

VARIABLES, EXPRESSIONS AND STATEMENTS

Agenda

- What is variable, How to declare and use a variable
- Re-declare a variable
- Concatenate variable
- Delete a variable
- Global and local variable
- Keywords, statements, Expressions
- Python Operators
- Identifiers
- Literals

Variable

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory

Variables and Keywords

- Variables names can start with letters or underscore , but not with numbers.
- Keywords cannot be used as variables

Python reserves 31 keywords for its use:

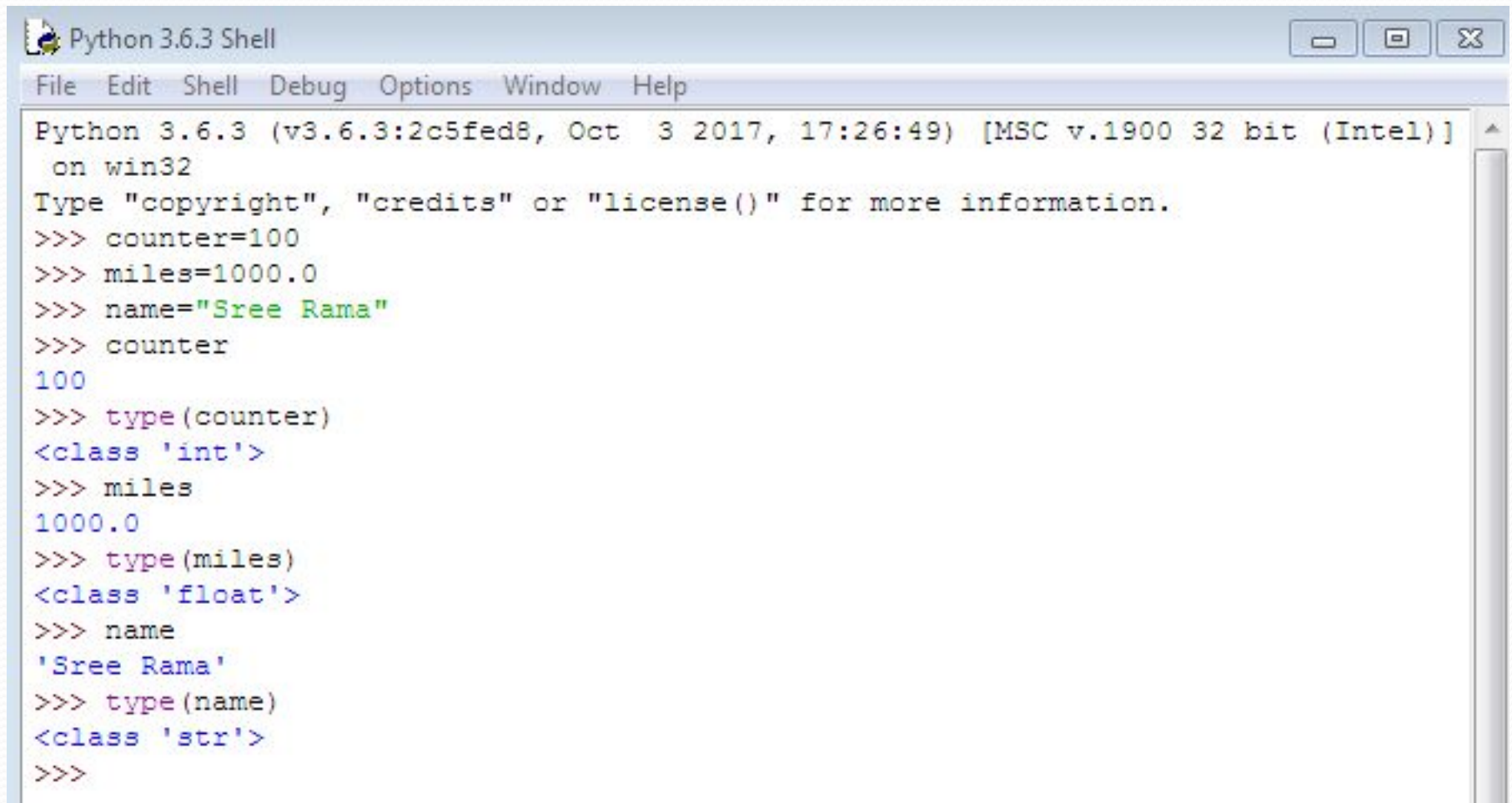
False	class	Finally	Is	Return
None	continue	For	Lambda	Try
True	def	from	nonlocal	While
And	Del	global	Not	With
As	Elif	If	Or	Yield
Assert	else	import	pass	
Break	Except	In	raise	

Assigning Values to Variables

- The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables

```
counter=100      #An integer Assignment  
miles=1000.0     #A floating point  
name="John"      | #A string
```

