

004 Python Data Type

Introduction

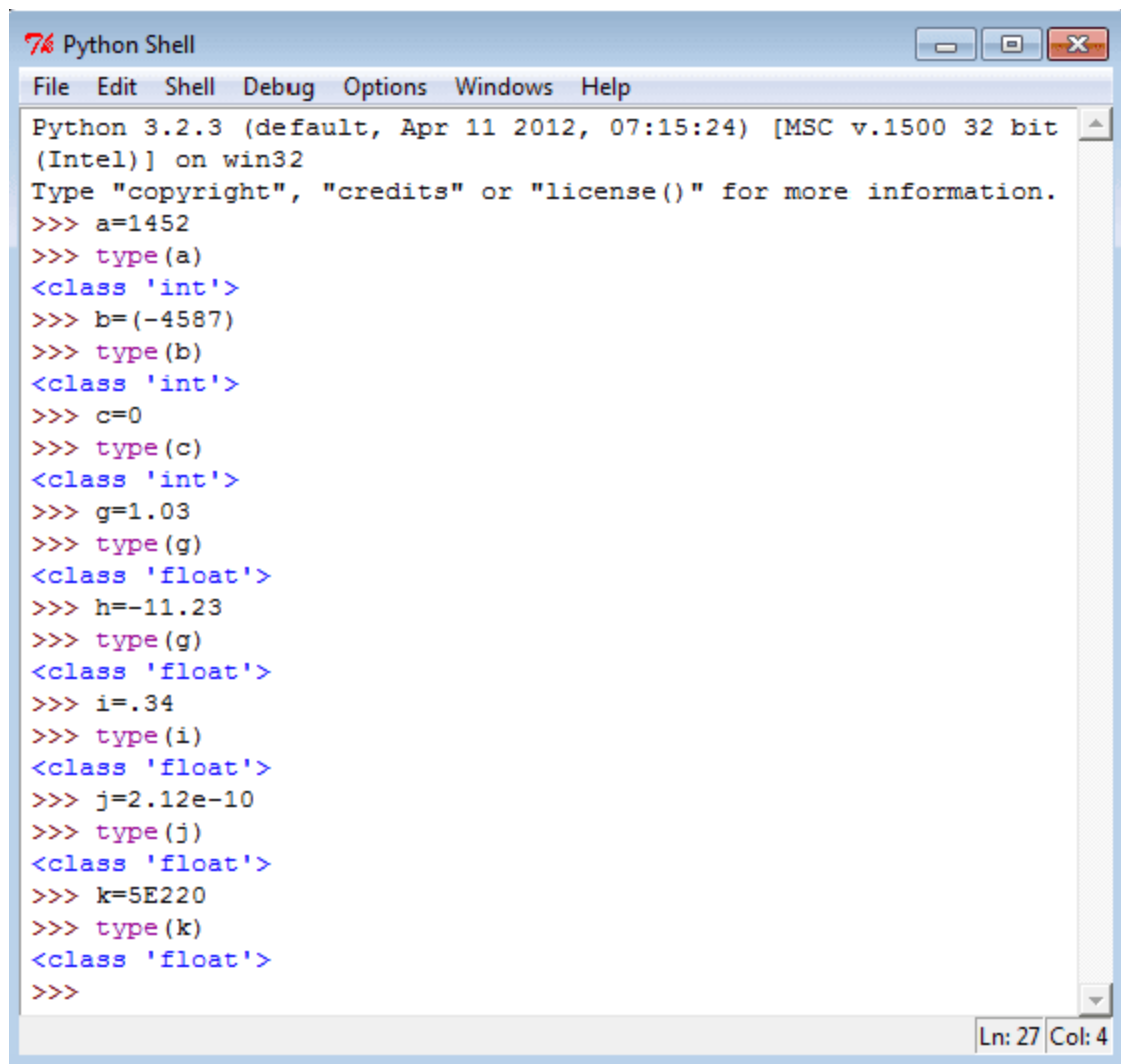
Type represents the kind of value and determines how the value can be used. All data values in Python are encapsulated in relevant object classes. Everything in Python is an object and every object has an identity, a type, and a value. Like another object-oriented language such as Java or C++, there are several data types which are built into Python. Extension modules which are written in C, Java, or other languages can define additional types.

To determine a variable's type in Python you can use the `type()` function. The value of some objects can be changed. Objects whose value can be changed are called mutable and objects whose value is unchangeable (once they are created) are called immutable. Here are the details of Python data types

Numbers

Numbers are created by numeric literals. Numeric objects are immutable, which means when an object is created its value cannot be changed.

Python has three distinct numeric types: integers, floating point numbers, and complex numbers. Integers represent negative and positive integers without fractional parts whereas floating point numbers represents negative and positive numbers with fractional parts. In addition, Booleans are a subtype of plain integers. See the following statements in Python shell.



The screenshot shows a Python Shell window titled "Python Shell" with a menu bar (File, Edit, Shell, Debug, Options, Windows, Help). The window displays the following text:

```
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit  
(Intel)] on win32  
Type "copyright", "credits" or "license()" for more information.  
>>> a=1452  
>>> type(a)  
<class 'int'>  
>>> b=(-4587)  
>>> type(b)  
<class 'int'>  
>>> c=0  
>>> type(c)  
<class 'int'>  
>>> g=1.03  
>>> type(g)  
<class 'float'>  
>>> h=-11.23  
>>> type(g)  
<class 'float'>  
>>> i=.34  
>>> type(i)  
<class 'float'>  
>>> j=2.12e-10  
>>> type(j)  
<class 'float'>  
>>> k=5E220  
>>> type(k)  
<class 'float'>  
>>>
```

The status bar at the bottom right indicates "Ln: 27 Col: 4".

Floats and Integers

```
>>> x = 8  
  
>>> y = 7  
  
>>> x+y  
  
>>> x-y  
  
>>> x/y  
  
>>> x*y
```

Output:

```
15
1
1.1428571428571428
56
```

Exponent

```
>>> 4**3
```

```
>>> 3**4
```

Output:

```
64
81
```

inter division

```
>>> 12/3
```

```
>>> 64//4
```

```
>>> 15//3
```

Output:

```
4.0
16
5
```

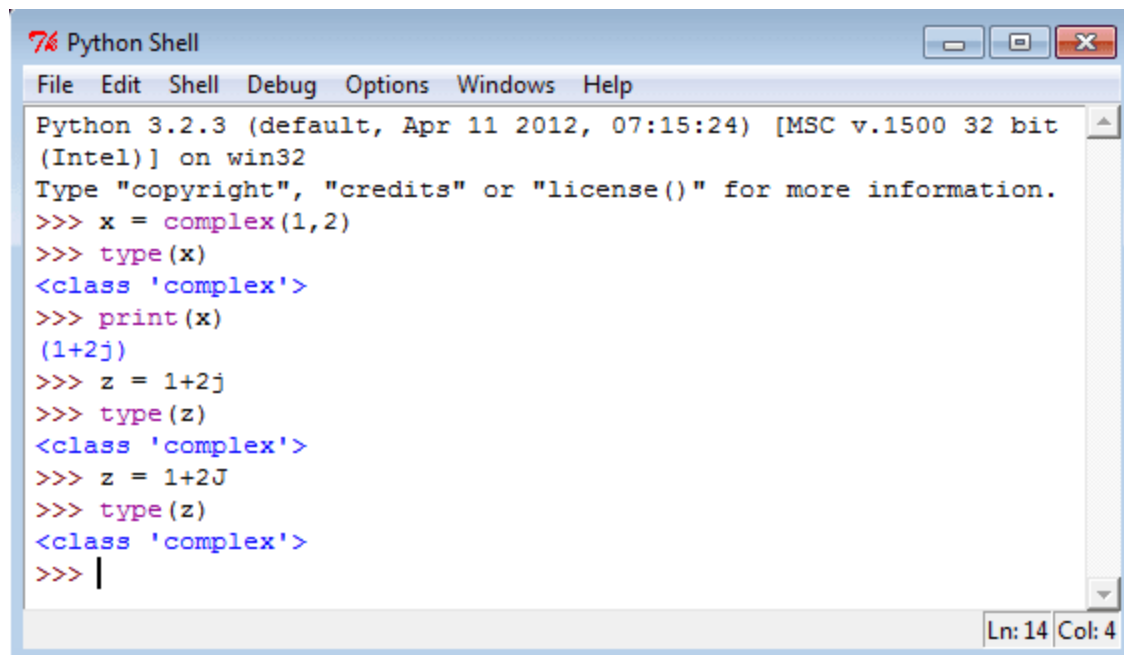
Remainder

```
>>> 15 % 4
```

Output:

```
3
```

Mathematically, a complex number (generally used in engineering) is a number of the form $A+Bi$ where i is the imaginary number. Complex numbers have a real and imaginary part. Python supports complex numbers either by specifying the number in (real + imagJ) or (real + imagj) form or using a built-in method `complex(x, y)`. See the following statements in Python Shell.



