- set.symmetric_difference(*new*) → set
- set.issubset(*other*) → boolean
- set.issuperset(*other*) → boolean
- len(set)
- max(set)
- min(set)
- sum(set)
- any(set)
- all(set)
- sorted(set)

Examples: Using Python Set Methods and Functions

- set.add(*new*)

```
>>> fib = set( (1,1,2,3,5,8,13) )
>>> fib
set([1, 2, 3, 5, 8, 13])
>>> fib.add(21)
>>> fib
set([1, 2, 3, 5, 8, 13, 21])
```

- set.remove(*old*)

```
>>> fib.remove(21)
>>> fib
set([1, 2, 3, 5, 8, 13])
```

- set.intersection_update(*new*)

```
>>> new = {2, 3, 5, 7}
>>> new
set([2, 3, 5, 7])
>>> fib.intersection(new)
set([2, 3, 5])
>>> fib  # to see if the original set was changed or not
set([1, 2, 3, 5, 8, 13])
>>> fib.intersection_update(new)
>>> fib   # the original set was  update
set([2, 3, 5])
```

- len(set)

```
>>> len(fib)
3
```

- max(set)

```
>>> max(fib)
5
```

- sum(set)

```
>>> sum(fib)
10
```

- sorted(set)

```
>>> employees   # from previous example
set(['Jack', 'Sam', 'Susan', 'Jane',
'Janice', 'John', 'Zack'])
>>> sorted(employees)
['Jack', 'Jane', 'Janice', 'John', 'Sam',
'Susan', 'Zack']
```

- set.clear()

```
>>> fib.clear(); employees.clear()
>>> fib; employees
set([])
set([])
```

A comparison of Some Different Types of Collections in Python and Their Characteristics

- **String:** ordered, characters, immutable
- **List:** ordered, heterogeneous, mutable
- **Tuple:** ordered, heterogeneous, immutable
- **Dictionary:** unordered, key/values pairs, mutable
- **Set:** unordered, heterogeneous, mutable, unique values

Python Operators

Basic arithmetic operators

- + (addition)
- - (subtraction)
- * (multiplication)
- ** (exponentiation)
- / (division)
- // (floor division)
- % (modulus)

Assignment operators

- = (simple assignment)
- += (add AND assignment)
- -= (subtract AND assignment))
- *= (multiply AND assignment)
- **= (exponent AND assignment)
- /= (divide AND assignment)
- //= (floor division AND assignment)
- %= (modulus AND assignment)