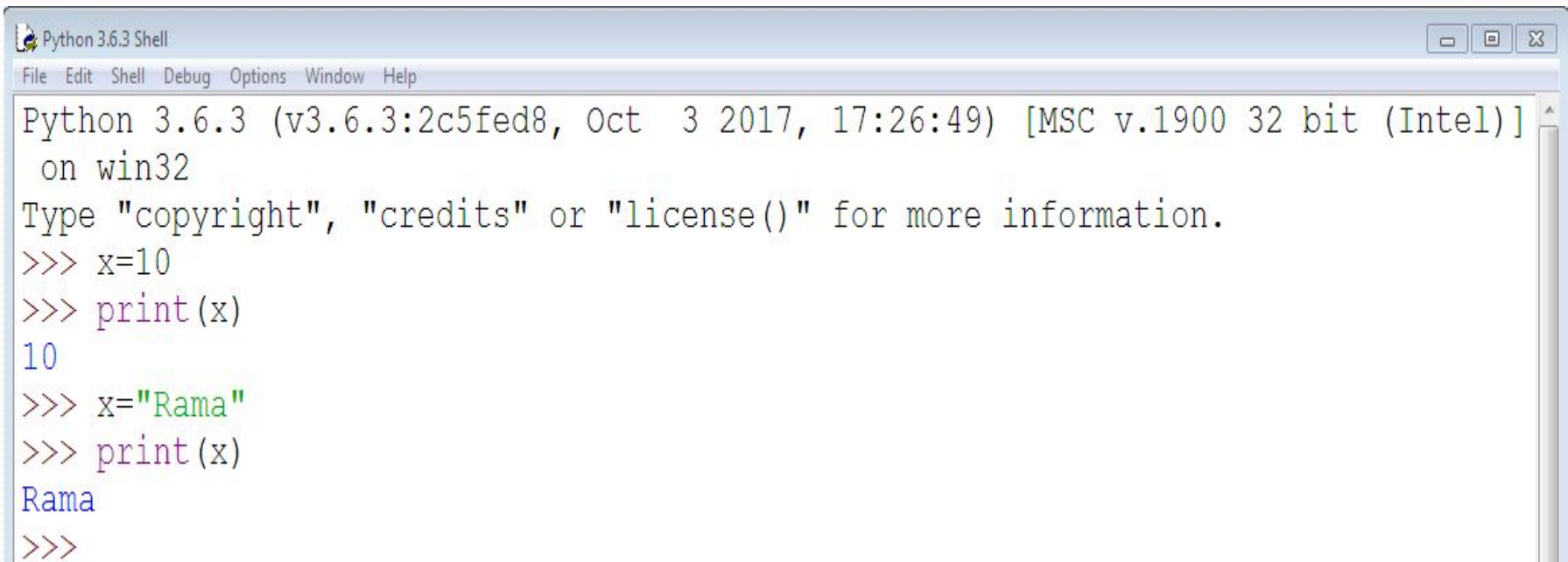
Type of a variable



```
Python 3.6.3 Shell
File Edit Shell Debug Options Window Help
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> counter=100
>>> miles=1000.0
>>> name="Sree Rama"
>>> counter
100
>>> type(counter)
<class 'int'>
>>> miles
1000.0
>>> type(miles)
<class 'float'>
>>> name
'Sree Rama'
>>> type(name)
<class 'str'>
>>>
```

Re-declare a variable

- You can re-declare the variable even after you have declared it once.

