

INTRODUCTION TO PYTHON

TABLE OF CONTENTS

1. INTRODUCTION TO PYTHON AND PYTHON IDLE	1
1.1. PYTHON AND PYTHON IDLE	1
1.1.1. Interactive Mode	5
1.1.2. Script Mode	7
1.2. VARIABLES AND TYPES	8
1.3. KEYWORDS	10
1.4. OPERATORS AND OPERANDS	10
1.4.1. Mathematical/Arithmetic Operators	11
1.4.2. Relational Operators	12
1.4.3. Logical Operators	13
1.4.4. Assignment Operators	14
1.5. EXPRESSION AND STATEMENTS	15
1.5.1. Precedence	15
1.5.2. Operation on Strings	16
2. DECISION MAKING AND LOOPING	18
2.1. DECISION MAKING STATEMENTS	18
2.1.1. If Statement	19
2.1.2. If...else Statement	20
2.1.3. Elif Statement	22
2.2. LOOPING STATEMENTS	23
2.2.1. While Loop	24
2.2.2. For Loop	26
2.2.3. Break Statement	28
2.2.4. Continue Statement	29
2.2.5. Nested Loop	29
3. STRING, LIST, TUPLE AND DICTIONARY	31
3.1. STRINGS	31
3.1.1. Length of String	32
3.1.2. String Comparison	33
3.1.3. String Traversal	34
3.1.4. String Traversal and Counting	35
3.1.5. String Slices	36
3.2. LIST	37
3.2.1. Accessing List Elements	38
3.2.2. List Traversal	38
3.2.3. Length of a List	40
3.2.4. Operations on a List	40

3.2.5.List Membership	41
3.2.6.List Slices	41
3.2.7.Are Lists Mutable?	42
3.2.8.Clone List	43
3.2.9.List Aliasing	43
3.2.10. Nested List	44
3.2.11. Deleting a List	45
3.3. TUPLES	46
3.3.1.Selecting Elements of Tuple	47
3.3.2.Length of a Tuple	48
3.3.3.Tuple Assignment	49
3.4. DICTIONARIES	50
4. FUNCTIONS	53
4.1. FUNCTIONS	53
4.1.1.Function in Module	53
4.1.2.Built in Functions	55
4.1.3.Composition	55
4.1.4.User Defined Functions	56
4.1.5.Scope of Variables	60
4.1.6.Default Arguments	63
4.1.7.Recursions	63
4.1.8.Type Conversion	66
5. FILES AND EXCEPTIONS	67
5.1. FILE OPENING AND WRITTING	67
5.2. REDING AND DISPLAYING FILE CONTENTS	69
5.3. PICKING	73
5.4. EXCEPTIONS	74
6. CLASSES AND OBJECTS	78
6.1. CLASSES	78
6.1.1.Passing Objects as Arguments in Function	80
6.1.2.Shallow Equality and Deep Equality	80
6.1.3.Objects as Return Values	82

ALKHAN MOHAN

CHAPTER 1

INTRODUCTION TO PYTHON AND PYTHON IDLE

1.1. PYTHON AND PYTHON IDLE

Python was developed by Guido Van Rossum while he was working at Centrum Wiskunde & Informatica (CWI). Centrum Wiskunde & Informatica is located in Netherlands and is a Research Institute for Mathematics and Computer Science . Python was released in the year of 1991, and the language got its name “Python” from a BBC comedy series from seventies- “Monty Python’s Flying Circus”. Python is a very flexible, simple programming language that can be used for both Procedural and Object Oriented programming approach . The most important fact about python is that it is open source and free to use.

Below point out some of the features of Python Programming Language

- Python can be used for both scientific and non-scientific programming, it is a general purpose programming language, easily learnable with very minimal expertise.
- Python is platform independent, so that it can be run on diverse operating systems and hardware configurations.
- The library support on Python is very vast and there are a lot of add on modules available too.
- Python programs are easily readable and understandable even for beginners
- Python is compiled by interpreter, and hence it provides immediate results.
- The programs written in Python are easily readable and understandable.
- Python is easy to learn and use and hence it can be used to develop application programs and extensions.
- It is easy to learn and use.

Python is widely used by many software and development companies as their preferred development language and environment because of the above mentioned key features. The language is used by many companies in real revenue generating products, such as:

- Operations of Google search engine and Other google products such as YouTube, Gmail etc.
- Intel, Cisco, HP, IBM, etc use Python for hardware testing
- Bit Torrent peer to peer file sharing is written using Python
- Maya provides a Python scripting API
- i-Robot uses Python to develop commercial Robot.
- NASA and others use Python for their scientific programming task.

Python is widely used in scientific research and programming, since python is free to use and open source, there are a vast variety of library's available to perform complex tasks. Python is used in the area of machine learning, robotics, AI, Astronomy, Bio-Chemical Research, and many more.

Example Python Libraries:

- TensorFlow.
- Scikit-Learn.
- Numpy.
- Keras.
- PyTorch.
- LightGBM.
- Eli5.
- SciPy.

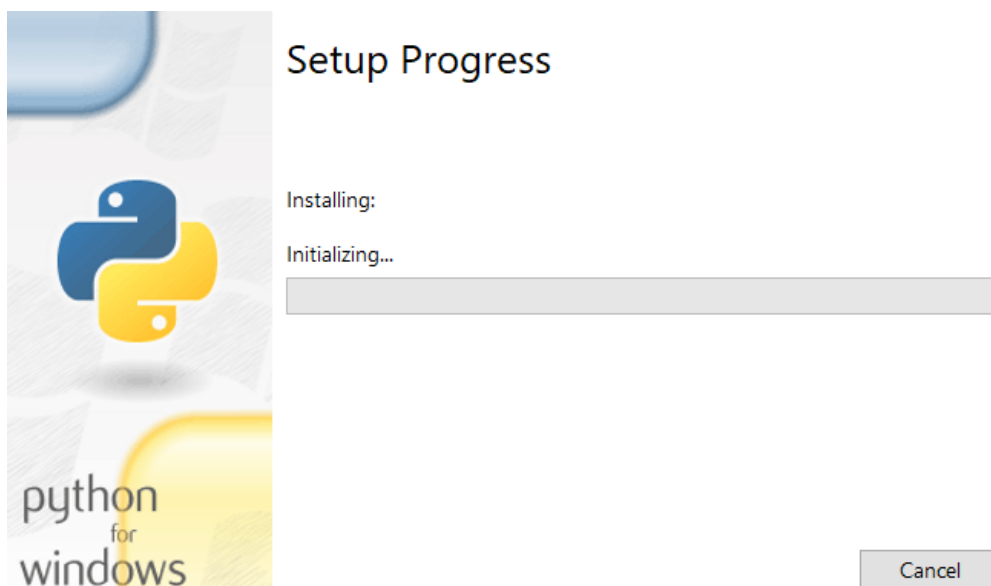
In the below section we will see how to install and configure python in your environment and familiarize Python IDLE to write and run python programs. You can consider a program as a instructions sequences to perform a specific intended task .The Computation might be mathematical, simulation, text/data manipulation and many more. The advantage is that python is very flexible and can operate on a vast variety of data types with the support of required libraries.

To write and run Python program, we need to have Python interpreter installed in our computer. You can download Python latest version (3.9.1) from <https://www.python.org/downloads/> and install it to your computer. You can download various packages for windows, Unix/Linux, and Mac OS X.

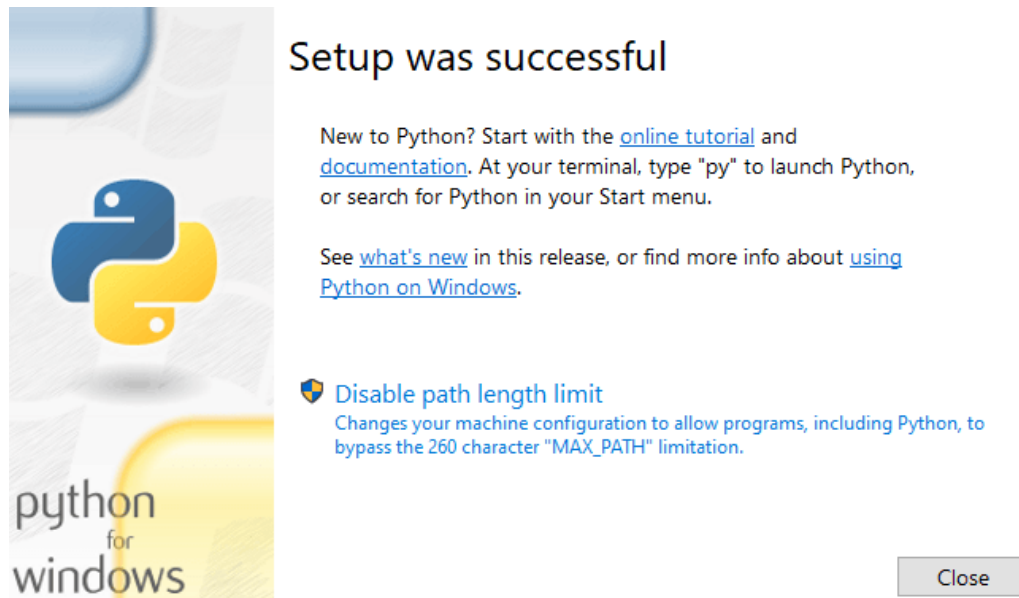
Step 1: Double click the downloaded installer to install python, choose install now and tick Add python 9.9 to PATH



Step 2: The installation will start and show a progress window and please be patient.