Logical operators

- and
- or
- not

Membership operators

- in
- not in

Identity operators

- is
- is not

Comparison operators

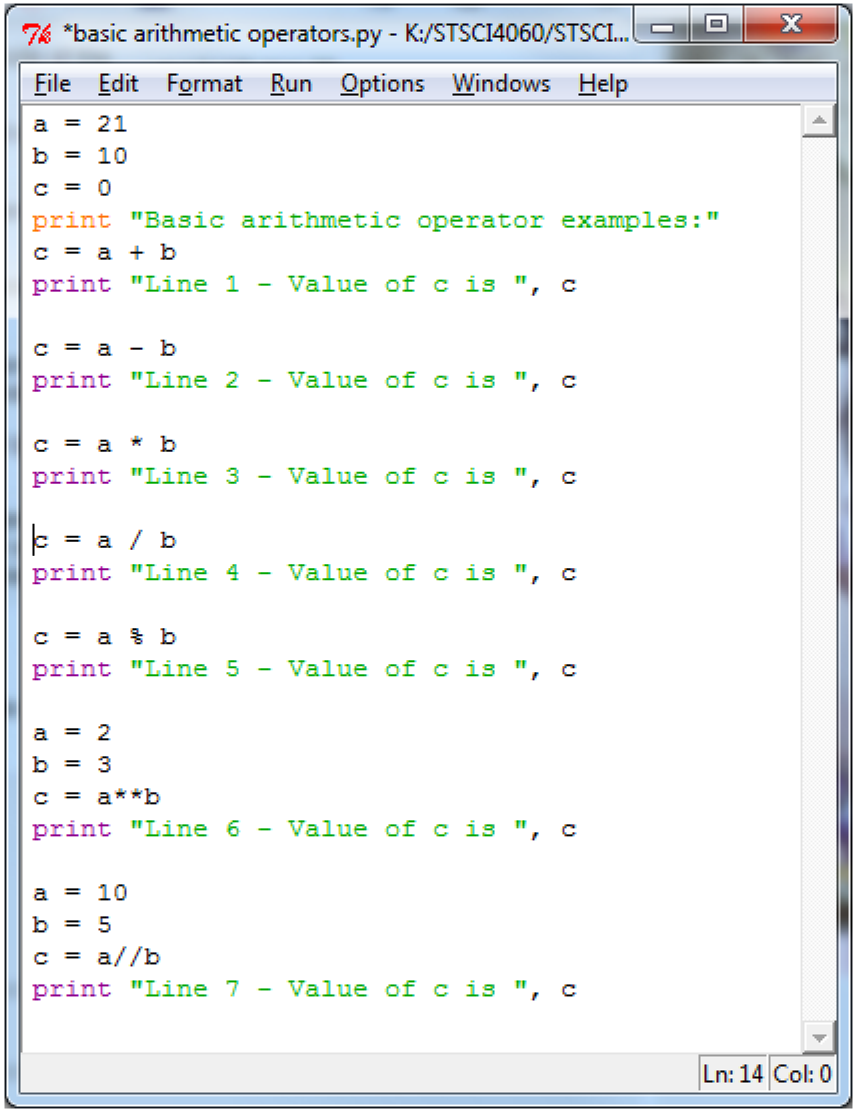
Meaning	Math symbol	Python symbol
Less than	<	<
Greater than	>	>
Less than or equal	≤	<=
Greater than or equal	≥	>=
Equals	=	==
Not equal	≠	!= or <>

Python operator precedence (highest to lowest)

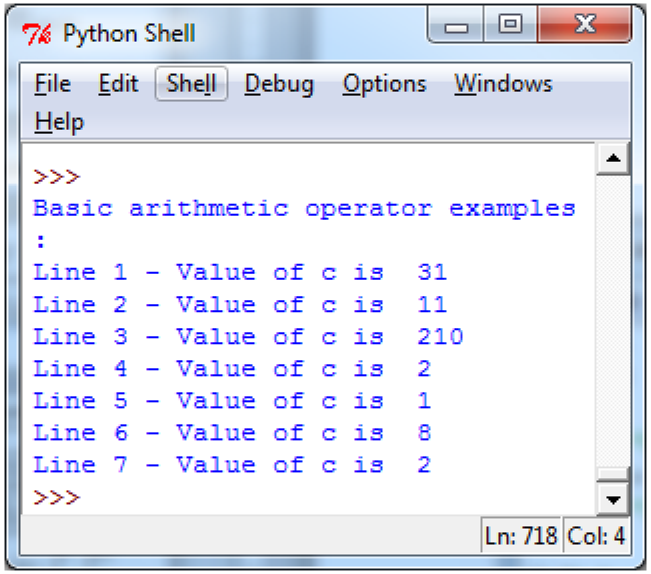
- **
- *, /, %, //
- + , -
- <=, <, >, >=
- <>, ==, !=
- =, %=, /=, //%, -=, +=, *=, **=
- is, is not
- in, not in
- not, or, and

Examples of Using Python Operators

Basic arithmetic operators


*basic arithmetic operators.py - K:/STSCI4060/STSCI...
File Edit Format Run Options Windows Help
Ln: 14 Col: 0

```
a = 21
b = 10
c = 0
print "Basic arithmetic operator examples:"
c = a + b
print "Line 1 - Value of c is ", c

c = a - b
print "Line 2 - Value of c is ", c

c = a * b
print "Line 3 - Value of c is ", c

c = a / b
print "Line 4 - Value of c is ", c

c = a % b
print "Line 5 - Value of c is ", c

a = 2
b = 3
c = a**b
print "Line 6 - Value of c is ", c

a = 10
b = 5
c = a//b
print "Line 7 - Value of c is ", c
```