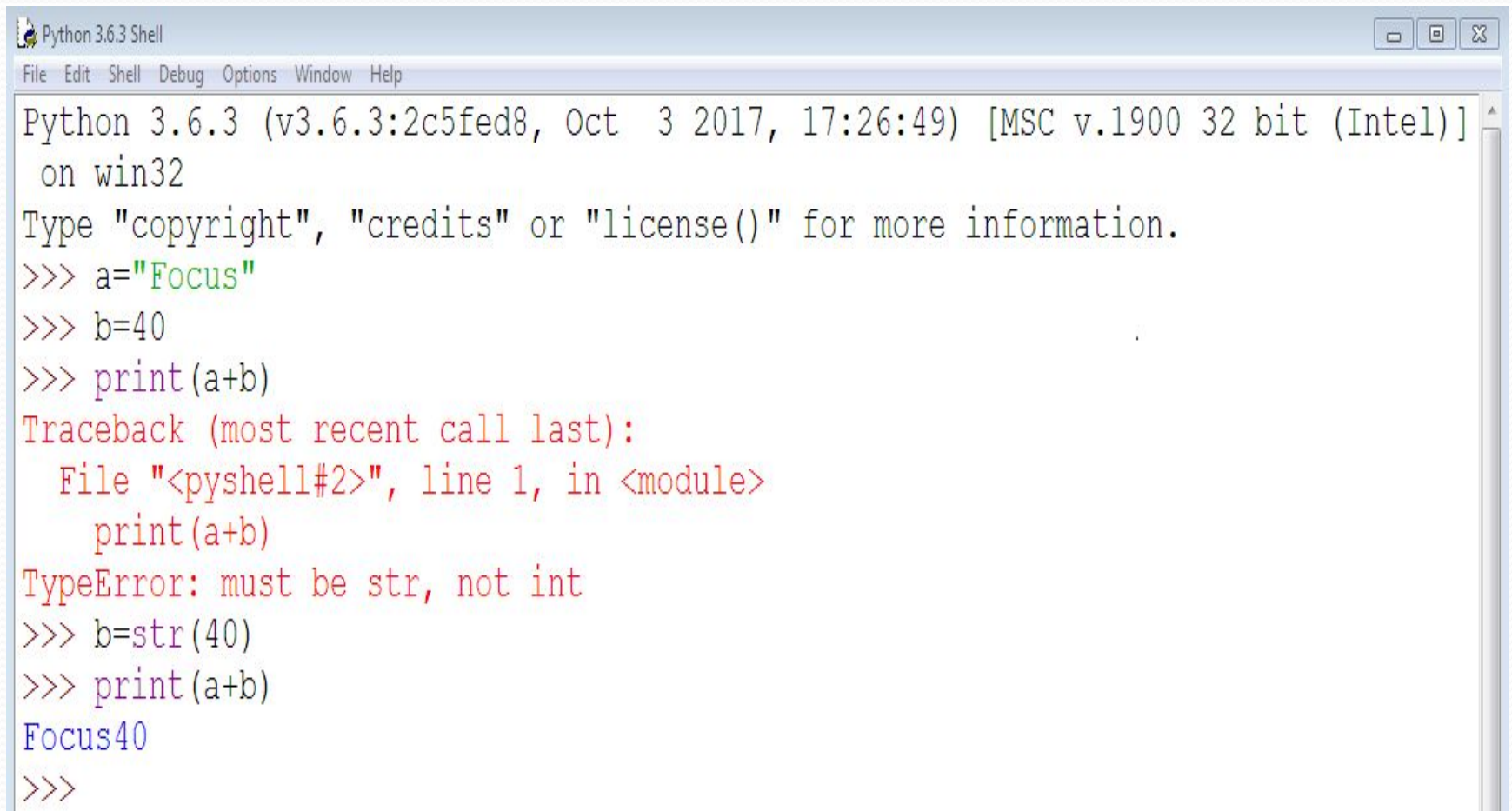
A screenshot of a Python 3.6.3 Shell window. The window has a title bar that says "Python 3.6.3 Shell" and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main area of the window shows the Python interpreter's startup message: "Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)] on win32". Below this, it says "Type 'copyright', 'credits' or 'license()' for more information." The user has entered two lines of code: ">>> x=10" followed by ">>> print(x)", which outputs "10". Then, the user enters ">>> x='Rama'" followed by ">>> print(x)", which outputs "Rama". The prompt ">>>" is shown again at the bottom, indicating the interpreter is ready for more input.

```
Python 3.6.3 Shell
File Edit Shell Debug Options Window Help
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> x=10
>>> print(x)
10
>>> x="Rama"
>>> print(x)
Rama
>>>
```

Concatenate Variable

- Let's see whether you can concatenate different data types like string and number together
- For example, we will concatenate “Focus” with number “40”.
- Unlike Java, which concatenates number with string without declaring number as string, Python requires declaring the number as string otherwise it will show a TypeError