A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following code and output:

```
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x = complex(1,2)
>>> type(x)
<class 'complex'>
>>> print(x)
(1+2j)
>>> z = 1+2j
>>> type(z)
<class 'complex'>
>>> z = 1+2J
>>> type(z)
<class 'complex'>
>>> |
```

The status bar at the bottom right indicates "Ln: 14 Col: 4".

Self-assignment

```
>>> count = 0
>>> count +=2
>>> count
```

Output:

```
2
>>> count -=2
>>> count
```

Output:

```
0
>>> count +=2
>>> count *=4
```

```
>>> count
```

Output:

```
8
```

```
>>> count **=4
```

```
>>> count
```

Output:

```
4096
```

```
>>> count /=4
```

```
>>> count
```

Output:

```
1024.0
```

```
>>> count //=4
```

```
>>> count
```

Output:

```
256.0
```

```
>>> 15.0 //2
```

Output:

```
7.0
```

Order of operations

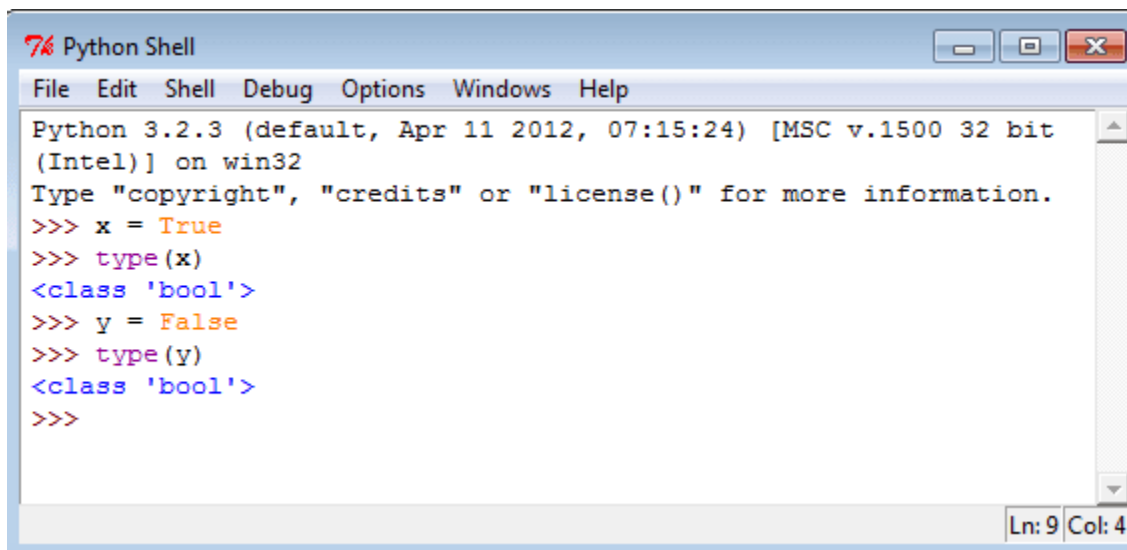
```
>>> (3+12) / 3
```

```
>>> 15 / (3 + 2)
```

Output:

Boolean (bool)

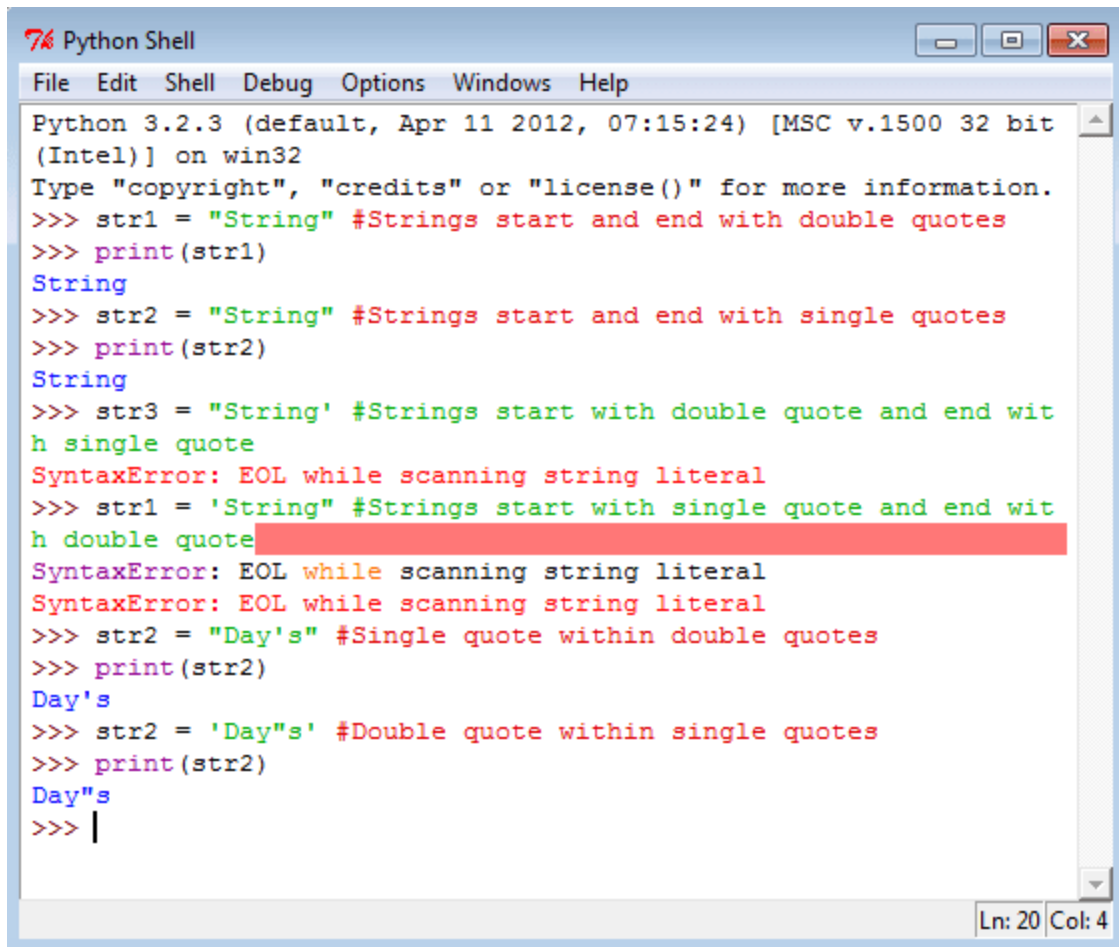
The simplest build-in type in Python is the bool type, it represents the truth values False and True. See the following statements in Python shell.

A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following text: "Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit (Intel)] on win32", "Type 'copyright', 'credits' or 'license()' for more information.", and a series of interactive commands and outputs: ">>> x = True", ">>> type(x)", "<class 'bool'>", ">>> y = False", ">>> type(y)", "<class 'bool'>", and ">>>". The status bar at the bottom right indicates "Ln: 9 Col: 4".

```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x = True
>>> type(x)
<class 'bool'>
>>> y = False
>>> type(y)
<class 'bool'>
>>>
Ln: 9 Col: 4
```

Strings

In Python, a string type object is a sequence (left-to- right order) of characters. Strings start and end with single or double quotes Python strings are immutable. Single and double quoted strings are same and you can use a single quote within a string when it is surrounded by double quote and vice versa. Declaring a string is simple, see the following statements.

A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following code and output:

```
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> str1 = "String" #Strings start and end with double quotes
>>> print(str1)
String
>>> str2 = "String" #Strings start and end with single quotes
>>> print(str2)
String
>>> str3 = "String' #Strings start with double quote and end with single quote
SyntaxError: EOL while scanning string literal
>>> str1 = 'String" #Strings start with single quote and end with double quote
SyntaxError: EOL while scanning string literal
SyntaxError: EOL while scanning string literal
>>> str2 = "Day's" #Single quote within double quotes
>>> print(str2)
Day's
>>> str2 = 'Day"s' #Double quote within single quotes
>>> print(str2)
Day"s
>>> |
```

The status bar at the bottom right shows "Ln: 20 Col: 4".

Special characters in strings

The backslash (\) character is used to introduce a special character. See the following table.