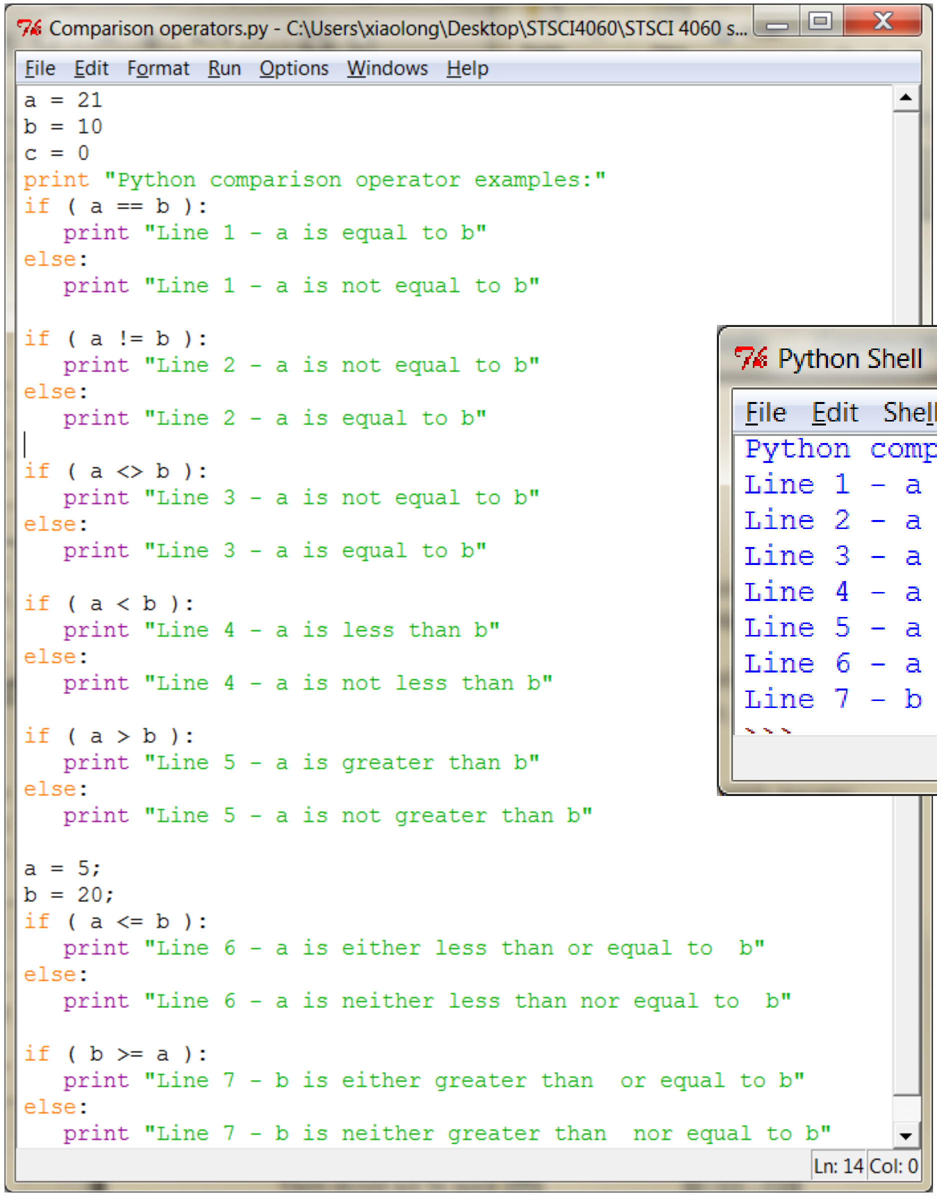
```
>>>
Basic arithmetic operator examples
:
Line 1 - Value of c is  31
Line 2 - Value of c is  11
Line 3 - Value of c is  210
Line 4 - Value of c is  2
Line 5 - Value of c is  1
Line 6 - Value of c is  8
Line 7 - Value of c is  2
>>>
```