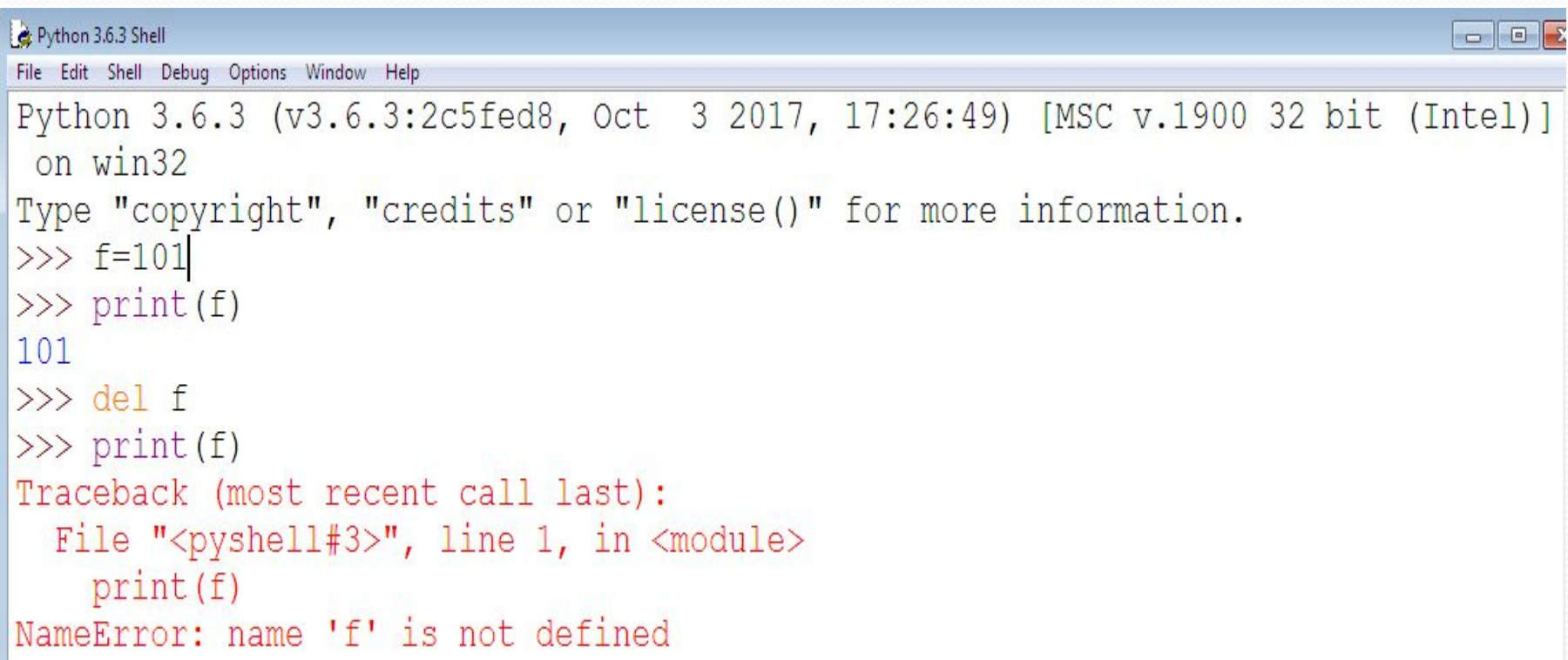
Example

A screenshot of a Python 3.6.3 Shell window. The window has a title bar that says "Python 3.6.3 Shell" and standard Windows window controls (minimize, maximize, close). Below the title bar is a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main area of the window contains a text-based interface for the Python interpreter. It shows the version and build information: "Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)] on win32". It then prompts the user to type "copyright", "credits", or "license()" for more information. The user enters three lines of code: ">>> a='Focus'", ">>> b=40", and ">>> print(a+b)". The interpreter responds with a "Traceback (most recent call last):" message, showing the error occurred in "<pyshell#2>", line 1, in <module>, at the line "print(a+b)". The error message is "TypeError: must be str, not int". The user then enters ">>> b=str(40)" and ">>> print(a+b)". The interpreter outputs "Focus40" and then ">>>" on the next line.

```
Python 3.6.3 Shell
File Edit Shell Debug Options Window Help
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> a="Focus"
>>> b=40
>>> print(a+b)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    print(a+b)
TypeError: must be str, not int
>>> b=str(40)
>>> print(a+b)
Focus40
>>>
```

Delete a variable

- You can also delete variable using the command **del** "variable name"

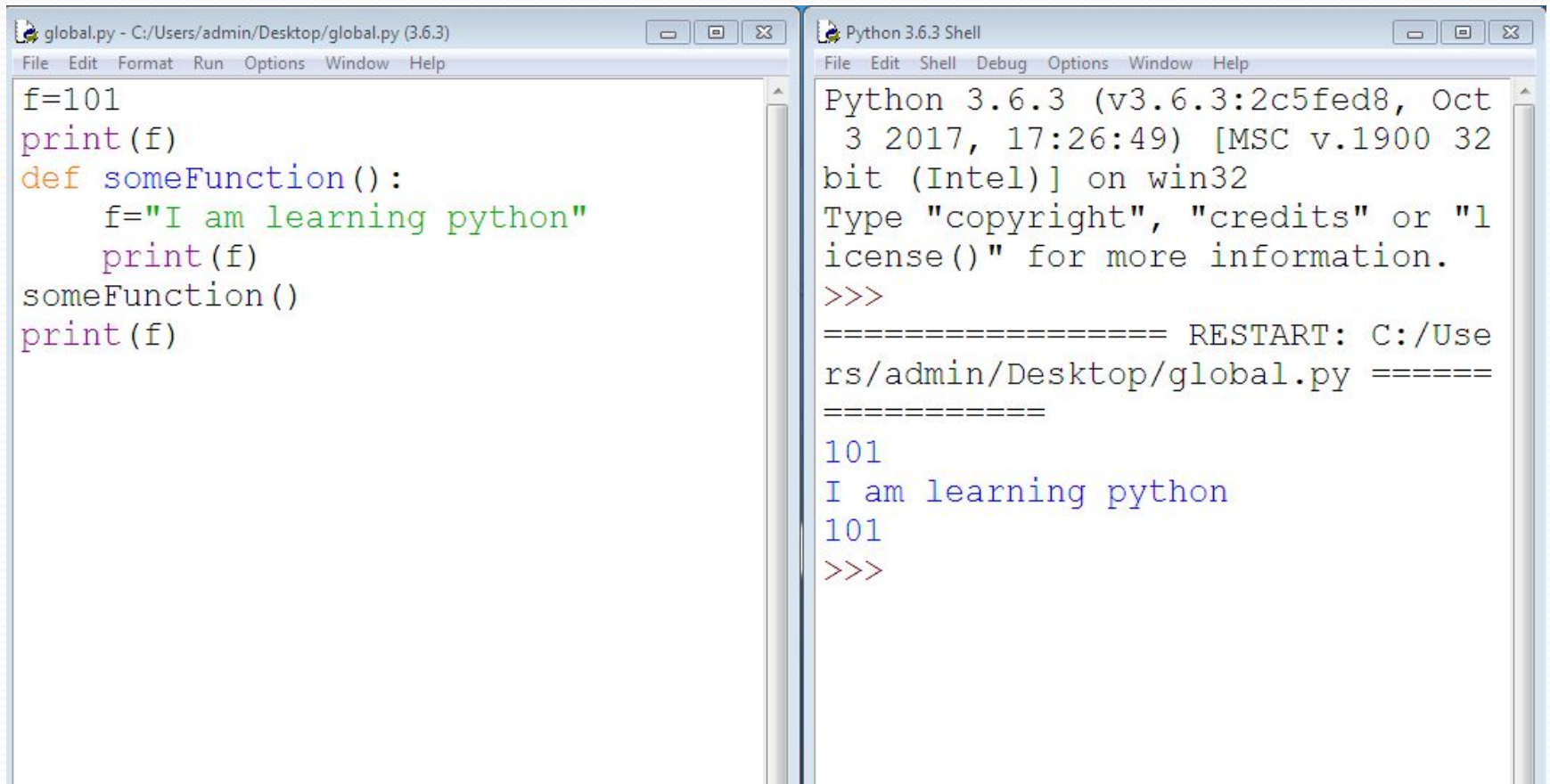
A screenshot of a Python 3.6.3 Shell window. The window has a title bar that says "Python 3.6.3 Shell" and a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main text area shows the following code and output:

```
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> f=101
>>> print(f)
101
>>> del f
>>> print(f)
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    print(f)
NameError: name 'f' is not defined
```