Escape sequence	Meaning
\n	Newline
\t	Horizontal Tab

\\	Backslash
\'	Single Quote
\"	Double Quote

See the following statements on special characters.

```

Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("The is a backslash (\\) mark.")
The is a backslash (\) mark.
>>> print("This is tab \t key")
This is tab      key
>>> print("These are \'single quotes\'")
These are 'single quotes'
>>> print("These are \"double quotes\"")
These are "double quotes"
>>> print("This is a new line\nNew line")
This is a new line
New line
>>>

```

String indices and accessing string elements

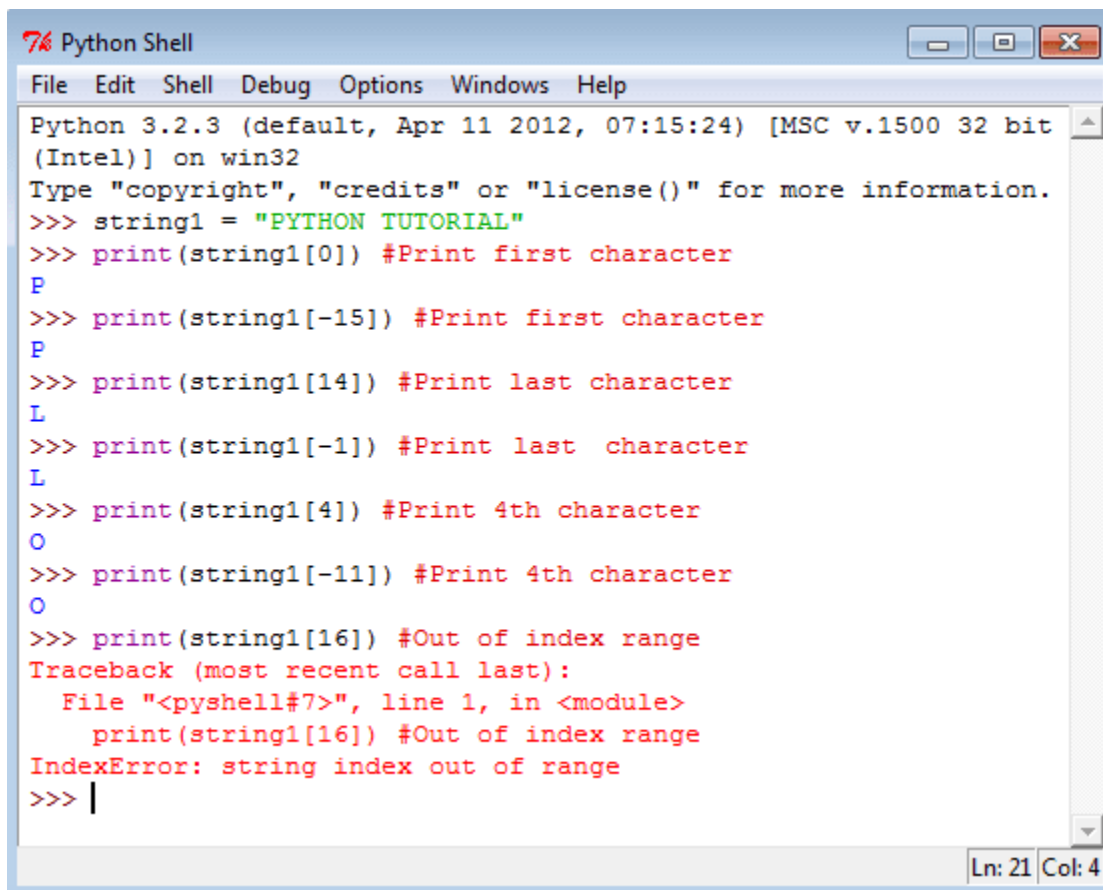
Strings are arrays of characters and elements of an array can be accessed using indexing. Indices start with 0 from left side and -1 when starting from right side.

```
string1="PYTHON TUTORIAL"
```

Character	P	Y	T	H	O	N		T	U	T	O	R	I	A	L
-----------	---	---	---	---	---	---	--	---	---	---	---	---	---	---	---

Index (from left)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Index (from right)	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

See the following statements to access single character from various positions.



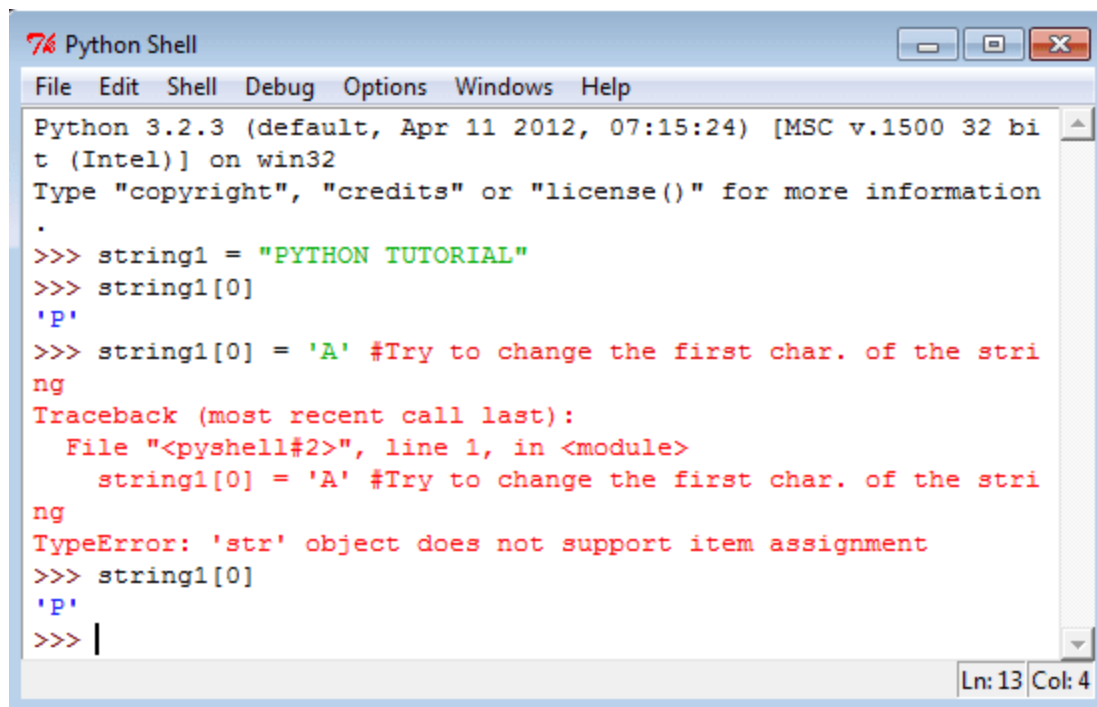
```

Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> string1 = "PYTHON TUTORIAL"
>>> print(string1[0]) #Print first character
P
>>> print(string1[-15]) #Print first character
P
>>> print(string1[14]) #Print last character
L
>>> print(string1[-1]) #Print last character
L
>>> print(string1[4]) #Print 4th character
O
>>> print(string1[-11]) #Print 4th character
O
>>> print(string1[16]) #Out of index range
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    print(string1[16]) #Out of index range
IndexError: string index out of range
>>> |

```

Strings are immutable

Strings are immutable character sets. Once a string is generated, you can not change any character within the string. See the following statements.

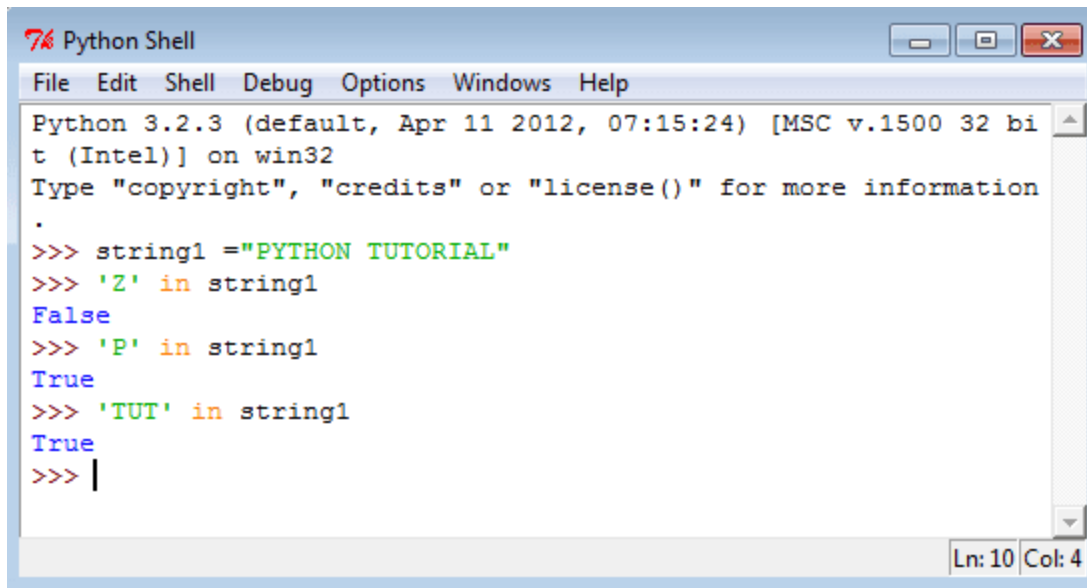


```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information
.
>>> string1 = "PYTHON TUTORIAL"
>>> string1[0]
'P'
>>> string1[0] = 'A' #Try to change the first char. of the string
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    string1[0] = 'A' #Try to change the first char. of the string
TypeError: 'str' object does not support item assignment
>>> string1[0]
'P'
>>> |
```

Ln: 13 Col: 4

'in' operator in Strings

The 'in' operator is used to check whether a character or a substring is present in a string or not. The expression returns a Boolean value. See the following statements.