A Brief Introduction: The if…else Statement

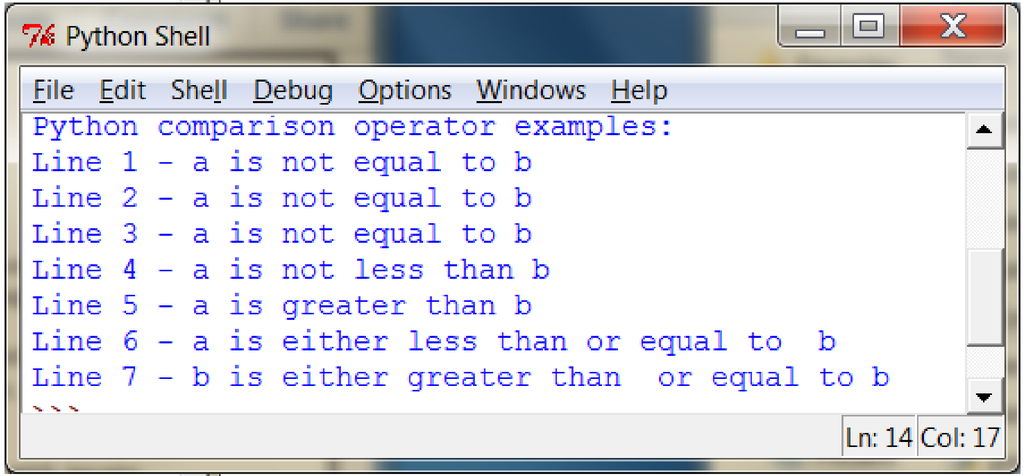
```
if condition:
        indentedStatementBlockForTrueCondition
else:
        indentedStatementBlockForFalseCondition
```

Examples of Using Python Operators

Comparison operators


Comparison operators.py - C:\Users\xiaolong\Desktop\STSCI4060\STSCI 4060 s...
File Edit Format Run Options Windows Help
Ln: 14 Col: 0

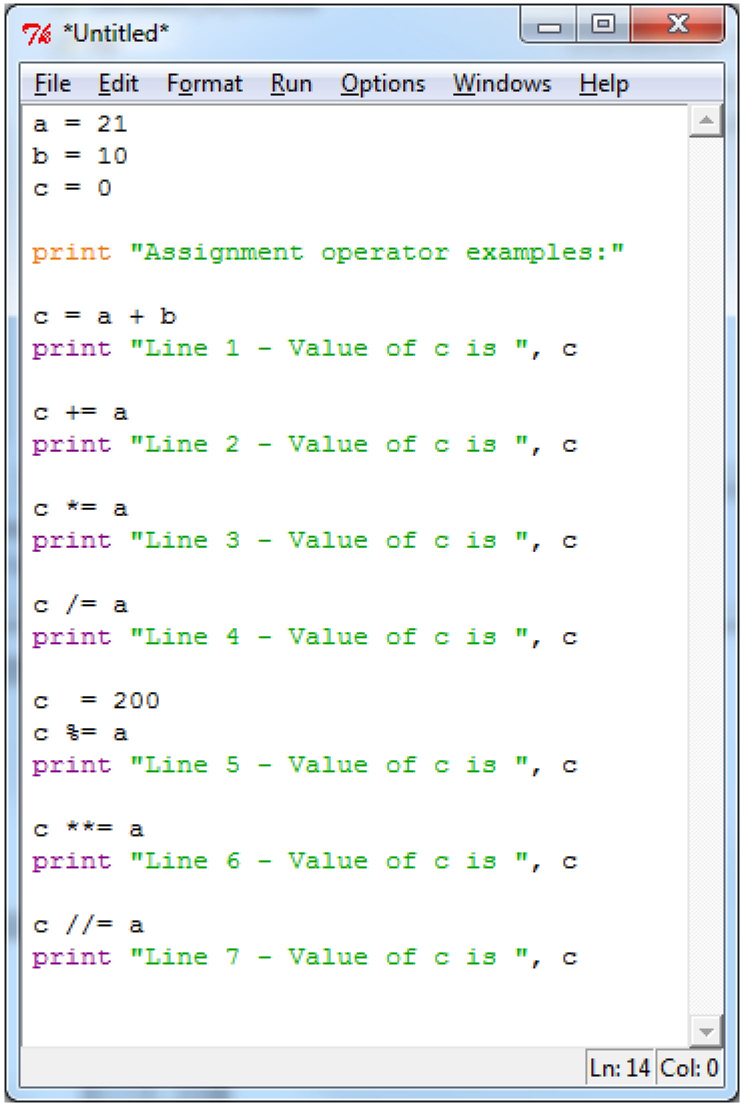
```
a = 21
b = 10
c = 0
print "Python comparison operator examples:"
if ( a == b ):
   print "Line 1 - a is equal to b"
else:
   print "Line 1 - a is not equal to b"

if ( a != b ):
   print "Line 2 - a is not equal to b"
else:
   print "Line 2 - a is equal to b"

if ( a <> b ):
   print "Line 3 - a is not equal to b"
else:
   print "Line 3 - a is equal to b"

if ( a < b ):
   print "Line 4 - a is less than b"
else:
   print "Line 4 - a is not less than b"

if ( a > b ):
   print "Line 5 - a is greater than b"
else:
   print "Line 5 - a is not greater than b"

a = 5;
b = 20;
if ( a <= b ):
   print "Line 6 - a is either less than or equal to  b"
else:
   print "Line 6 - a is neither less than nor equal to  b"

if ( b >= a ):
   print "Line 7 - b is either greater than  or equal to b"
else:
   print "Line 7 - b is neither greater than  nor equal to b"
```