Global and Local Variable

- **Global Variable**:- when you want to use the same variable for rest of your program or module you declare it global variable
- **Local Variable**:- while if you want to use the variable in a specific function or method you use local variable.

Example



The image shows a side-by-side comparison of a Python script and its execution. On the left is a text editor window titled 'global.py - C:/Users/admin/Desktop/global.py (3.6.3)'. It contains the following code:

```
f=101
print(f)
def someFunction():
    f="I am learning python"
    print(f)
someFunction()
print(f)
```

On the right is a 'Python 3.6.3 Shell' window. It displays the output of running the script. The first two lines show the Python version and system information. The third line shows the prompt 'Type "copyright", "credits" or "license()" for more information.' followed by the prompt '>>>'.

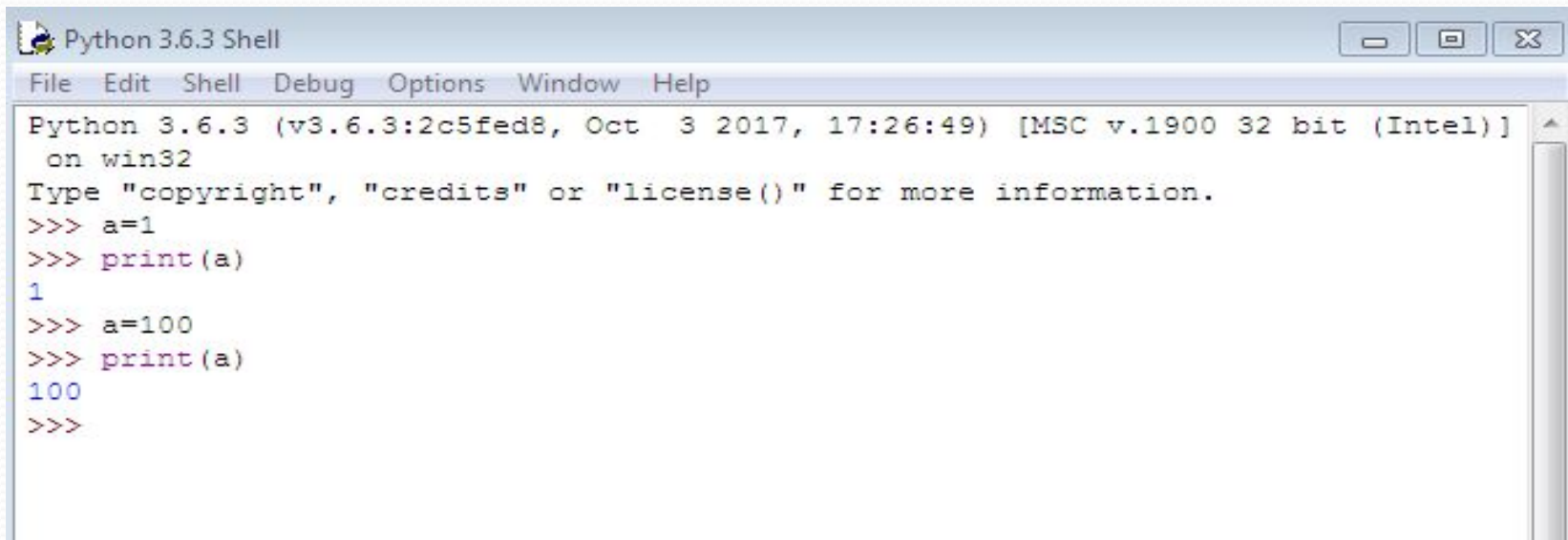
```
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/admin/Desktop/global.py =====
=====
101
I am learning python
101
>>>
```

Explanation

- Variable "f" is **global** in scope and is assigned value 101 which is printed in output
- Variable f is again declared in function and assumes **local** scope. It is assigned value "I am learning Python." which is printed out as an output. This variable is different from the global variable "f" define earlier
- Once the function call is over, the local variable f is destroyed. At line 12, when we again, print the value of "f" is it displays the value of global variable f=101

Statements

- A **statement** is an instruction that the Python interpreter can execute. We have seen two kinds of statements: print and assignment