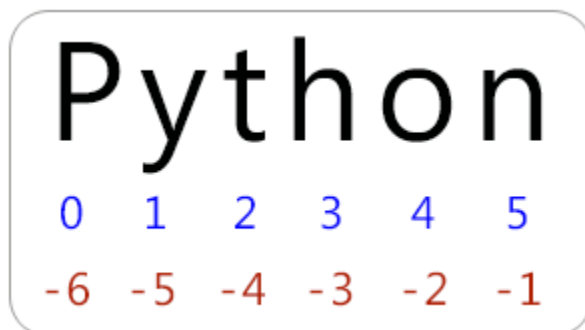
A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following code and output:

```
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bi
t (Intel)] on win32
Type "copyright", "credits" or "license()" for more information
.>>> string1 = "PYTHON TUTORIAL"
>>> 'Z' in string1
False
>>> 'P' in string1
True
>>> 'TUT' in string1
True
>>> |
```

The status bar at the bottom right indicates "Ln: 10 Col: 4".

String Slicing

To cut a substring from a string is called string slicing. Here two indices are used separated by a colon (:). A slice 3:7 means indices characters of 3rd, 4th, 5th and 6th positions. The second integer index i.e. 7 is not included. You can use negative indices for slicing. See the following statements.



Conversion

```
>>> float("4.5")
>>> int("25")
```

```
>>> int(5.625)
>>> float(6)
>>> int(True)
>>> float(False)
>>> str(True)
>>> bool(0)
>>> bool('Hello world')
>>> bool(223.5)
```

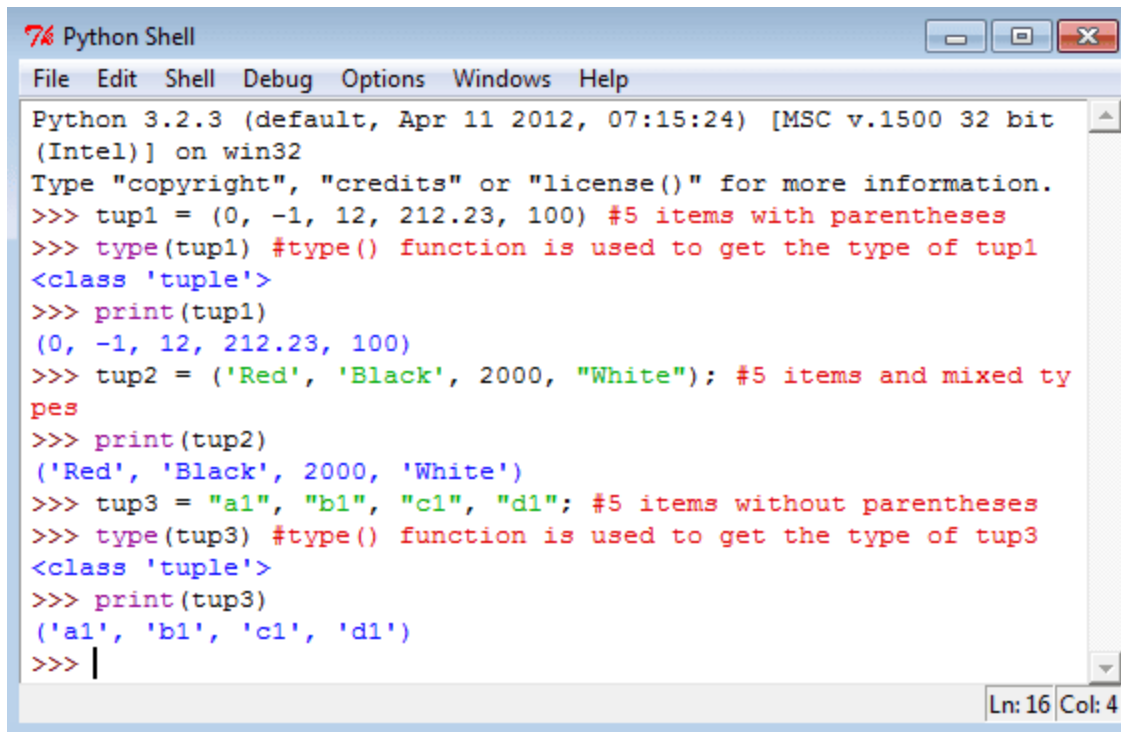
Output:

```
4.5
25
5
6.0
1
0.0
'True'
False
True
True
```

Tuples

A tuple is a container which holds a series of comma-separated values (items or elements) between parentheses. Tuples are immutable (i.e. you cannot change its content once created) and can hold mix data types.

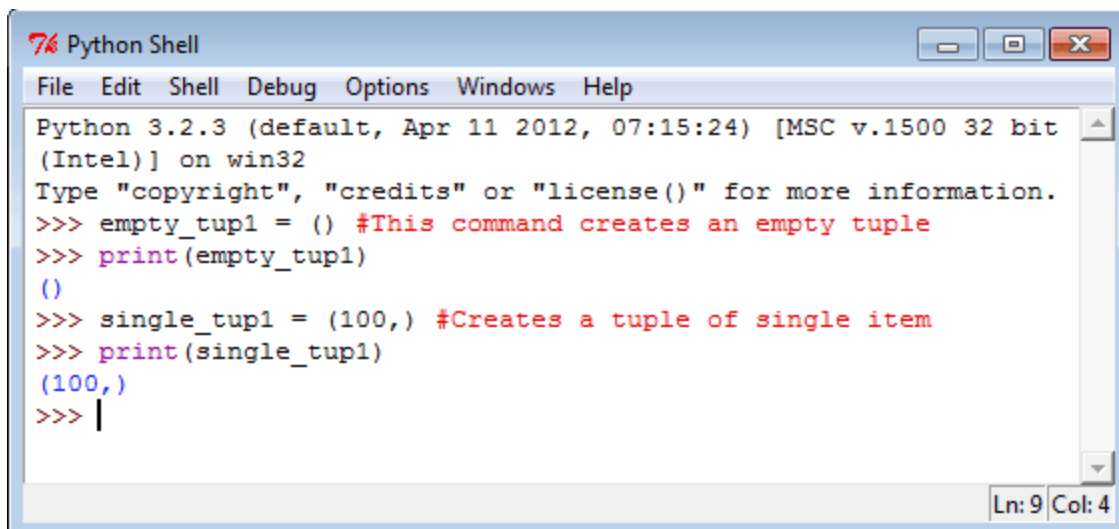
Creating Tuples

A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following code:

```
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> tup1 = (0, -1, 12, 212.23, 100) #5 items with parentheses
>>> type(tup1) #type() function is used to get the type of tup1
<class 'tuple'>
>>> print(tup1)
(0, -1, 12, 212.23, 100)
>>> tup2 = ('Red', 'Black', 2000, "White"); #5 items and mixed ty
pes
>>> print(tup2)
('Red', 'Black', 2000, 'White')
>>> tup3 = "a1", "b1", "c1", "d1"; #5 items without parentheses
>>> type(tup3) #type() function is used to get the type of tup3
<class 'tuple'>
>>> print(tup3)
('a1', 'b1', 'c1', 'd1')
>>> |
```

The status bar at the bottom right shows "Ln: 16 Col: 4".

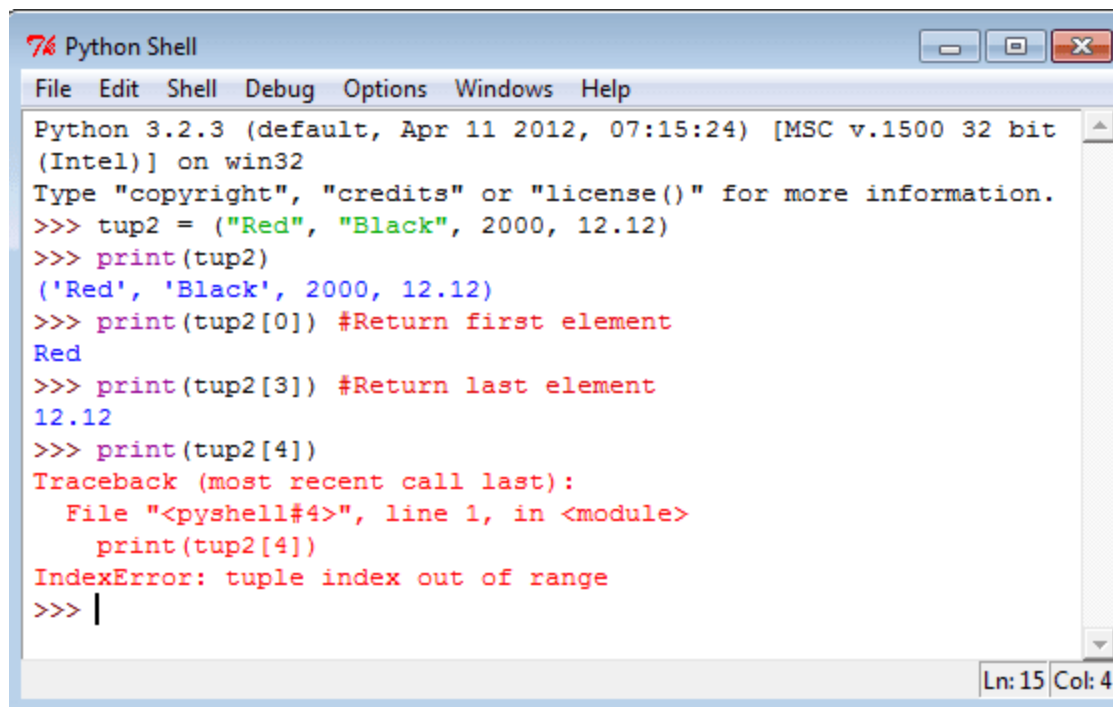
To create an empty tuple or create a tuple with single element use the following commands.