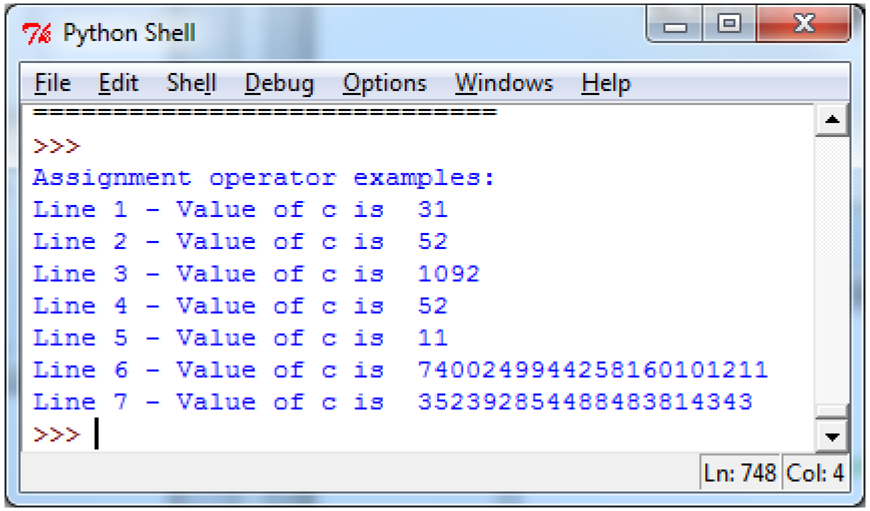
```
Python comparison operator examples:
Line 1 - a is not equal to b
Line 2 - a is not equal to b
Line 3 - a is not equal to b
Line 4 - a is not less than b
Line 5 - a is greater than b
Line 6 - a is either less than or equal to  b
Line 7 - b is either greater than  or equal to b
>>>
```

Examples of Using Python Operators

Assignment operators



```
a = 21
b = 10
c = 0

print "Assignment operator examples:"

c = a + b
print "Line 1 - Value of c is ", c

c += a
print "Line 2 - Value of c is ", c

c *= a
print "Line 3 - Value of c is ", c

c /= a
print "Line 4 - Value of c is ", c

c  = 200
c %= a
print "Line 5 - Value of c is ", c

c **= a
print "Line 6 - Value of c is ", c

c //= a
print "Line 7 - Value of c is ", c
```



```
==============================
>>>
Assignment operator examples:
Line 1 - Value of c is  31
Line 2 - Value of c is  52
Line 3 - Value of c is  1092
Line 4 - Value of c is  52
Line 5 - Value of c is  11
Line 6 - Value of c is  7400249944258160101211
Line 7 - Value of c is  352392854488483814343
>>>
```