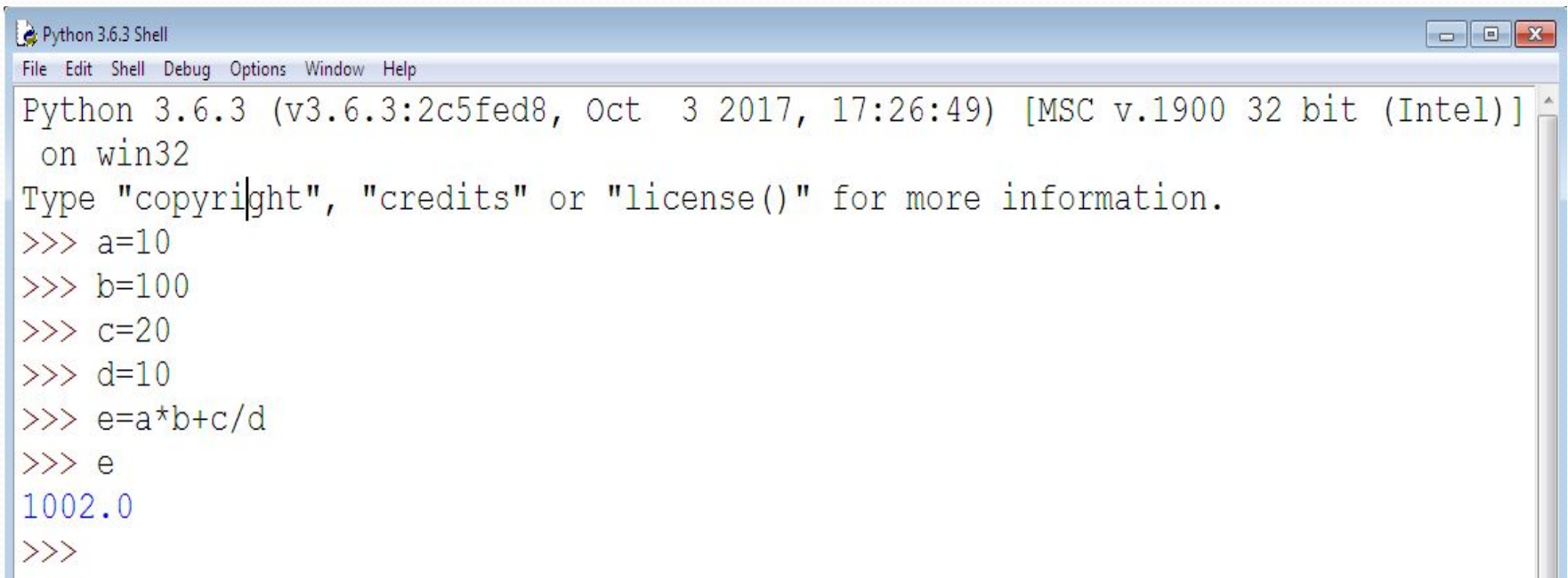
A screenshot of a Python 3.6.3 Shell window. The window has a title bar with the text 'Python 3.6.3 Shell' and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main area of the window displays the following text:

```
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=1
>>> print(a)
1
>>> a=100
>>> print(a)
100
>>>
```

Expressions

- An **expression** is a combination of values, variables, and operators

```
>>> 2 + 3
5
>>> 3 * 5
15
>>>
```



```
Python 3.6.3 Shell
File Edit Shell Debug Options Window Help
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=10
>>> b=100
>>> c=20
>>> d=10
>>> e=a*b+c/d
>>> e
1002.0
>>>
```


Operators and operands

- **Operators** are special symbols that represent computations like addition and multiplication etc
- The values the operator uses are called **operands**.
- Example: $C=A+B$
- The operators $+$, $-$, $*$, $/$ and $**$ perform addition, subtraction, multiplication, division and exponentiation
- Here A ,B, C are Operands

Operators

Operators

These operations (operators) can be applied to all numeric types:

Operator	Description	Example
+, -	Addition, Subtraction	10 - 3
*, %	Multiplication, Modulo	27 % 7 Result: 6
/	Division This operation results in different results for Python 2.x (like floor division) and Python 3.x	Python3: <pre>>>> 10 / 3 3.3333333333333335</pre> and in Python 2.x: <pre>>>> 10 / 3 3</pre>
//	Truncation Division (also known as floordivision or floor division) The result of this division is the integral part of the result, i.e. the fractional part is truncated, if there is any. It works both for integers and floating-point numbers, but there is a difference in the type of the results: If both the dividend and the divisor are integers, the result will be also an integer. If either the dividend or the divisor is a float, the result will be the truncated result as a float.	<pre>>>> 10 // 3 3</pre> If at least one of the operands is a float value, we get a truncated float value as the result.

Continue...