Step 3: After a short period of time the setup will be finished and show the below window, You may disable path length limit to overcome the 256 character path limit of a windows system.



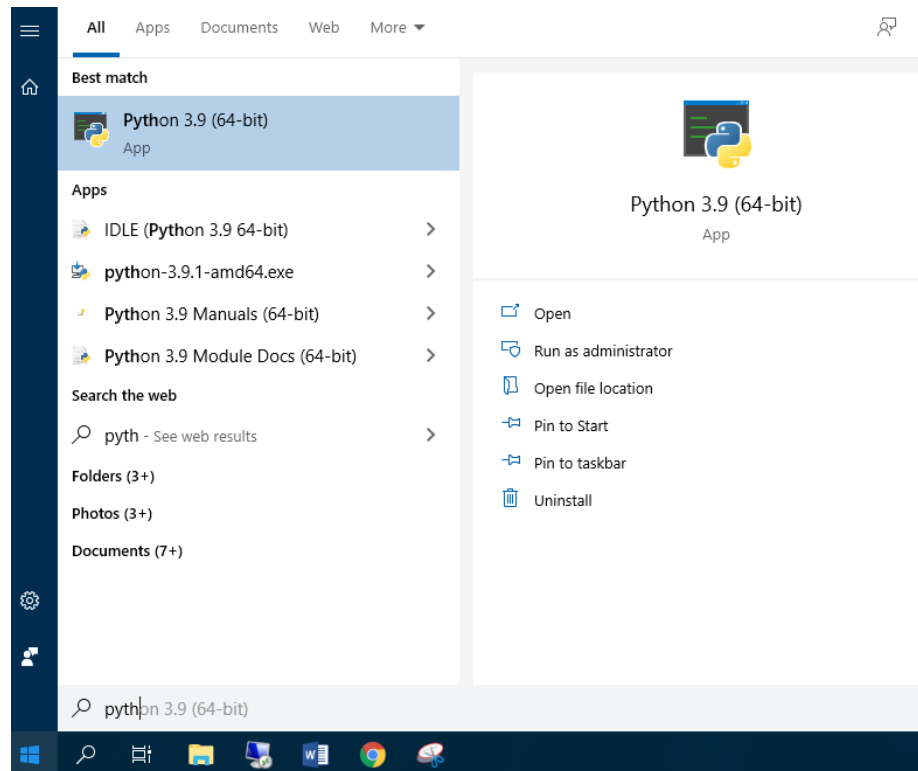
Congratulation, now you have installed python in your windows computer.

Please note that a standard python installation automatically include IDLE, but if you have selected custom installation please remember to include IDLE (GUI integrated) which is the standard, most popular Python development environment. IDLE stands for Integrated Development Environment which allows you to write, modify, run, browse and debug python program from a single environment. Python IDLE helps a developer to easily create and execute python programs.

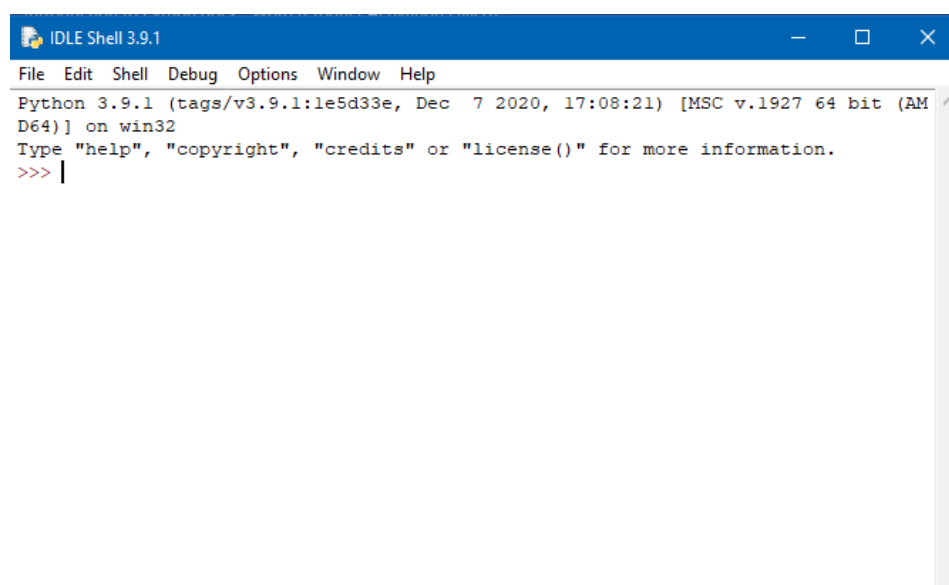
There are two modes in which you can use python shell, interactive mode and script mode. Interactive Mode, as the name suggests, allows us to interact with OS; script mode let us create and edit python source file. Let us start with interactive mode first we will type a Python statement and see the interpreter displays the result(s) immediately.

1.1.1 Interactive Mode

For working in the interactive mode, we will start Python on our computer by searching for python, the easy way is to press the windows button in your computer and start type python. And select IDLE Python 3.9(64-bit)



We will see a welcoming message of Python interpreter with version details and Python prompt, i.e., ">>>".



This prompt represent, the python interpreter is ready and expecting a python command to execute. We can say that the python is working in interactive mode if we see a python prompt as shown in above figure.

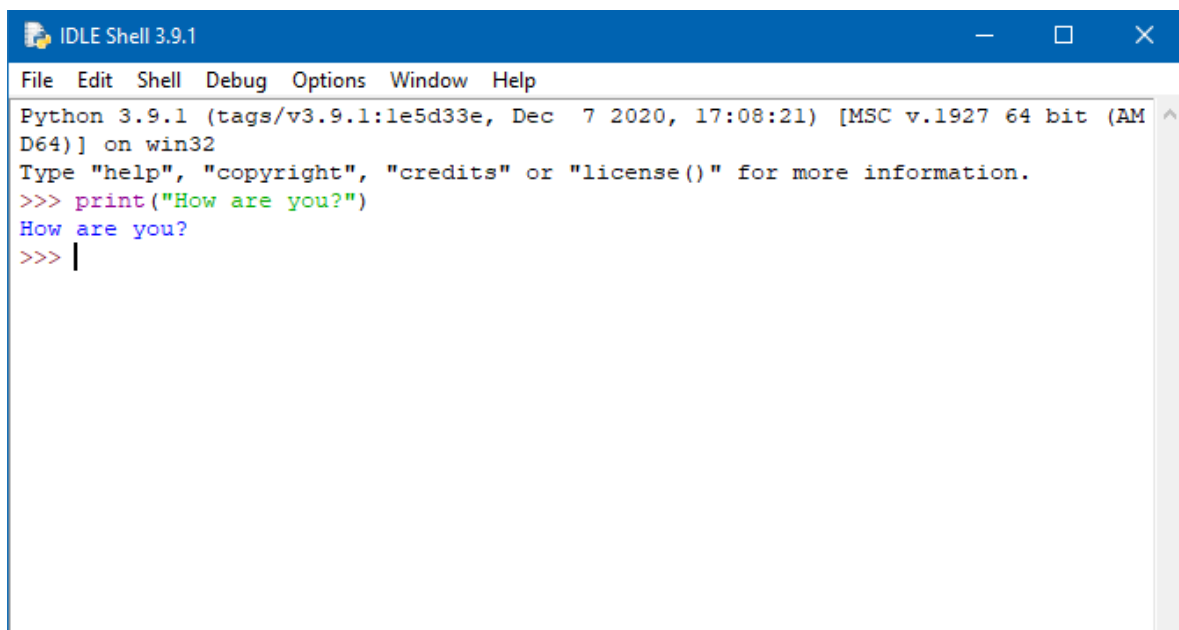
Here you can type expression / statement / command after the prompt following correct syntax, the program will respond immediately and shows the resulting output. Now let us print “This is my first program ” in python interactive mode.

Program

```
>>>print “How are you?”
```

Output

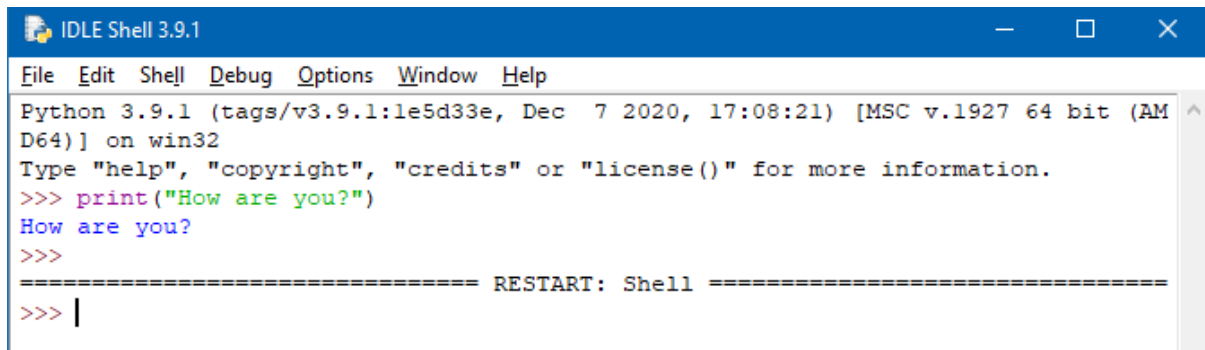
How are you?