Examples of Using Python Operators

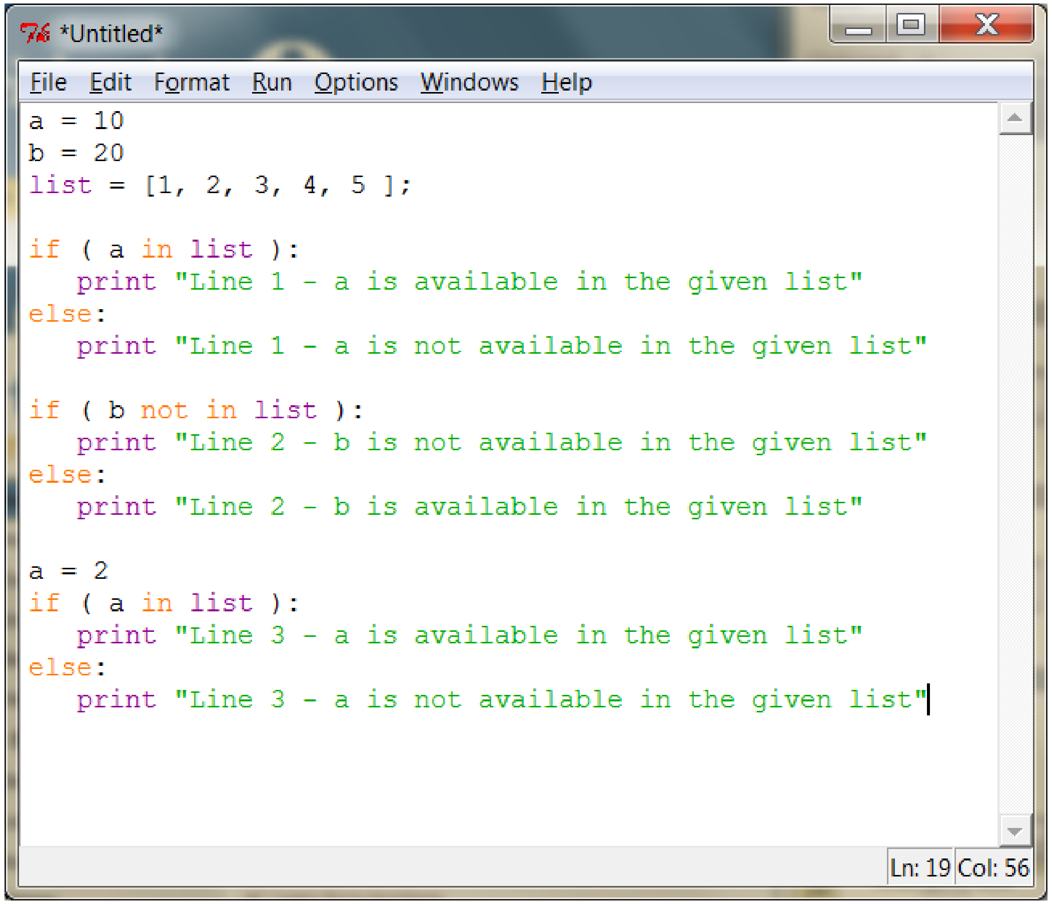
Logical operators

```
a = 21
b = 10


print "Logical operator examples:"

if ( a and b ):
   print "Line 1 - a and b are true"
else:
   print "Line 1 - Either a is not true or b is not true"

if ( a or b ):
   print "Line 2 - Either a is true or b is true or both are true"
else:
   print "Line 2 - Neither a is true nor b is true"


a = 0
if ( a and b ):
   print "Line 3 - a and b are true"
else:
   print "Line 3 - Either a is not true or b is not true"

if ( a or b ):
   print "Line 4 - Either a is true or b is true or both are true"
else:
   print "Line 4 - Neither a is true nor b is true"

if not( a and b ):
   print "Line 5 - Either a or b or both are not true"
else:
   print "Line 5 - a and b are true"
```

```
Logical operator examples:
Line 1 - a and b are true
Line 2 - Either a is true or b is true or both are true
Line 3 - Either a is not true or b is not true
Line 4 - Either a is true or b is true or both are true
Line 5 - Either a or b or both are not true
```