+x, -x	Unary minus and Unary plus (Algebraic signs)	-3
~x	Bitwise negation	~3 - 4 Result: -8
**	Exponentiation	10 ** 3 Result: 1000
or, and, not	Boolean Or, Boolean And, Boolean Not	(a or b) and c
in	"Element of"	1 in [3, 2, 1]
<, <=, >, >=, !=, ==	The usual comparison operators	2 <= 3
~, &, ^	Bitwise Or, Bitwise And, Bitwise XOR	6 ^ 3
<<, >>	Shift Operators	6 << 3

Order of precedence

- When more than one operator appears in an expression, the order of evaluation depends on the **rules of precedence**. For mathematical operators, Python follows mathematical convention. The acronym **PEMDAS** is a useful way to remember the rules:
- Paranthesis, Exponentiation, Multiplication, Division, Addition, Subtraction

Identifiers

- Identifier is the name given to entities like class, functions, variables etc. in Python.
- It helps differentiating one entity from another.

Rules For writing Identifiers

- Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_). Names like myClass, var_1 and print_this_to_screen, all are valid example.
- An identifier cannot start with a digit. 1 Variable is invalid, but variable1 is perfectly fine.
- Keywords cannot be used as identifiers.
- We cannot use special symbols like !, @, #, \$, % etc. in our identifier.

Python Literals

- Literals can be defined as a data that is given in a variable or constant
- **String literals:**String literals can be formed by enclosing a text in the quotes. We can use both single as well as double quotes for a String.
- **Eg:**"Aman" , '12345'
- **Numeric literals:**Numeric Literals are immutable.

Types of Strings

- There are two types of Strings supported in Python:
- **a)Single line String**- Strings that are terminated within a single line are known as Single line Strings.
- **Ex: a='apssdc'**
- **b)Multi line String**- A piece of text that is spread along multiple lines is known as Multiple line String.

Continue...

- There are two ways to create Multiline Strings:
- 1) Adding black slash at the end of each line
- 2) Using triple quotation marks

```
>>> text1='hello\  
user'  
>>> text1  
'hellouser'  
>>>
```

```
>>> str2="""welcome  
to  
APSSDC"""  
>>> print(str2)  
welcome  
to  
APSSDC  
>>> str2='' 'WELCOME  
TO  
APSSDC'  
>>> print(str2)  
WELCOME  
TO  
APSSDC  
>>>
```


Numeric literals

- Numeric Literals are immutable. Numeric literals can belong to following four different numerical types.

<u>Int</u> (signed integers)	Long(long integers)	float(floating point)	Complex(complex)
Numbers(can be both positive and negative) with no fractional part.eg: 100	Integers of unlimited size followed by lowercase or uppercase L eg: 87032845L	Real numbers with both integer and fractional part eg: -26.2	In the form of <u>a+bj</u> where the real part and b for imaginary part of complex eg: 3.14j

Boolean and Special literals

- **Boolean literals:** A Boolean literal can have any of the two values: True or False
- **Special literals:** Python contains one special literal i.e., None. None is used to specify to that field that is not created. It is also used for end of lists in Python.

```
>>> val1=10
>>> val2=None
>>> print(val1,val2)
10 None
>>>
```