A screenshot of the IDLE Shell 3.9.1 window. The window has a blue title bar with the text "IDLE Shell 3.9.1" and standard window controls. Below the title bar is a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main text area shows the following text: "Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32", "Type 'help', 'copyright', 'credits' or 'license()' for more information.", ">>> print('How are you?)" (where 'print' is in green and the string is in red), "How are you?" (in blue), and ">>> |" (where the vertical bar is in blue).

```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print('How are you?)"
How are you?
>>> |
```

^D (Ctrl+D)+ Enter or quit () is used to leave the interpreter.

^F6 will restart the shell.



```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("How are you?")
How are you?
>>>
===== RESTART: Shell =====
>>> |
```

1.1.2. Script Mode

Script mode is another mode in which you can run a python program, in this mode a user will type Python program in a file and then the interpreter is called to execute the statements from the file. Script mode is useful for the writing of large programs, so that there is large lines of code and we can always save, modify and reuse the code. Interactive mode is convenient for beginners and for testing small pieces of code, as interactive mode allows to test and show the result immediately.

The drawback of interactive mode is that we are not saving the statements and we have to retype the codes again and re-run them, but is useful to explore and experiment with python.

If the script mode is not made available by default with IDLE environment follow the below steps in IDLE. This allows to create and run a python script.

Follow below script after opening Python IDLE

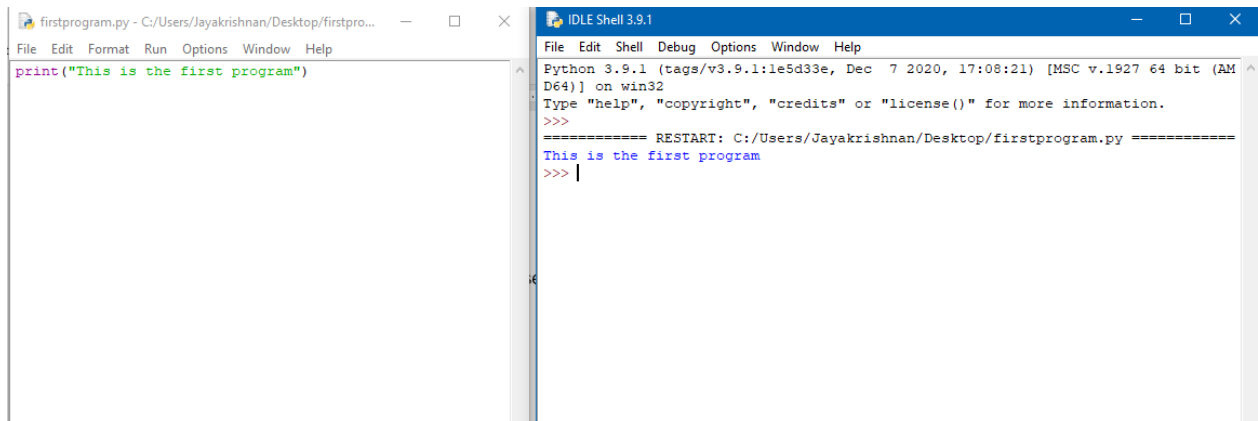
Step 1: File>Open OR File>New File (for creating a new script file)

Step 2: Write the Python code as function i.e. script

Step 3: Save it (^S) as py file. (Give a file name, say firstprogram.py)

Step 4: Execute it in interactive mode- by using RUN option (^F5)

Here “^” indicate pressing control button



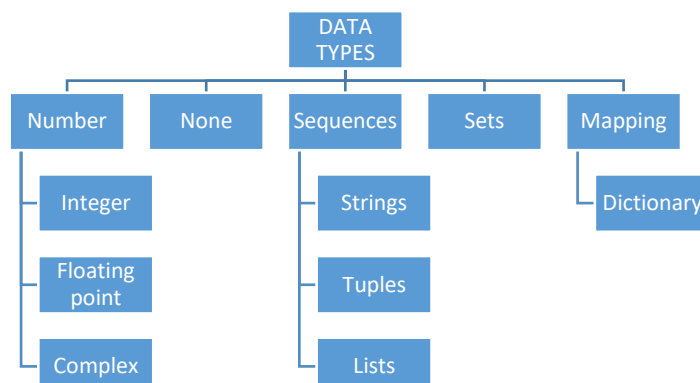
1.2. VARIABLES AND TYPES

We know that object or variable is a name which refers to a values. During programming it is more often that we have to store values, so that we can use those values later. We use objects to capture the data and then computer can manipulate to provide information. Every such object has three main components as given below:

1. An Identity, - refers to the unique id of the created object. `id()` function returns the id of the object.
2. A type – can be checked using `type ()` function
3. A value

Identity of the object: It is an integer value is the object's address in memory and does not change once it has been created. `id ()` is used to find identity of an object

Type (i.e data type): Data type refers to the type of data that variable can hold and the `type()` function can be used to print the data type of a variable. It can be one of the following: and the data types will be discussed in coming chapters.



Value of the Object (Variable): Data assigned to the variable, to bind a value to a variable we can use assignment operator (=), this is also known as binding of a variable.

Example:

```
>>> pi = 3.1415
```

Here the value is 3.1415, data type is float and the name of the variable is “pi”, the names given to the variables are otherwise called as identifiers

There are certain conditions for a variable name they are:

- can be of any size
- have allowed characters, which are a-z, A-Z, 0-9 and underscore (_)
- should begin with an alphabet or underscore
- should not be a keyword (keyword will be discussed later)
- names are case sensitive

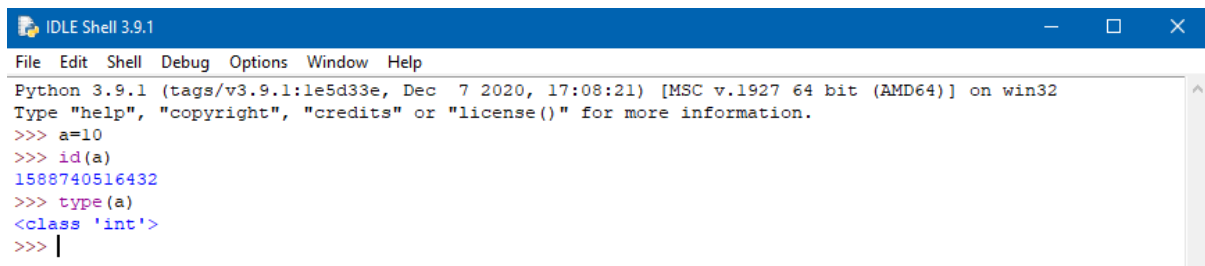
Although you can name a variable anything that satisfy above rules, it is a good practice to follow below naming conventions.

- Variable name should be meaningful and short
- Generally they are written in lower case letters

Program

```
>>> a=10
>>> id(a)
1588740516432
>>> type(a)
<class 'int'>
>>>
```

Screenshot

A screenshot of the IDLE Shell 3.9.1 window. The window has a blue title bar with the text 'IDLE Shell 3.9.1' and standard window controls. Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following code and output:

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> a=10
>>> id(a)
1588740516432
>>> type(a)
<class 'int'>
>>> |
```

1.3. KEYWORDS

Python interpreter identifies the structure of a program, by means of keywords. These are specific words and they cannot be used for any other purpose since they provide specific meaning for the interpreter to understand program structure. The keywords in python programming language is listed below.

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

1.4. OPERATORS AND OPERANDS

Operators are certain symbols that represent computations in the program statements. Operators are applied on values or variables, and such values or variables are called as operands. Operators applied on operands forms and expression. Same operators behave differently on different data types

E.g.: **c = a + b**

Here a, b, c are the operands and +, = are the operators.

Operators are categorized as follows:

- Arithmetic/ Mathematical Operators
- Relational Operators
- Logical Operators
- Assignment Operators

Let us discuss each operator with examples.

1.4.1. Mathematical/Arithmetic Operators

Arithmetical or Mathematical operators are used to perform arithmetic operations on given operands. Mathematical operators and their examples are shown in below table.

You can try these examples in your Python IDLE

Symbol	Description	Example 1	Example 2
+	Addition	>>>10+15 25	>>> "Hi" + "Hello" HiHello
-	Subtraction	>>>10-5 5	>>>30-40 -10
*	Multiplication	>>>10*40 400	>>> "Hai" * 3 HaiHaiHai
/	Division	>>>19/5 3.8 >>>19/5.0 3.8	>>> 19.0/5 3.8 >>>28/3 9
%	Remainder/ Modulo	>>>19%5 4	>>> 23%2 1
**	Exponentiation	>>>3**3 27 >>>20**.5 4.472	>>>2**10 1024
//	Integer Division	>>>5.0//2 2	>>>7/ / 2 3

1.4.2. Relational Operators

Relational operators, test the relationship between two entities, i.e. the relationship between the given operands. The expression will evaluate the relationship and return either true or false. Relational operators and their examples are given in the below table. During string comparison, the characters in both strings are compared one by one and when different characters are found their Unicode value is compared. The character with lower Unicode value is considered to be smaller.

Eg: `>>>"Hi"<= "Goodday"`

Here Character “H” is compared against “G” and the character H is higher than G, and hence the expression "Hi"<= "Goodday" is evaluated as FALSE

Symbol	Description	Example 1	Example 2
<	Less than	<code>>>>2<10</code> True	<code>>>>"Goodday"< "Hi"</code> True
>	Greater than	<code>>>>15>5</code> True	<code>>>> "Goodday" > "Hi"</code> False
<=	less than equal to	<code>>>> 1<=6</code> True	<code>>>>"Hi"<= "Goodday"</code> False
>=	greater than equal to	<code>>>>18>=18</code> True	<code>>>>"Hi">= "Goodday"</code> True
!=, <>	not equal to	<code>>>>20!=31</code> True	<code>>>>"Hello" != "HELLO"</code> True
==	equal to	<code>>>>111==111</code> True	<code>>>>"Hello" == "Hello"</code> True

Eg: `>>>"Hello" != "HELLO"`

Since the python is case sensitive “Hello” is not equal to “HELLO”, hence the expression "Hello" != "HELLO" returns True.

1.4.3. Logical Operators

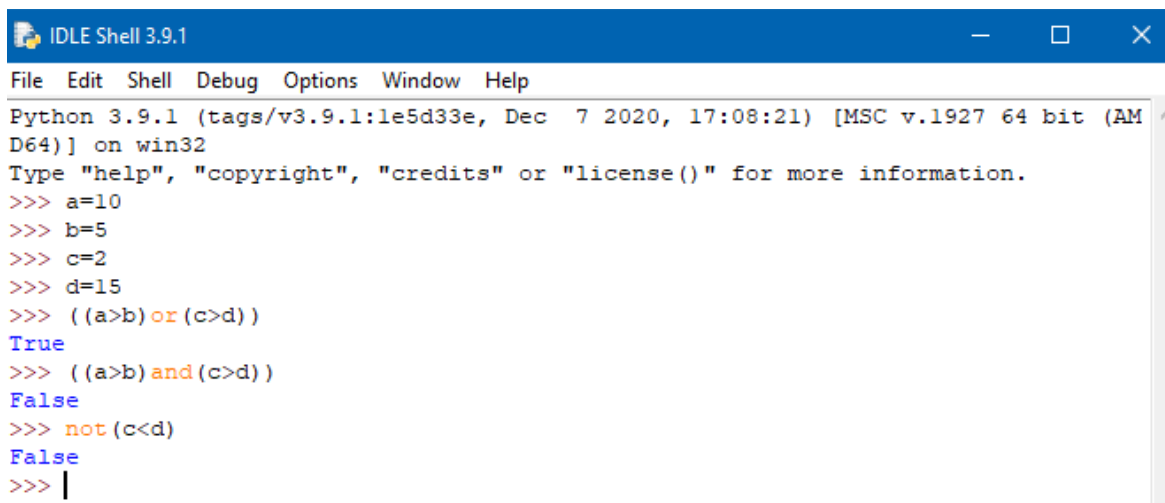
Logical operators in Python are used to check if the conditional statements returns true or false. Logical operators in Python are AND, OR and NOT

Symbol (Operator)	Description
or	If any one of the operand is true, then the condition becomes true.
and	If both the operands are true, then the condition becomes true.
not	Reverses the state of operand/condition. I.e. it returns true if the operand is false

Program

```
>>> a=10
>>> b=5
>>> c=2
>>> d=15
>>> ((a>b)or(c>d))
True
>>> ((a>b)and(c>d))
False
>>> not(c<d)
False
>>>
```

Screenshot

A screenshot of the IDLE Shell 3.9.1 window. The window has a blue title bar with the text 'IDLE Shell 3.9.1' and standard window controls. Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main area shows the Python 3.9.1 shell prompt and the execution of the same code as in the 'Program' section. The output shows 'True' for the OR condition and 'False' for both the AND condition and the NOT condition. The cursor is at the end of the last prompt line.

```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> a=10
>>> b=5
>>> c=2
>>> d=15
>>> ((a>b)or(c>d))
True
>>> ((a>b)and(c>d))
False
>>> not(c<d)
False
>>> |
```

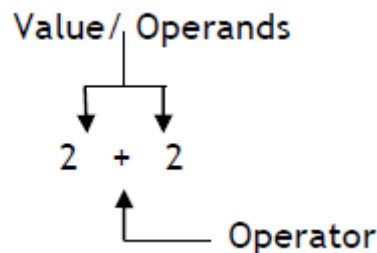
1.4.4. Assignment Operators

Assignment Operator assigns values to the left side operands. Assignment operator can be combined with arithmetic operators to get desired output.

Symbol	Description	Example	Explanation
=	Assigned values from right side operands to left variable	>>>x=12 >>>y="greetings"	x=12
+=	added and assign back the result to left operand	>>>x+=2	x=x+2 x=14
-=	subtracted and assign back the result to left operand	x-=2	x will become 10
=	multiplied and assign back the result to left operand	x=2	x will become 24
/=	divided and assign back the result to left operand	x/=2	x will become 6
%=	taken modulus using two operands and assign the result to left operand	x%=2	x will become 0
=	performed exponential (power) calculation on operators and assign value to the left operand	x=2	x will become 144
//=	performed floor division on operators and assign value to the left operand	x /= 2	x will become 6

1.5. EXPRESSION AND STATEMENTS

Expressions can be considered as a combination of value(s) (i.e. constant), variable and operators. The expressions generates a single value itself is an expression. Let us consider one example.



The expression is evaluated by the computer and find its value, in the above example $2+2$ is the expression, where each 2 can be considered as operands/values. In the .above example, the resulting value after evaluation will be 4, and we say the expression is evaluated.

1.5.1. Precedence

Now let us discuss the expression evaluation when the expression consist of sub expression(s), how does Python decides the order in which operations are done? The answer is that the expression is evaluated on the basis of the precedence of operator. Higher precedence operator is operated before lower precedence operator, where parentheses have highest order of precedence. What if operators are of same precedence? Operator associativity decides the order of evaluation when operators are of same precedence, and are not grouped by parenthesis. An operator can be either Left-associative or Right –associative.

If an operator is left associative, the operator falling on left side will be evaluated first, while in right associative operator falling on right will be evaluated first. Python Language uses the same precedence rules for operators in mathematics. The acronym **PEMDAS** is a useful way to remember the order of operations. The below listed are the precedence rules to follow in python programming.

- The highest precedence is for Parentheses and they can be used to force an expression to evaluate in the order programmer want (expressions in parentheses are evaluated first)
e.g. $2 * (4-1)$ is 6, and $(1+1)**(5-2)$ is 8.
- The next highest precedence is for exponentiation, so $2**2+1$ is 4 and not 8, and $3*1**3$ is 3 and not 27.
- Multiplication and Division operations have the similar precedence, but higher than Addition and Subtraction, which also have the same precedence. So $2*3-1$ results 5 rather than 4, and $2/3-1$ is -1, not 1 (remember that in integer division, $2/3=0$).
- Operators with the same precedence are evaluated from left to right. So in the expression $50*100/50$, the multiplication happens first, yielding $5000/50$, which in turn yields 100.

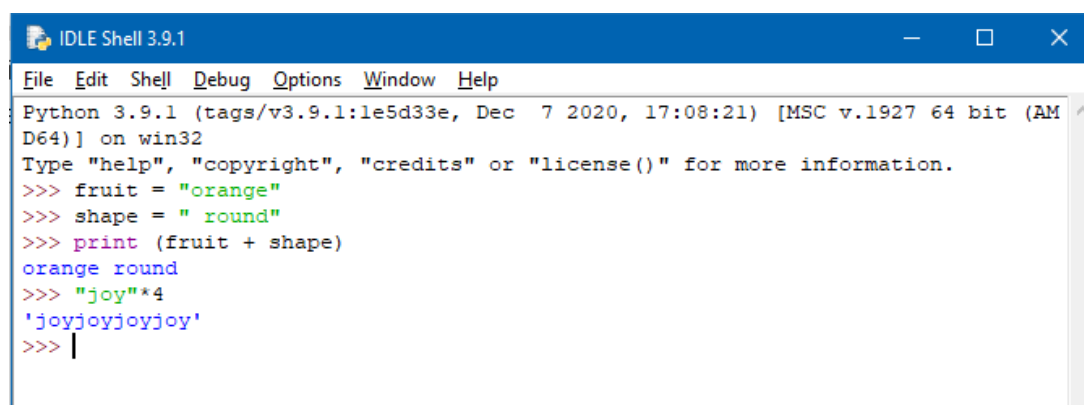
1.5.1. Operations on Strings

If the operands are strings the operator + represents concatenation, which joins two operands by linking them end-to-end. Let us consider one example

Example

```
>>> fruit = "orange"
>>> shape = " round"
>>> print (fruit + shape)
```

The output of this program is orange round.