A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following code:

```
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> empty_tup1 = () #This command creates an empty tuple
>>> print(empty_tup1)
()
>>> single_tup1 = (100,) #Creates a tuple of single item
>>> print(single_tup1)
(100,)
>>> |
```

The status bar at the bottom right shows "Ln: 9 Col: 4".

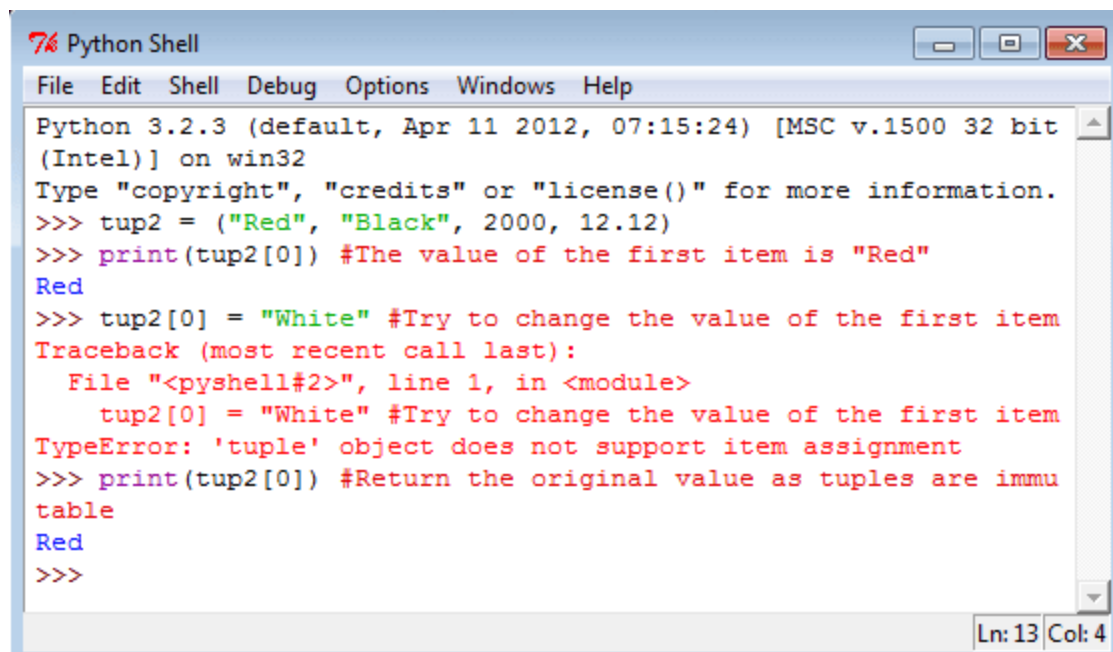
Elements of a tuple are indexed like other sequences. The tuple indices start at 0. See the following statements.

A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following code and output:

```
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> tup2 = ("Red", "Black", 2000, 12.12)
>>> print(tup2)
('Red', 'Black', 2000, 12.12)
>>> print(tup2[0]) #Return first element
Red
>>> print(tup2[3]) #Return last element
12.12
>>> print(tup2[4])
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    print(tup2[4])
IndexError: tuple index out of range
>>> |
```

The status bar at the bottom right shows "Ln: 15 Col: 4".

Tuples are immutable which means it's items values are unchangeable. See the following statements.

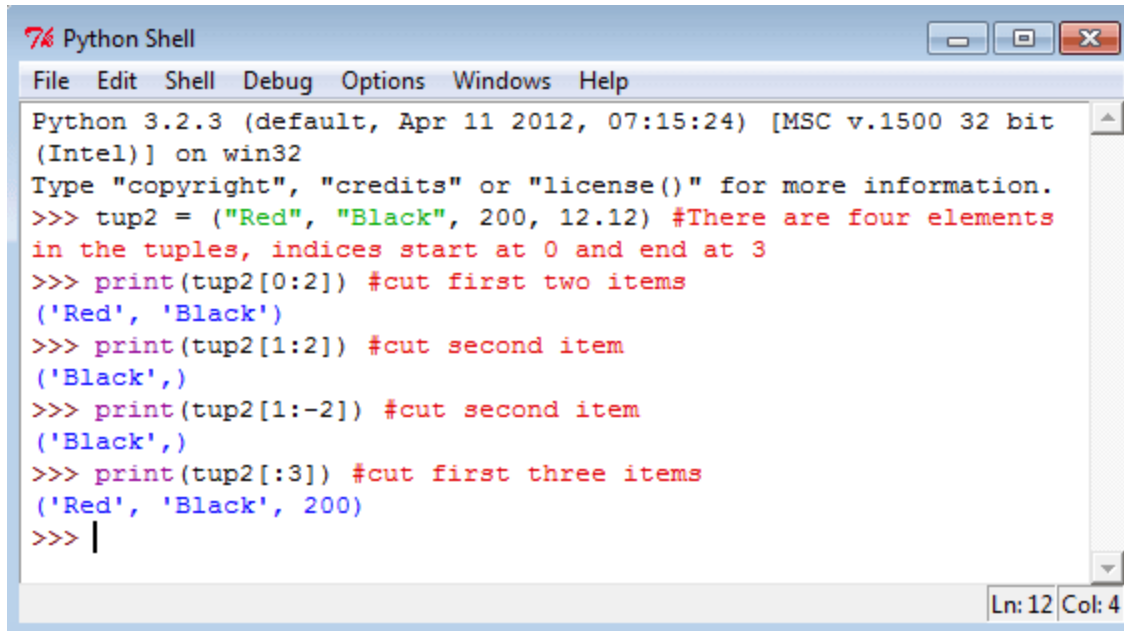
A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following code and output:

```
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> tup2 = ("Red", "Black", 2000, 12.12)
>>> print(tup2[0]) #The value of the first item is "Red"
Red
>>> tup2[0] = "White" #Try to change the value of the first item
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    tup2[0] = "White" #Try to change the value of the first item
TypeError: 'tuple' object does not support item assignment
>>> print(tup2[0]) #Return the original value as tuples are immutable
Red
>>>
```

The status bar at the bottom right shows "Ln: 13 Col: 4".

Slicing a tuple

Like other sequences like strings, tuples can be sliced. Slicing a tuple creates a new tuple but it does not change the original tuple.

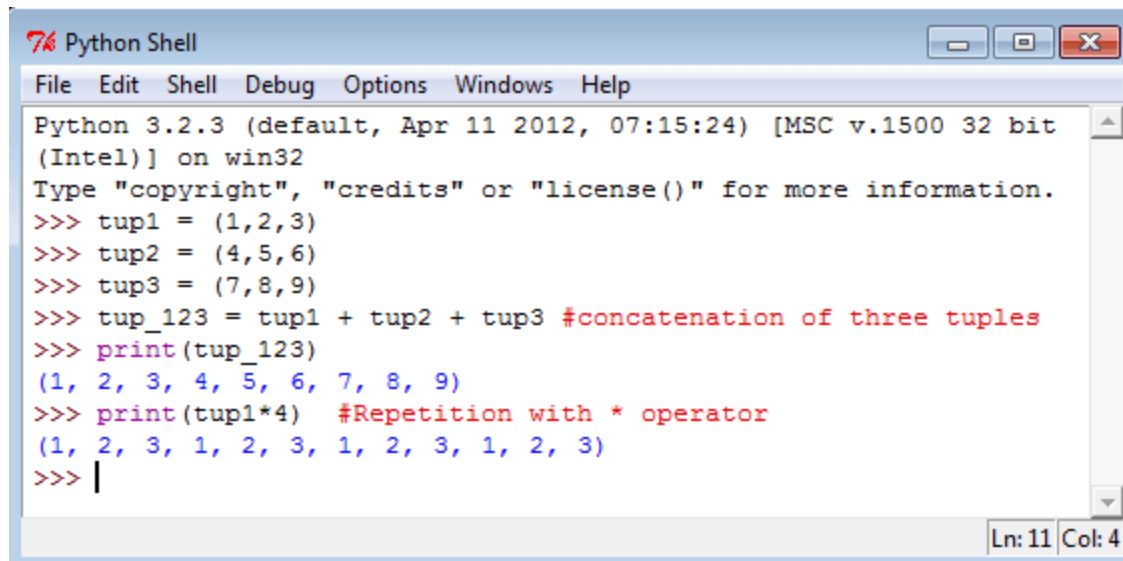
A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following code and output:

```
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> tup2 = ("Red", "Black", 200, 12.12) #There are four elements
in the tuples, indices start at 0 and end at 3
>>> print(tup2[0:2]) #cut first two items
('Red', 'Black')
>>> print(tup2[1:2]) #cut second item
('Black',)
>>> print(tup2[1:-2]) #cut second item
('Black',)
>>> print(tup2[:3]) #cut first three items
('Red', 'Black', 200)
>>> |
```

The status bar at the bottom right indicates "Ln: 12 Col: 4".