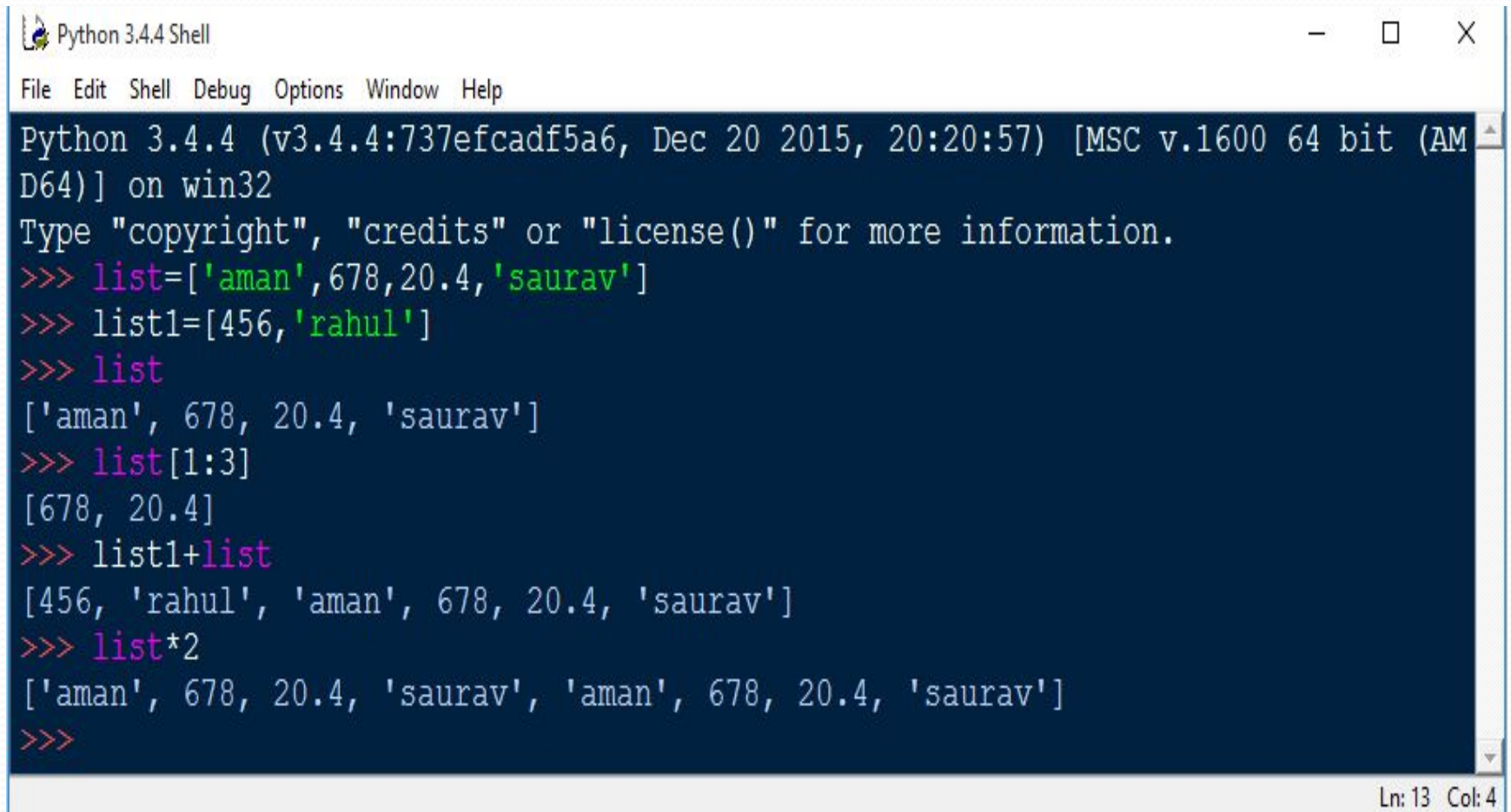
Literal Collections

- **Literal Collections:** Collections such as tuples, lists and Dictionary are used in Python.
- **List:** List contain items of different data types. Lists are mutable i.e., modifiable.
- **Tuple:** A **tuple** is a sequence of immutable python objects
- **Dictionary:** Python dictionary is an unordered collection of items. While other compound data types have only value as an element, a dictionary has a key: value pair.

Lists

- List contain items of different data types. Lists are mutable i.e., modifiable.
- The values stored in List are separated by commas(,) and enclosed within a square brackets([]). We can store different type of data in a List.
- Value stored in a List can be retrieved using the slice operator([] and [:]).
- The plus sign (+) is the list concatenation and asterisk(*) is the repetition operator.

Continue...

A screenshot of a Python 3.4.4 Shell window. The window has a title bar with the text 'Python 3.4.4 Shell' and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main area is a dark blue terminal with white text. It shows the Python version and build information, followed by instructions to type 'copyright', 'credits', or 'license()' for more information. Then, several Python commands are entered and executed: creating a list 'list' with elements 'aman', 678, 20.4, and 'saurav'; creating a list 'list1' with elements 456 and 'rahul'; printing 'list'; slicing 'list' from index 1 to 3; concatenating 'list1' and 'list'; and doubling 'list' with the '*' operator. The status bar at the bottom right shows 'Ln: 13 Col: 4'.

```
Python 3.4.4 Shell
File Edit Shell Debug Options Window Help
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> list=['aman',678,20.4,'saurav']
>>> list1=[456,'rahul']
>>> list
['aman', 678, 20.4, 'saurav']
>>> list[1:3]
[678, 20.4]
>>> list1+list
[456, 'rahul', 'aman', 678, 20.4, 'saurav']
>>> list*2
['aman', 678, 20.4, 'saurav', 'aman', 678, 20.4, 'saurav']
>>>
```

Ln: 13 Col: 4

Tuple

- A **tuple** is a sequence of immutable python objects

```
>>> mytuple=()
>>> print(mytuple)
()
>>> integer_tuple=(1,2,3)
>>> print(integer_tuple)
(1, 2, 3)
>>> mixed_tuple=(1,"Hello",3.4)
>>> print(mixed_tuple)
(1, 'Hello', 3.4)
>>> nested_tuple=("mouse",[1,2,3],(10,20,30))
>>> print(nested_tuple)
('mouse', [1, 2, 3], (10, 20, 30))
>>>
```


Dictionaries

- Python dictionary is an unordered collection of items.

```
>>> mydict={1:'one',2:'two',3:'three'}
>>> mydict
{1: 'one', 2: 'two', 3: 'three'}
>>> mydict.items
<built-in method items of dict object at 0x022DE4B0>
>>> mydict.values
<built-in method values of dict object at 0x022DE4B0>
>>> mydict.items()
dict_items([(1, 'one'), (2, 'two'), (3, 'three')])
>>> mydict.values()
dict_values(['one', 'two', 'three'])
>>> mydict.keys()
dict_keys([1, 2, 3])
>>>
```

Continue.....

```
>>> dictionary={}
>>> intdictionary={1:'apple',2:'ball'}
>>> intdictionary
{1: 'apple', 2: 'ball'}
>>> mixed_key_dic={'name':'John',1:[2,4,3]}
>>> mixed_key_dic
{'name': 'John', 1: [2, 4, 3]}
>>> dic=dict({1:'apple',2:'ball'})
>>> dic
{1: 'apple', 2: 'ball'}
>>> dic2=dict([(1,'apple'),(2,'ball')])
>>> dic2
{1: 'apple', 2: 'ball'}
>>>
```