```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> fruit = "orange"
>>> shape = " round"
>>> print (fruit + shape)
orange round
>>> "joy"*4
'joyjoyjoyjoy'
>>> |
```

The * operator performs repetition while operated on strings. For example, "joy" * 4 is " joyjoyjoyjoy ". One of the operands has to be a string; the other has to be an integer.

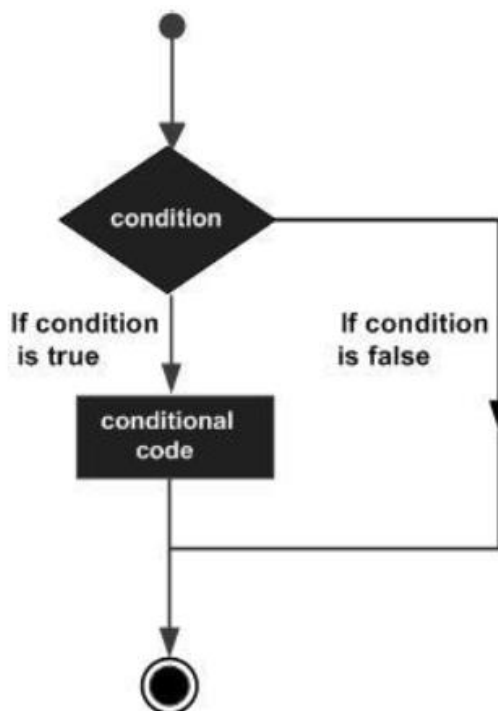
CHAPTER 2

DECISION MAKING AND LOOPING

2.1. DECISION MAKING STATEMENTS

Decision making involves the evaluation of conditions occurring while execution of the program and given actions corresponding to the condition are taken. A very simple example of a condition statement in English is “if there is rain take umbrella”. Here the condition is whether there is rain or not, if there is rain the corresponding action would be to take umbrella.

The decision statements have conditions, such conditions are evaluated to produce TRUE or FALSE as outcome. The outcome TRUE or FALSE has separate statements to execute. Below given is a very general form of decision making system found in many programming languages, including python.



Please keep in mind that any non-zero and non-null values are assumed as TRUE and zero or null values are assumed as FALSE.

Let us consider some of the conditional statements in python programming language, they are

- if statement
- if...else statement
- elif statement

2.1.1. If Statement

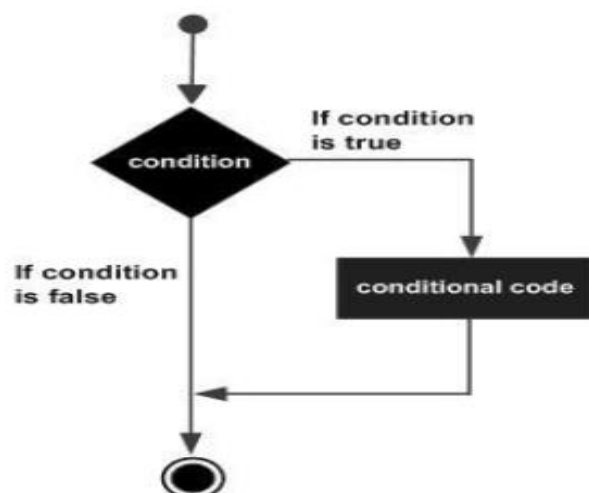
The working of IF statements are very similar to that of any programming languages and English language. If statement contains a logical expression, which is evaluated and produce an output either TRUE or FALSE. Based on the outcome corresponding statements are executed.

Syntax

if expression:

statement(s)

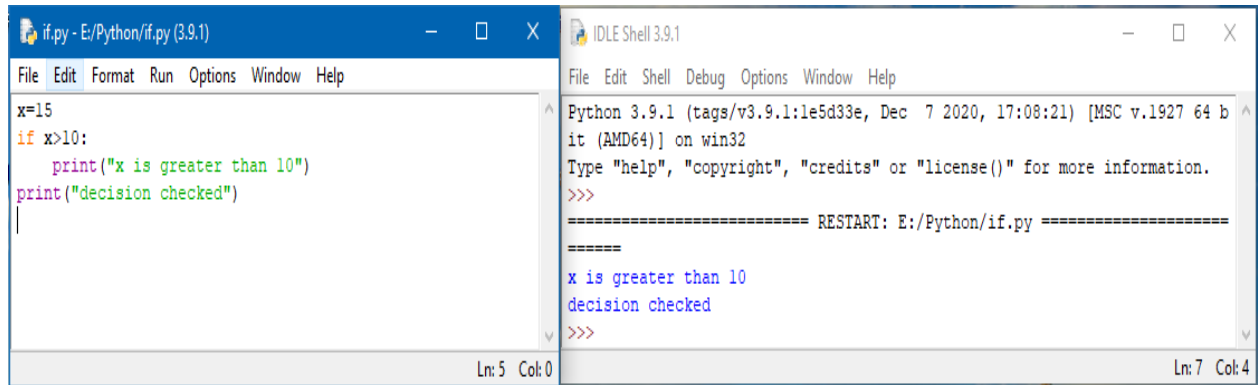
Let us discuss the below given flow chart, The block of statement(s) inside the if statement is executed only if the Boolean expression is evaluated as TRUE. If Boolean expression evaluates to FALSE, then the statements after the end of if construct are executed



Consider the below example program,

Example

```
x=15
if x>10:
    print('x is greater than 10')
print("decision checked")
```



```
if.py - E:/Python/if.py (3.9.1)
File Edit Format Run Options Window Help
x=15
if x>10:
    print('x is greater than 10')
print('decision checked')
Ln: 5 Col: 0

IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 b
it (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: E:/Python/if.py =====
>>>
x is greater than 10
decision checked
>>>
Ln: 7 Col: 4
```

here the condition is evaluated and found true hence the statement “print("x is greater than 10")” is executed and exited from if structure. Up on exiting if structure the statement “print("decision checked")” executed.

2.1.2. If...else Statement

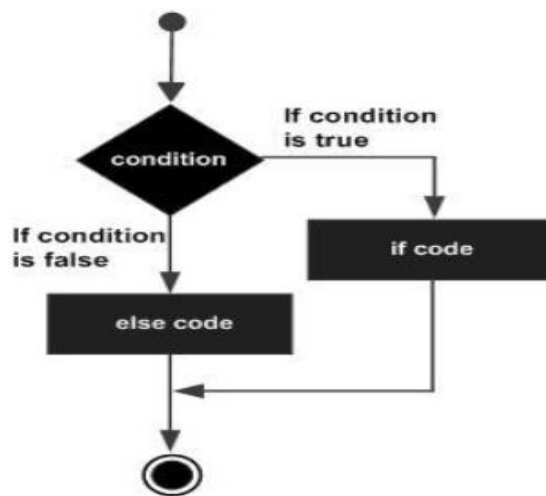
If statements can be combined with else statement to specify what actions to be taken if the condition evaluation is FALSE. Hence If else is known as a two way conditional statements.

There can be at most only one else statement following an if construct, and the else statement is purely an optional statement. Below given is the syntax for if else statement.

Syntax :

```
if expression:
    statement(s)
else:
    statement(s)
```


See the below flow chat to understand the working of if else construct.



Let us consider a sample example

Example

```
x=3
if x%2==0:
    print("even")
else:
    print ("odd")
```

The screenshot shows the Python IDLE environment. The top window, titled 'ifelse.py - E:/Python/ifelse.py (3.9.1)', contains the following code:

```
x=3
if x%2==0:
    print("even")
else:
    print ("odd")
```

The bottom window, titled 'IDLE Shell 3.9.1', shows the output of the program. It displays the Python version and system information, followed by a restart message and the output 'odd'.

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more informatio
n.
>>>
===== RESTART: E:/Python/ifelse.py =====
>>>
odd
>>> |
```

Here the if statement produce FALSE and statement in the else construct executed.

2.1.3. elif Statement

The elif statement is a multi-way decision statement, elif checks multiple expressions for TRUE and execute corresponding block of code. The elif statement is optional same as we discussed in the else statement. However, unlike else, for which there can be at most one statement, there can be an arbitrary number of elif statements following an if. Let us see the syntax of elif statement.

Syntax

```
if expression1:  
    statement(s)  
elif expression2:  
    statement(s)  
elif expression3:  
    statement(s)  
else:  
    statement(s)
```

Now let's consider a simple example program.

Example:

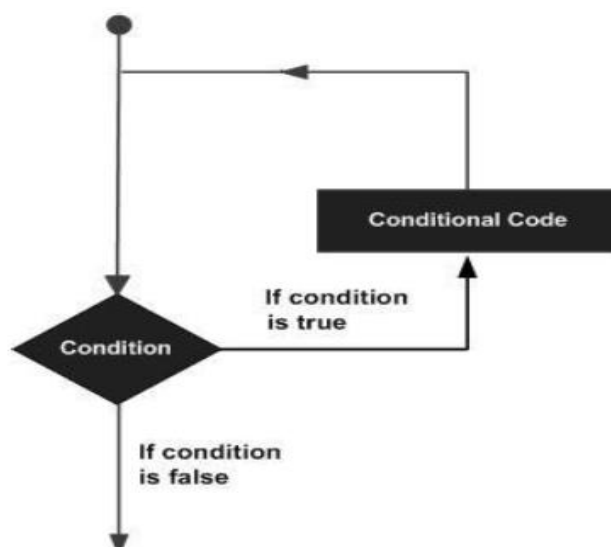
```
x=14  
if x>0:  
    print("positive")  
elif x<0:  
    print("negative")  
else:  
    print("zero")
```

```
elif.py - E:\Python\elif.py (3.9.1)
File Edit Format Run Options Window Help
x=14
if x>0:
    print("positive")
elif x<0:
    print("negative")
else:
    print("zero")
Ln: 8 Col: 0

IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD 64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: E:\Python\elif.py =====
positive
>>> |
Ln: 6 Col: 4
```

2.2. LOOPING STATEMENTS

Looping construct in a programming language is applied when you need to execute a block of code several number of times. In general statements in a function is executed line by line (sequentially), but according to the requirement we may want to repeatedly execute a number of statements. Such complicated program execution paths are possible by the use of looping statements.