Using + and * operators in Tuples

Use + operator to create a new tuple that is a concatenation of tuples and use * operator to repeat a tuple. See the following statements.

A screenshot of a Python Shell window. The title bar says "Python Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following code:

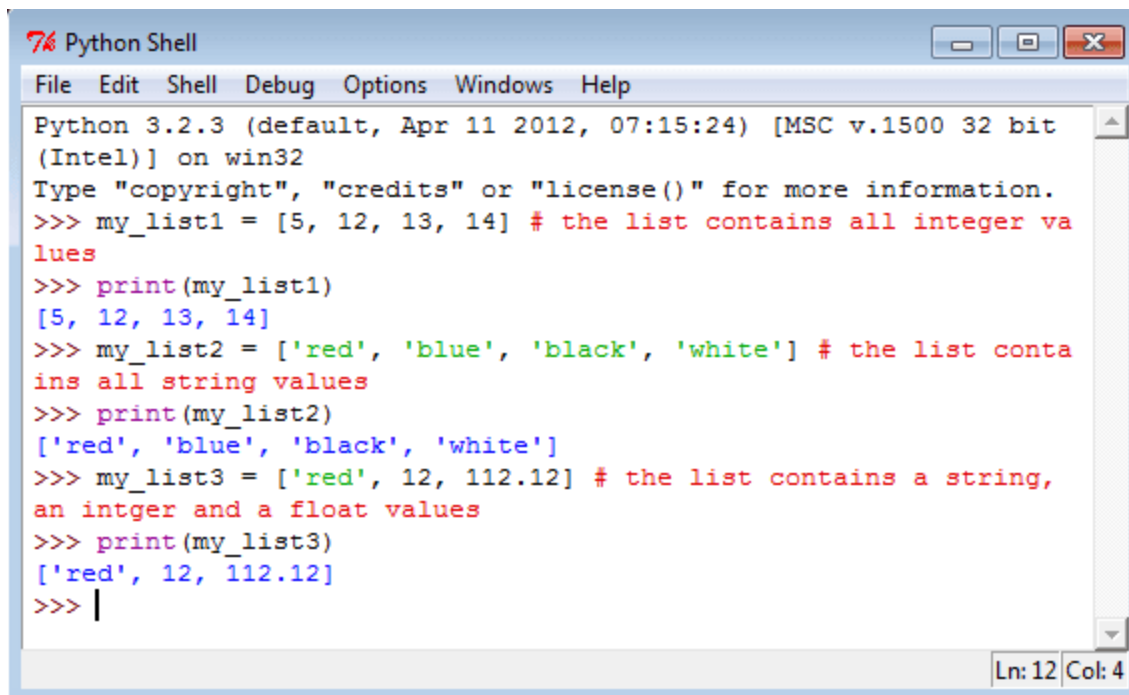
```
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> tup1 = (1,2,3)
>>> tup2 = (4,5,6)
>>> tup3 = (7,8,9)
>>> tup_123 = tup1 + tup2 + tup3 #concatenation of three tuples
>>> print(tup_123)
(1, 2, 3, 4, 5, 6, 7, 8, 9)
>>> print(tup1*4) #Repetition with * operator
(1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3)
>>> |
```

The status bar at the bottom right shows "Ln: 11 Col: 4".

Lists

A list is a container which holds comma-separated values (items or elements) between square brackets where items or elements need not all have the same type.

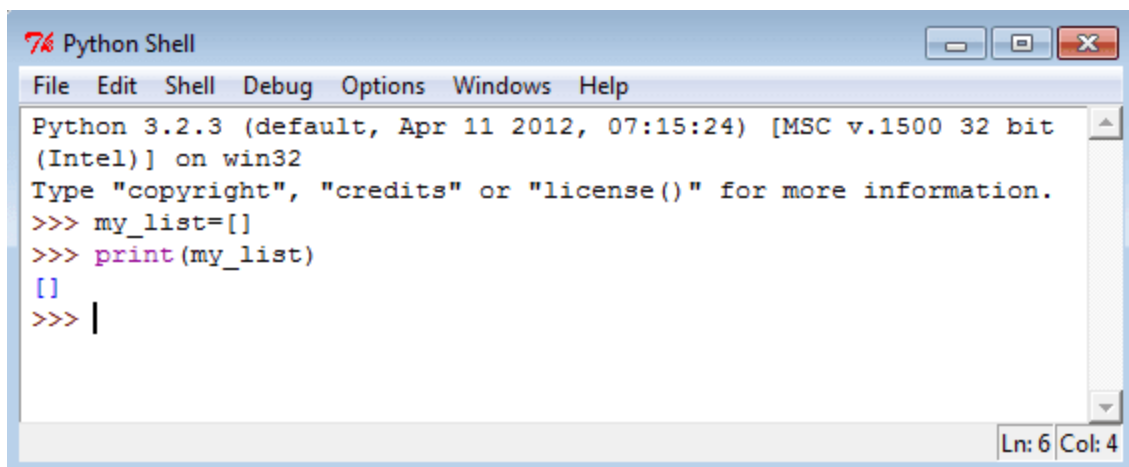
Creating Lists

A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following code:

```
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> my_list1 = [5, 12, 13, 14] # the list contains all integer values
>>> print(my_list1)
[5, 12, 13, 14]
>>> my_list2 = ['red', 'blue', 'black', 'white'] # the list contains all string values
>>> print(my_list2)
['red', 'blue', 'black', 'white']
>>> my_list3 = ['red', 12, 112.12] # the list contains a string, an integer and a float values
>>> print(my_list3)
['red', 12, 112.12]
>>> |
```

The status bar at the bottom right shows "Ln: 12 Col: 4".

A list without any element is called an empty list. See the following statements.

A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following code:

```
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> my_list=[]
>>> print(my_list)
[]
>>> |
```

The status bar at the bottom right shows "Ln: 6 Col: 4".

List indices

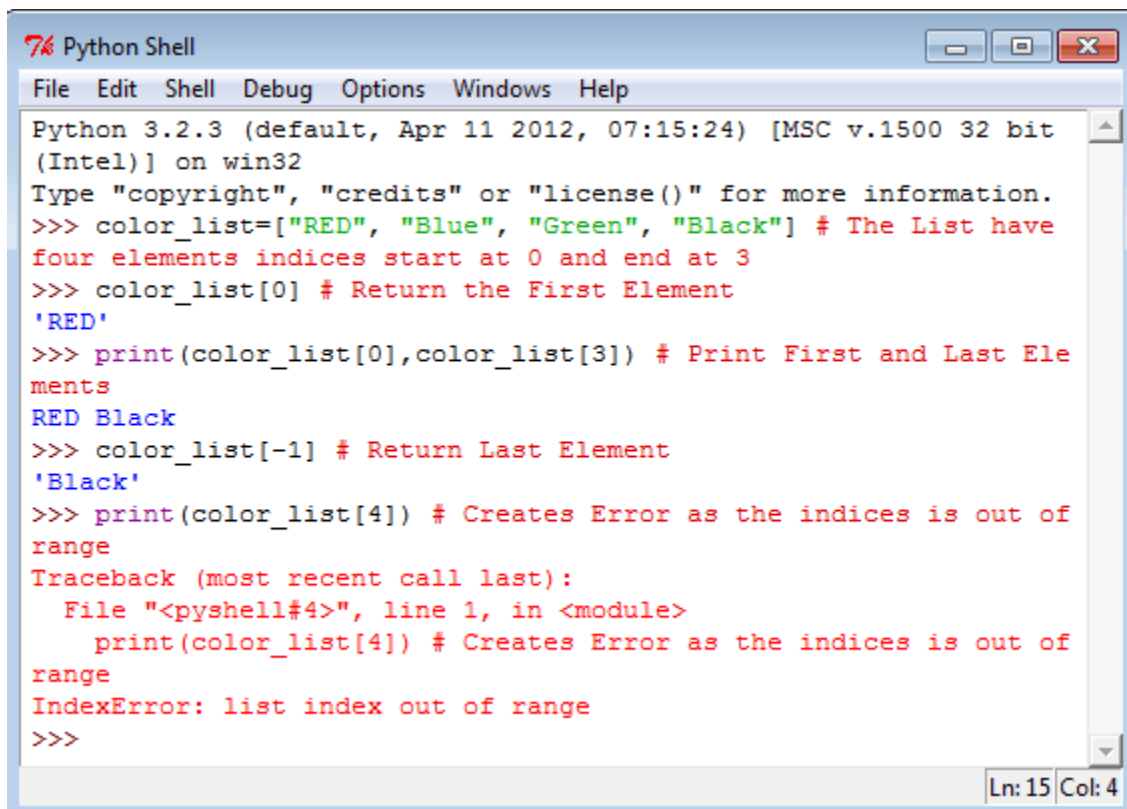
List indices work the same way as string indices, list indices start at 0. If an index has a positive value it counts from the beginning and similarly it counts backward if the index has a negative value. As positive integers are used to index from the

left end and negative integers are used to index from the right end, so every item of a list gives two alternatives indices. Let create a list called color_list with four items.

```
color_list=["RED", "Blue", "Green", "Black"]
```

Item	RED	Blue	Green	Black
Index (from left)	0	1	2	3
Index (from right)	-4	-3	-2	-1

If you give any index value which is out of range then interpreter creates an error message. See the following statements.