Examples of Using Python Operators

Membership operators



```
a = 10
b = 20
list = [1, 2, 3, 4, 5 ];

if ( a in list ):
   print "Line 1 - a is available in the given list"
else:
   print "Line 1 - a is not available in the given list"

if ( b not in list ):
   print "Line 2 - b is not available in the given list"
else:
   print "Line 2 - b is available in the given list"

a = 2
if ( a in list ):
   print "Line 3 - a is available in the given list"
else:
   print "Line 3 - a is not available in the given list"
```

```
Line 1 - a is not available in the given list
Line 2 - b is not available in the given list
Line 3 - a is available in the given list
```

Examples of Using Python Operators

Identity operators

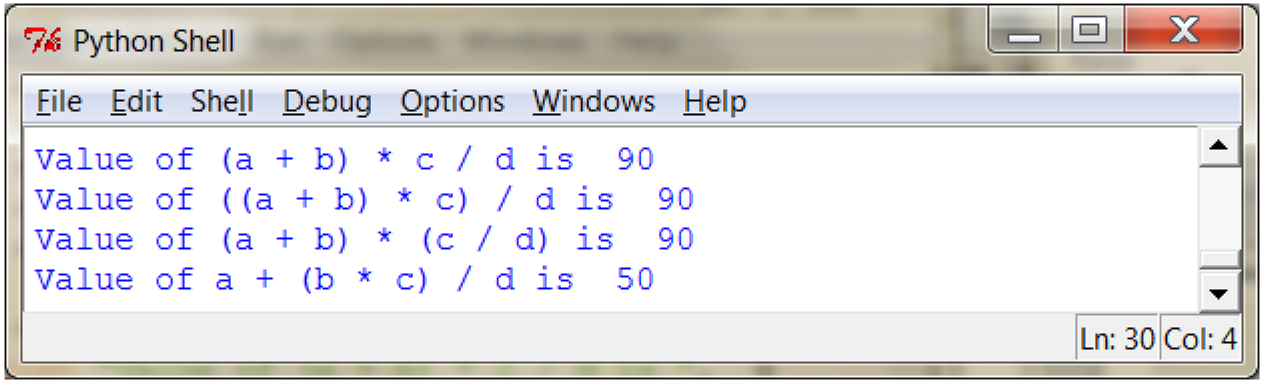
```
a = 20
b = 20

if ( a is b ):
   print "Line 1 - a and b have same identity"
else:
   print "Line 1 - a and b do not have same identity"

if ( id(a) == id(b) ):
   print "Line 2 - a and b have same identity"
else:
   print "Line 2 - a and b do not have same identity"

b = 30
if ( a is b ):
   print "Line 3 - a and b have same identity"
else:
   print "Line 3 - a and b do not have same identity"

if ( a is not b ):
   print "Line 4 - a and b do not have same identity"
else:
   print "Line 4 - a and b have same identity"
```

```
Line 1 - a and b have same identity
Line 2 - a and b have same identity
Line 3 - a and b do not have same identity
Line 4 - a and b do not have same identity
```

Examples of Using Python Operators

Operator precedence

```
a = 20
b = 10
c = 15
d = 5
e = 0

e = (a + b) * c / d        #( 30 * 15 ) / 5
print "Value of (a + b) * c / d is ",  e

e = ((a + b) * c) / d      # (30 * 15 ) / 5
print "Value of ((a + b) * c) / d is ",  e

e = (a + b) * (c / d);     # (30) * (15/5)
print "Value of (a + b) * (c / d) is ",  e

e = a + (b * c) / d;       #  20 + (150/5)
print "Value of a + (b * c) / d is ",  e
```



```
Value of (a + b) * c / d is  90
Value of ((a + b) * c) / d is  90
Value of (a + b) * (c / d) is  90
Value of a + (b * c) / d is  50
```