Consider the above given flow chat for understanding the program flow of a looping statement. The looping statements repeatedly execute the conditional code till the condition become FALSE. In the below section we will see a number of looping statements with the aid of examples.

2.2.1. While Loop

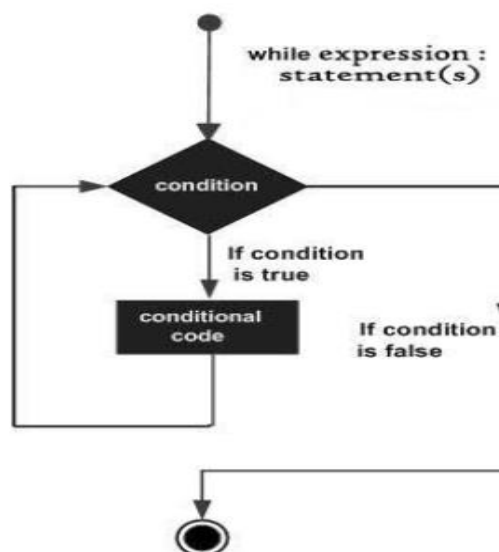
The while loop in python execute the given statements till the loop condition become false. The syntax of while loop is given below.

Syntax:

while expression:

statement(s)

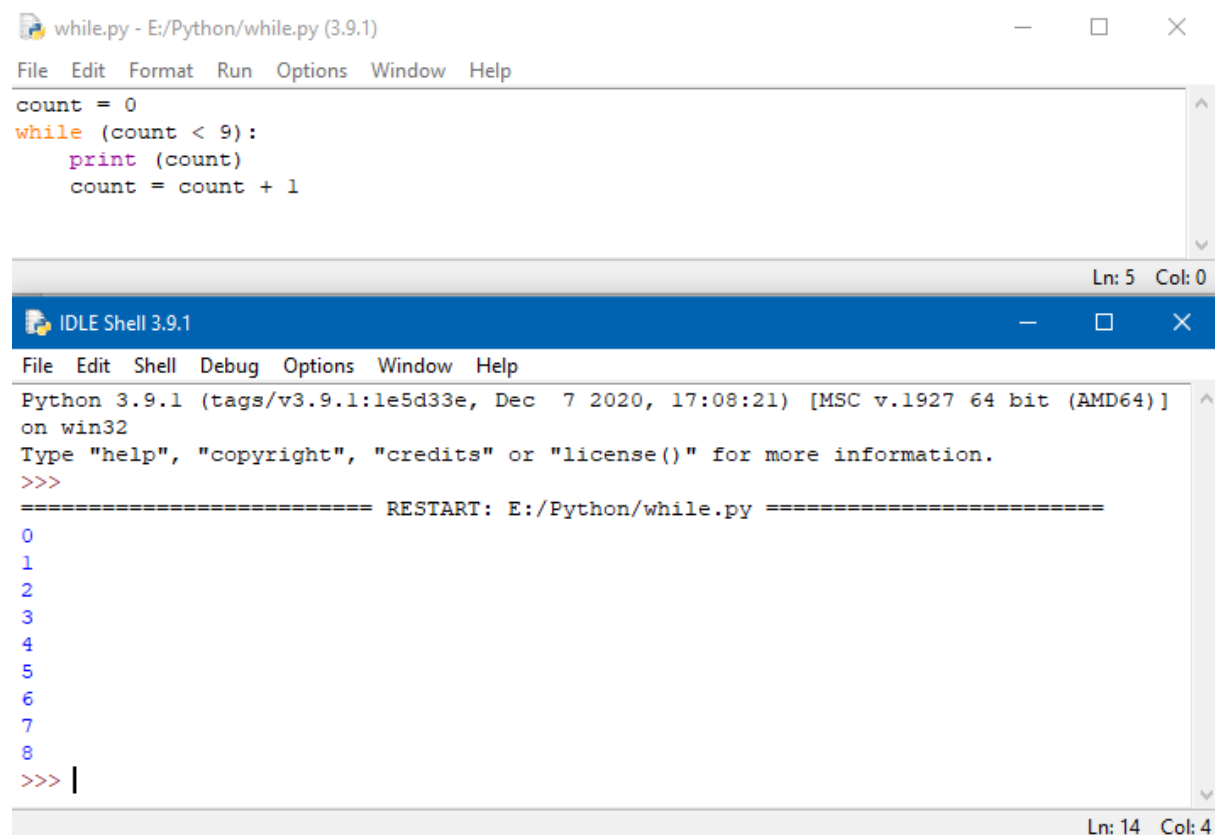
The while statements can be a single statement or a block of statements and condition can be any expression, any non-zero value and true value. The loop statements executes as long as loop condition stays TRUE. When the loop condition produce FALSE, the loop execution stops and program control passes to the statements immediately following the loop. Below flowchart represents the program execution in a while loop. You must be very careful while designing loop conditions, because the loop becomes an infinite loop if the loop condition never becomes FALSE. In such cases there won't be any exit from loop, and the loop statements execute forever, and such loops are called infinite loops.



Now let us consider a simple example program. Here we wish to print numbers from 0 to 8, and we iteratively print the number inside a while block.

Example

```
count = 0
while (count < 9):
    print (count)
    count = count + 1
```



The screenshot displays the Python IDLE 3.9.1 environment. The top window, titled 'while.py - E:/Python/while.py (3.9.1)', contains the following code:

```
count = 0
while (count < 9):
    print (count)
    count = count + 1
```

The bottom window, titled 'IDLE Shell 3.9.1', shows the execution output. It includes the Python version and build information, followed by a restart message for the script. The output of the while loop is a list of numbers from 0 to 8, each on a new line.

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: E:/Python/while.py =====
0
1
2
3
4
5
6
7
8
>>> |
```

Now Let us see an example for infinite loop.

Example