A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following code and output:

```
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> color_list=["RED", "Blue", "Green", "Black"] # The List have four elements indices start at 0 and end at 3
>>> color_list[0] # Return the First Element
'RED'
>>> print(color_list[0],color_list[3]) # Print First and Last Elements
RED Black
>>> color_list[-1] # Return Last Element
'Black'
>>> print(color_list[4]) # Creates Error as the indices is out of range
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    print(color_list[4]) # Creates Error as the indices is out of range
IndexError: list index out of range
>>>
```

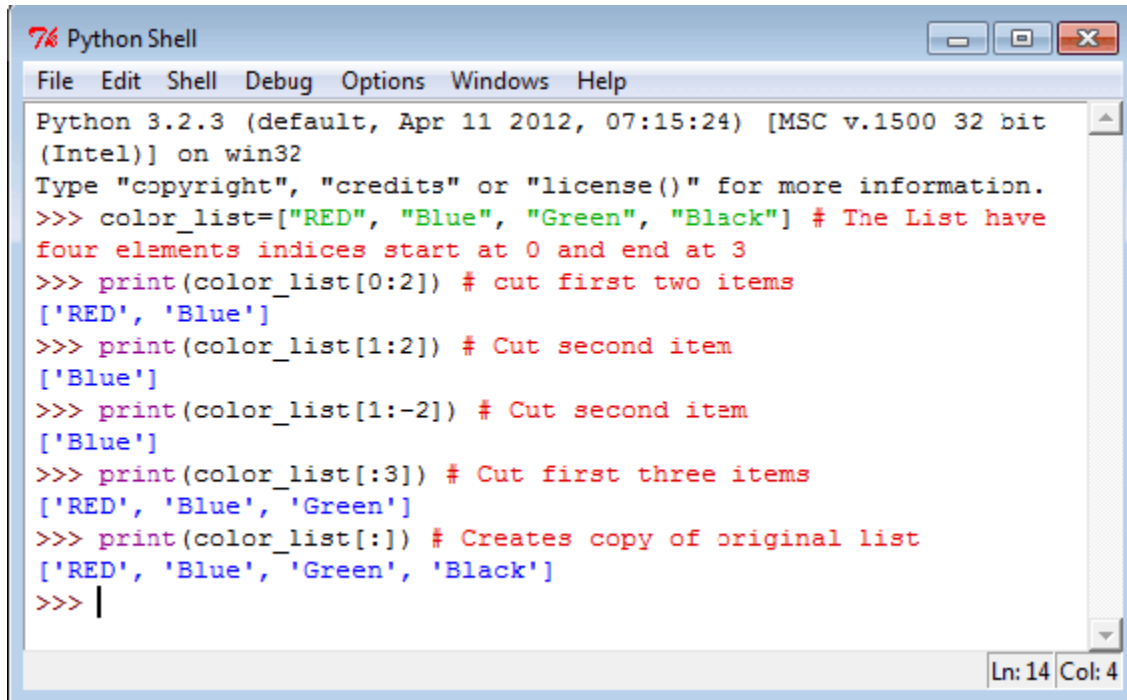
The status bar at the bottom right shows "Ln:15 Col:4".

List Slices

Lists can be sliced like strings and other sequences. The syntax of list slices is easy :

```
sliced_list = List_Name[startIndex:endIndex]
```

This refers to the items of a list starting at index startIndex and stopping just before index endIndex. The default values for list are 0 (startIndex) and the end (endIndex) of the list. If you omit both indices, the slice makes a copy of the original list. See the following statements.

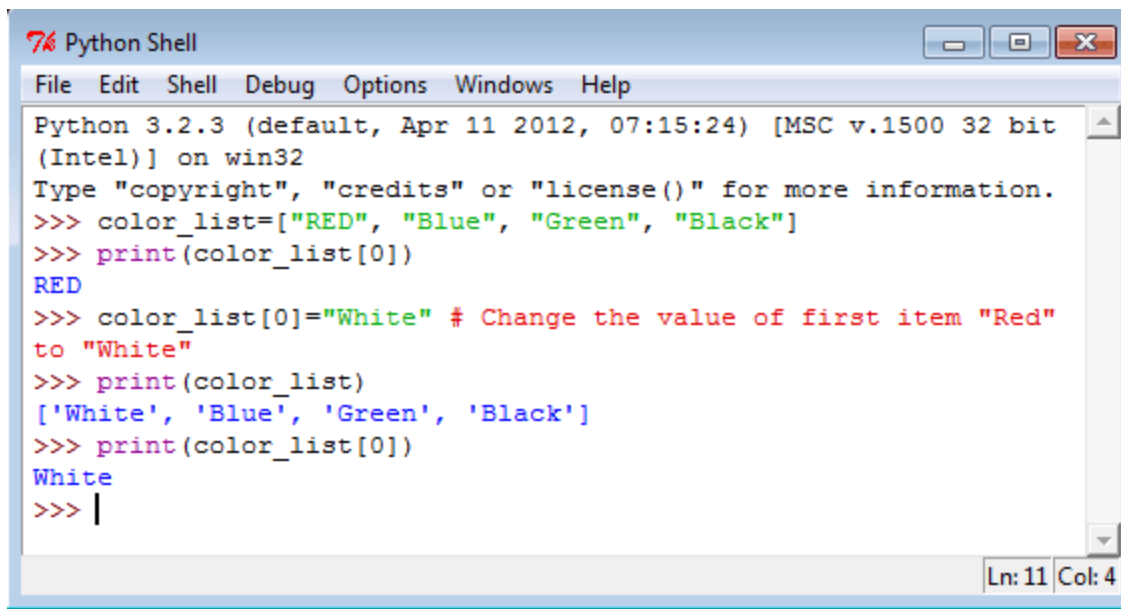
A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following code:

```
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> color_list=["RED", "Blue", "Green", "Black"] # The List have four elements indices start at 0 and end at 3
>>> print(color_list[0:2]) # cut first two items
['RED', 'Blue']
>>> print(color_list[1:2]) # Cut second item
['Blue']
>>> print(color_list[1:-2]) # Cut second item
['Blue']
>>> print(color_list[:3]) # Cut first three items
['RED', 'Blue', 'Green']
>>> print(color_list[:]) # Creates copy of original list
['RED', 'Blue', 'Green', 'Black']
>>> |
```

The status bar at the bottom right shows "Ln: 14 Col: 4".

Lists are Mutable

Items in the list are mutable i.e. after creating a list you can change any item in the list. See the following statements.

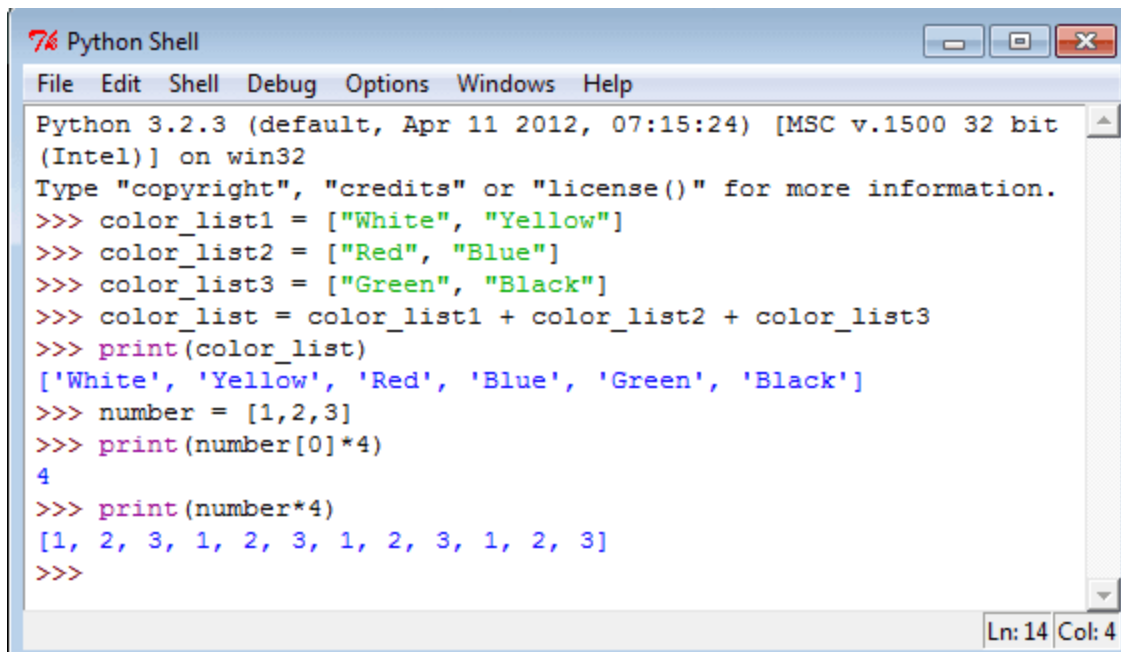
A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following code and output:

```
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> color_list=["RED", "Blue", "Green", "Black"]
>>> print(color_list[0])
RED
>>> color_list[0]="White" # Change the value of first item "Red" to "White"
>>> print(color_list)
['White', 'Blue', 'Green', 'Black']
>>> print(color_list[0])
White
>>> |
```

The status bar at the bottom right shows "Ln: 11 Col: 4".

Using + and * operators in List

Use + operator to create a new list that is a concatenation of two lists and use * operator to repeat a list. See the following statements.

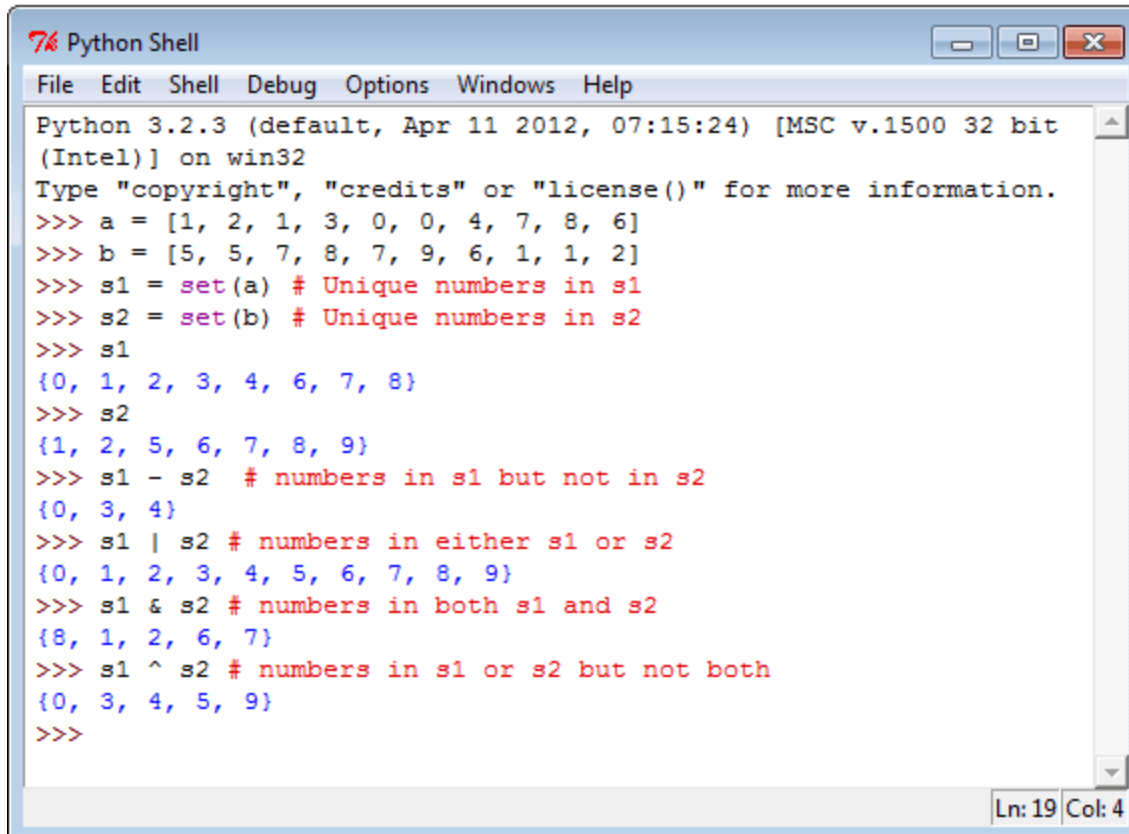
A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following code and output:

```
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> color_list1 = ["White", "Yellow"]
>>> color_list2 = ["Red", "Blue"]
>>> color_list3 = ["Green", "Black"]
>>> color_list = color_list1 + color_list2 + color_list3
>>> print(color_list)
['White', 'Yellow', 'Red', 'Blue', 'Green', 'Black']
>>> number = [1,2,3]
>>> print(number[0]*4)
4
>>> print(number*4)
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
>>>
```

The status bar at the bottom right shows "Ln: 14 Col: 4".

Sets

A set is an unordered collection of unique elements. Basic uses include dealing with set theory (which support mathematical operations like union, intersection, difference, and symmetric difference) or eliminating duplicate entries. See the following statements.

A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following code and output:

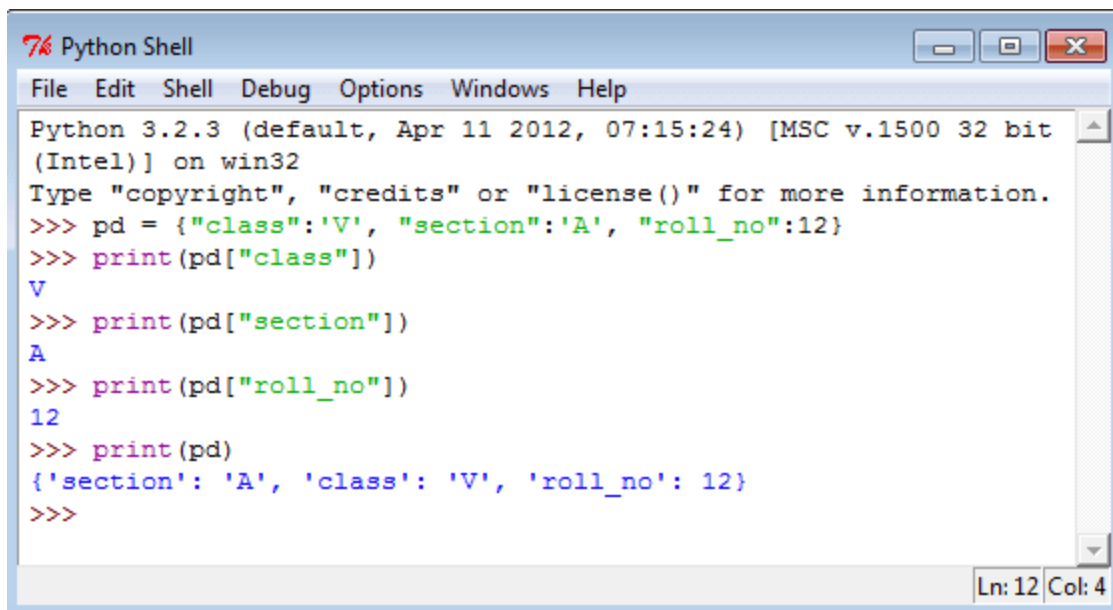
```
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a = [1, 2, 1, 3, 0, 0, 4, 7, 8, 6]
>>> b = [5, 5, 7, 8, 7, 9, 6, 1, 1, 2]
>>> s1 = set(a) # Unique numbers in s1
>>> s2 = set(b) # Unique numbers in s2
>>> s1
{0, 1, 2, 3, 4, 6, 7, 8}
>>> s2
{1, 2, 5, 6, 7, 8, 9}
>>> s1 - s2 # numbers in s1 but not in s2
{0, 3, 4}
>>> s1 | s2 # numbers in either s1 or s2
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> s1 & s2 # numbers in both s1 and s2
{8, 1, 2, 6, 7}
>>> s1 ^ s2 # numbers in s1 or s2 but not both
{0, 3, 4, 5, 9}
>>>
```

The status bar at the bottom right shows "Ln: 19 Col: 4".

Dictionaries

Python dictionary is a container of the unordered set of objects like lists. The objects are surrounded by curly braces `{ }`. The items in a dictionary are a comma-separated list of key:value pairs where keys and values are Python data type. Each object or value accessed by key and keys are unique in the dictionary. As keys are used for indexing, they must be the immutable type

(string, number, or tuple). You can create an empty dictionary using empty curly braces

A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following code and output:

```
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> pd = {"class": 'V', "section": 'A', "roll_no": 12}
>>> print(pd["class"])
V
>>> print(pd["section"])
A
>>> print(pd["roll_no"])
12
>>> print(pd)
{'section': 'A', 'class': 'V', 'roll_no': 12}
>>>
```

The status bar at the bottom right indicates "Ln: 12 Col: 4".

None

This type has a single value. There is a single object with this value. This object is accessed through the built-in name `None`. It is used to signify the absence of a value in many situations, e.g., it is returned from functions that don't explicitly return anything. Its truth value is false.