```
while (1):
    print ("Hi")
```

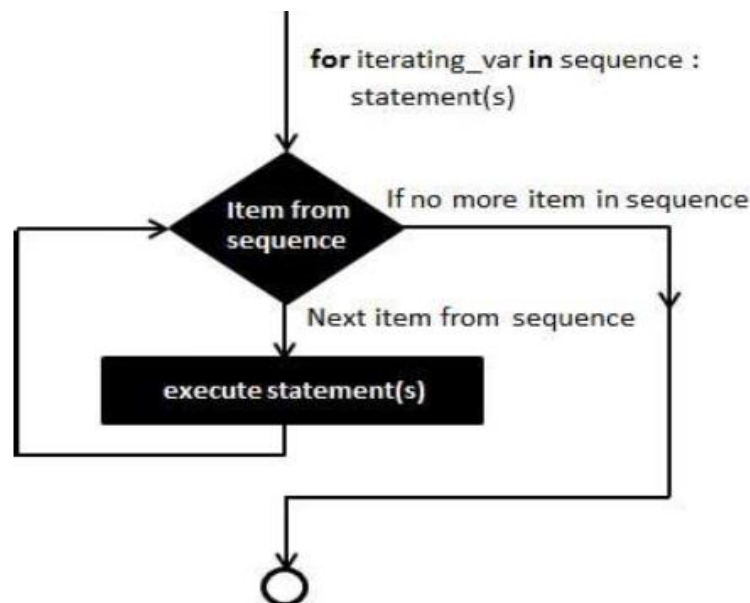
2.2.2. For Loop

The for loop has the ability to iterate on any sequence such as a list or a string. The basic functionality is the same to iterate a block of instructions as long as the loop condition stays TRUE. The syntax of loop is given below.

Syntax:

for iterating_var in sequence:
statements(s)

If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable iterating_var. Next, the statements block is executed. Each item in the list is assigned to iterating_var, and the statement(s) block is executed until the entire sequence is exhausted. Now let us see the program flow through the below given flow chart.



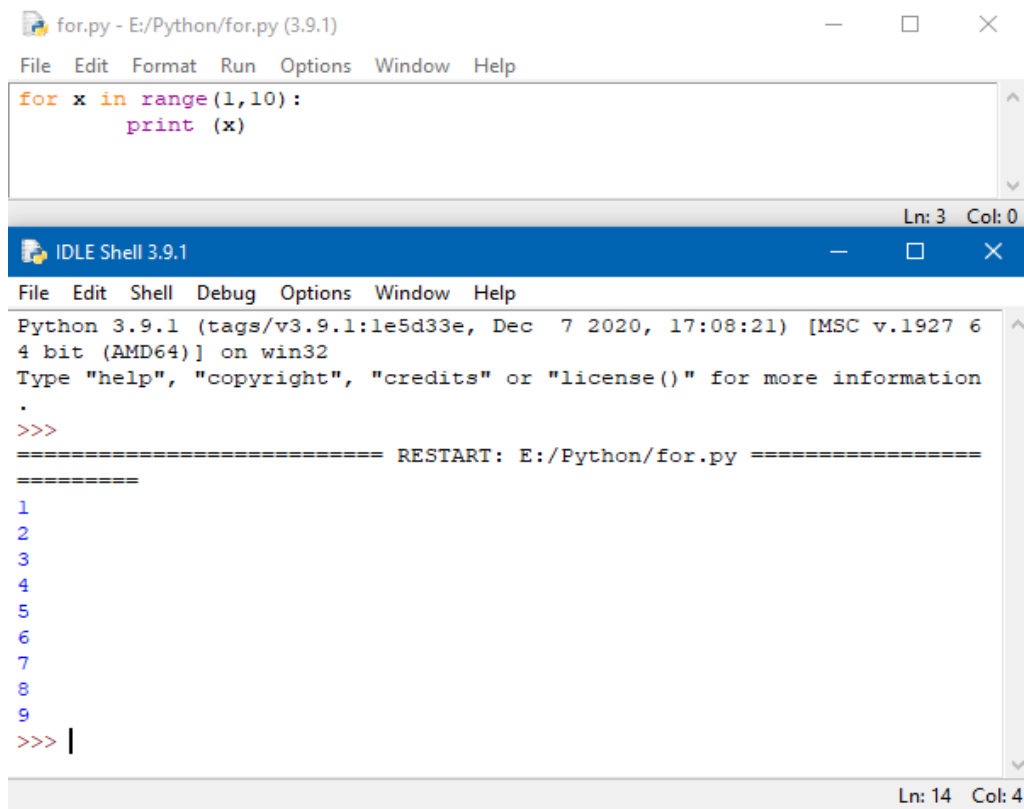
range(stop) or range(start, stop, step)

Range function is most commonly used for loops, this creates list containing arithmetic progression. One condition is that arguments must be plain integers. There are three arguments start, stop and step. If the step argument is not specified, the default will be 1. Similarly if the start argument is omitted, the default will be 0.

Let us see some example of for loop using range function.

Example-1

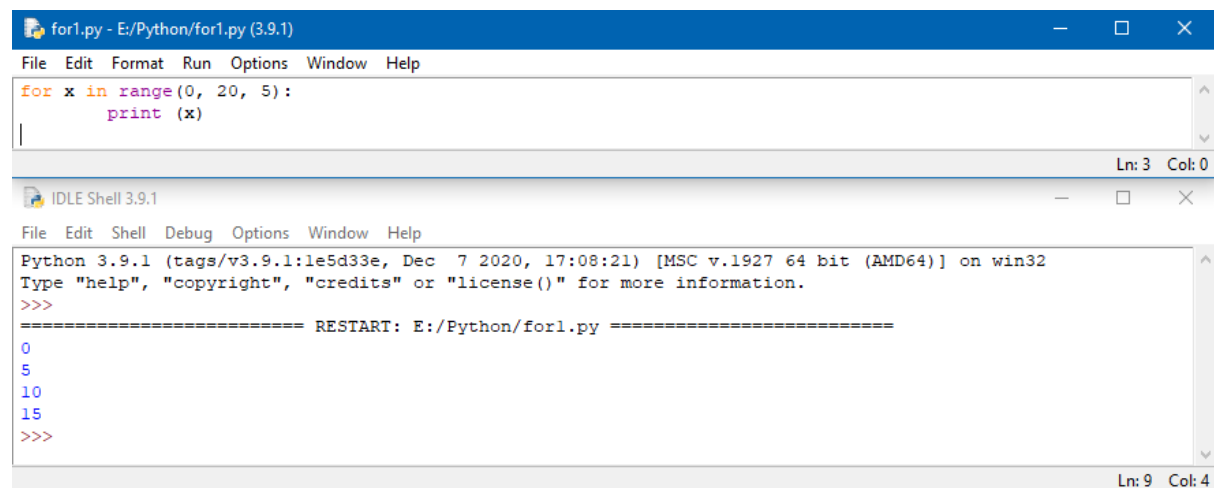
```
for x in range(1,10):  
    print (x)
```



```
for.py - E:/Python/for.py (3.9.1)  
File Edit Format Run Options Window Help  
for x in range(1,10):  
    print (x)  
Ln: 3 Col: 0  
IDLE Shell 3.9.1  
File Edit Shell Debug Options Window Help  
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license()" for more information  
>>>  
===== RESTART: E:/Python/for.py =====  
1  
2  
3  
4  
5  
6  
7  
8  
9  
>>> |  
Ln: 14 Col: 4
```

Example-2

```
for x in range(0, 20, 5):  
    print (x)
```



```
for1.py - E:/Python/for1.py (3.9.1)  
File Edit Format Run Options Window Help  
for x in range(0, 20, 5):  
    print (x)  
Ln: 3 Col: 0  
IDLE Shell 3.9.1  
File Edit Shell Debug Options Window Help  
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: E:/Python/for1.py =====  
0  
5  
10  
15  
>>>  
Ln: 9 Col: 4
```

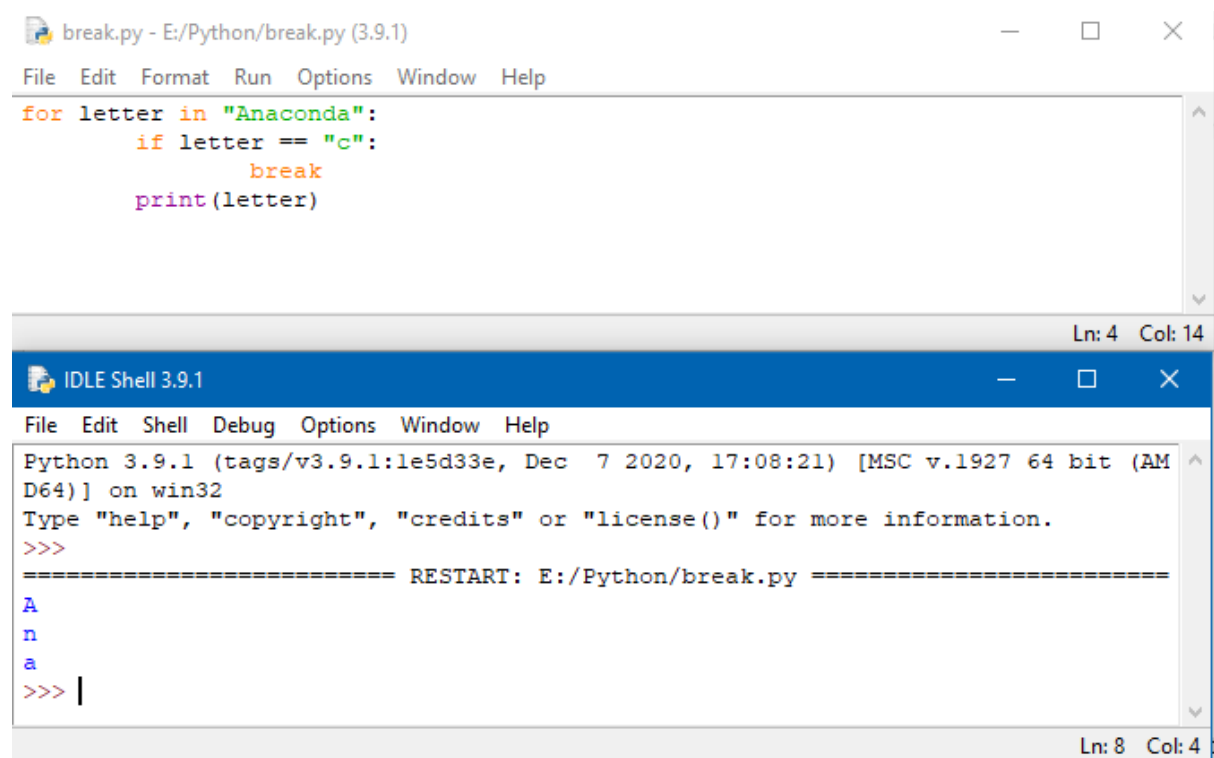
2.2.3. Break Statement

Break statement is used to unconditionally exit from the loop, it break loop execution and pass program control to the statement following the loop. Break can be used with for loop and while loop. Break is normally used when the program has to exit from the loop due to some external conditions. Let us consider one example.

Example

```
for letter in "Anaconda":  
    if letter == "c":  
        break  
    print(letter)
```

This program prints the letters in "Anaconda" and when it encounter "c", the program execution jumps from the for loop.



The screenshot displays the Python IDLE 3.9.1 environment. The top window, titled 'break.py - E:/Python/break.py (3.9.1)', contains the following Python code:

```
for letter in "Anaconda":  
    if letter == "c":  
        break  
    print(letter)
```

The bottom window, titled 'IDLE Shell 3.9.1', shows the execution output. It starts with the Python version and system information, followed by a restart message for the script. The output of the script is:

```
A  
n  
a  
>>> |
```

The shell window status bar at the bottom indicates 'Ln: 8 Col: 4'.

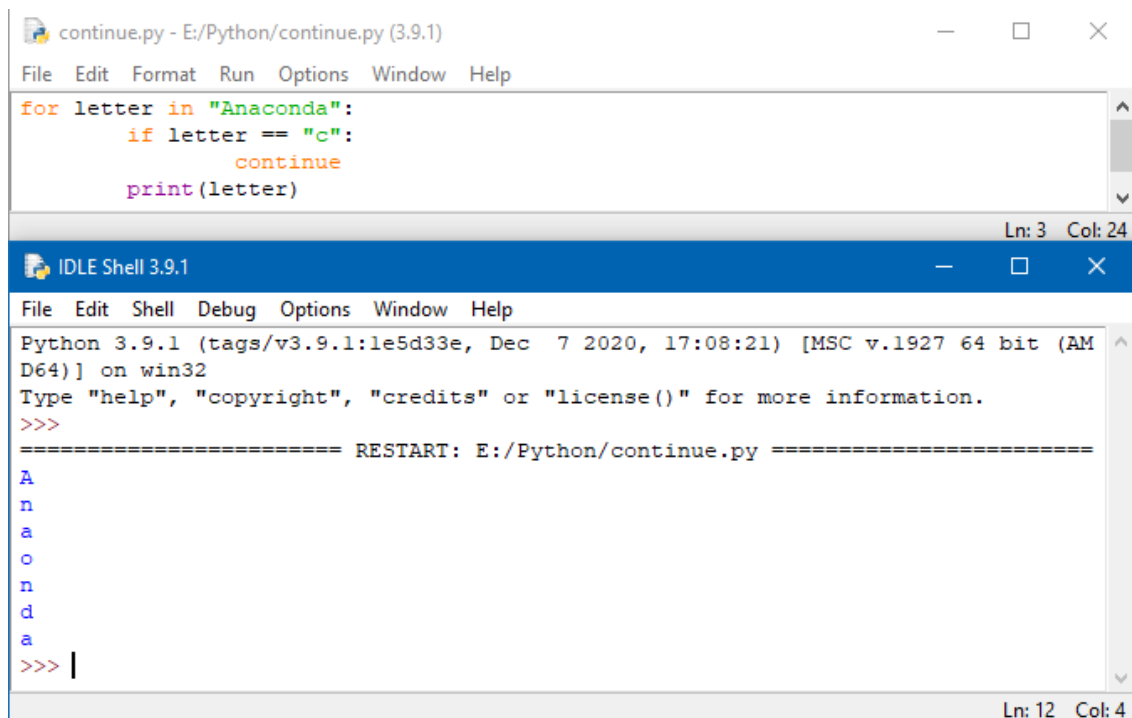
2.2.4. Continue Statement

The continue statement is a loop control statement which skips rest of the statements of the current loop iteration and jumps to the next iteration. The continue statement can also be used with while or for. Let us see one example of continue statement.

Example

```
for letter in "Anaconda":  
    if letter == "c":  
        continue  
    print(letter)
```

In this program, as the loop encounter the condition to check letter “c” and found true, the current iteration to print the letter is aborted and next iteration is started.



The screenshot displays the Python IDLE 3.9.1 environment. The top window, titled 'continue.py - E:/Python/continue.py (3.9.1)', contains the following code:

```
for letter in "Anaconda":  
    if letter == "c":  
        continue  
    print(letter)
```

The bottom window, titled 'IDLE Shell 3.9.1', shows the execution output. It includes the standard Python startup message and a restart notice for the script. The output of the script is the string 'Anaconda', with each character printed on a new line, except for the letter 'c' which is skipped.

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: E:/Python/continue.py =====  
A  
n  
a  
o  
n  
d  
a  
>>> |
```

2.2.5. Nested Loop

Nested loop constructs are to program one loop inside another loop, you can have multiple for loops, or while loops inside loop constructs. Let us see some of the nested loops syntax.

Syntax 1: Nested For Loop

for iterating_var in sequence:

for iterating_var in sequence:

statements(s)

statements(s)

Syntax 1: Nested While Loop

while expression:

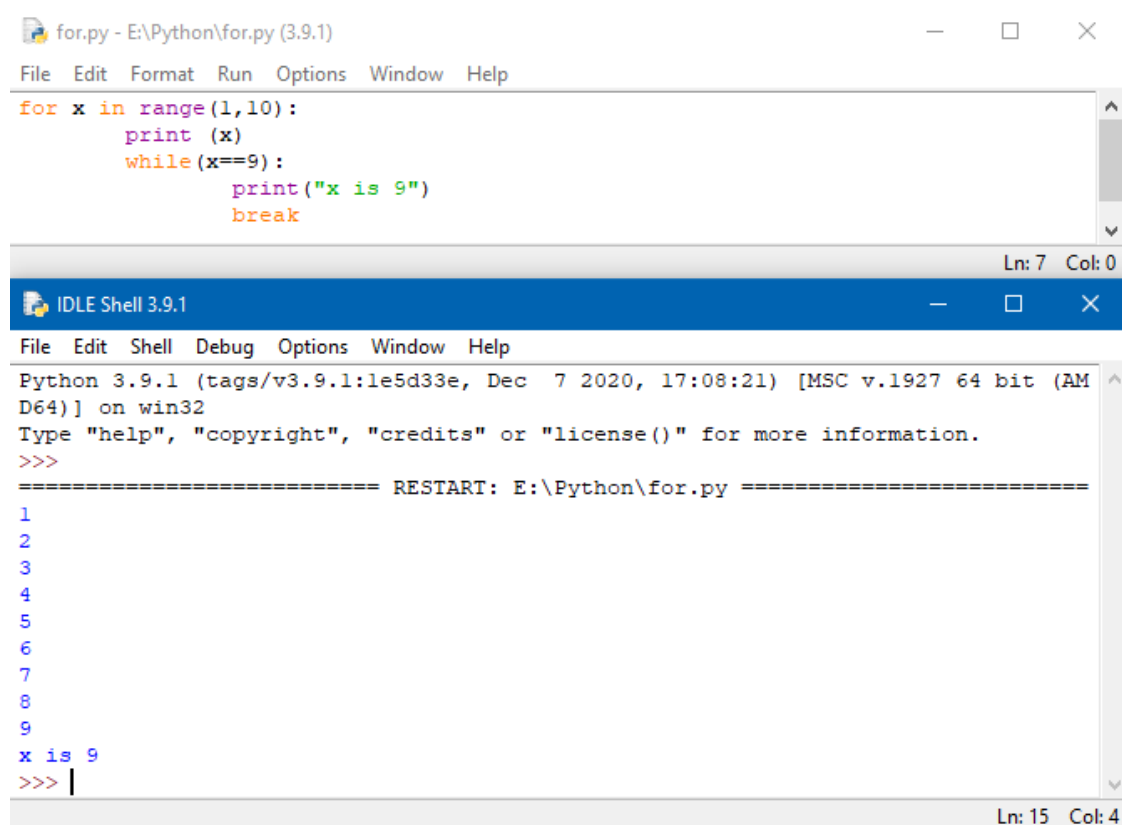
while expression:

statement(s)

statement(s)

Example:

```
for x in range(1,10):  
    print (x)  
    while(x==9):  
        print("x is 9")  
        break
```



The screenshot displays a Python IDE window titled 'for.py - E:\Python\for.py (3.9.1)'. The code editor contains the following Python code:

```
for x in range(1,10):  
    print (x)  
    while (x==9):  
        print("x is 9")  
        break
```

Below the code editor is the 'IDLE Shell 3.9.1' window. It shows the output of the program after execution. The first nine lines of the output are the numbers 1 through 9, corresponding to the first loop. The tenth line is 'x is 9', which is the output of the nested while loop when x equals 9. The prompt '>>>' is visible at the end of the line.

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: E:\Python\for.py =====  
1  
2  
3  
4  
5  
6  
7  
8  
9  
x is 9  
>>> |
```

CHAPTER 3

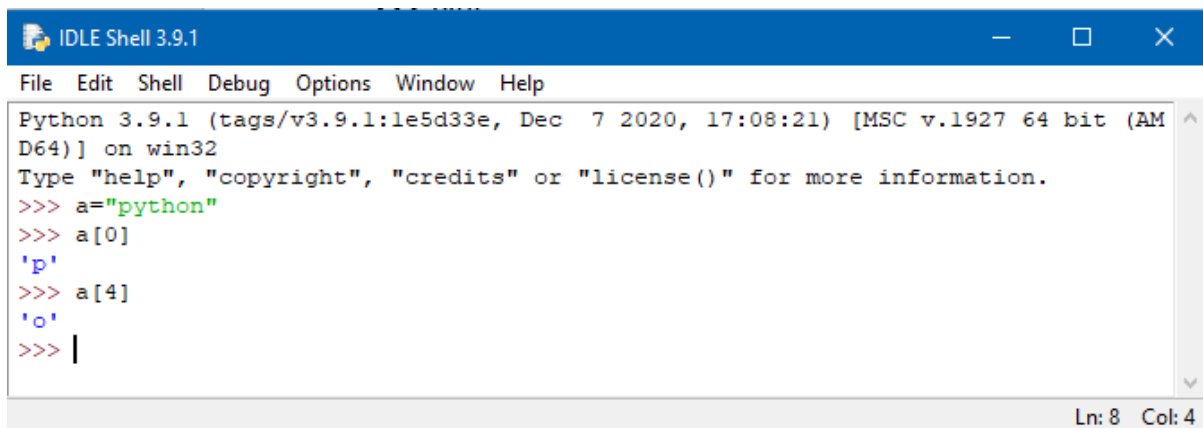
STRINGS, LIST, TUPLE AND DICTIONARY

3.1. STRINGS

Strings are consecutive sequence of characters. And string is a compound data type which composed of smaller pieces. Subscripts can be used to access individual characters of a string. Subscripts are otherwise called as index. The index is always a positive or a negative integer. A index in python starts from 0. Now let us consider an example of a string and how the individual characters are accessed via index.

Example:

```
>>> a="python"
>>> a[0]
'p'
>>> a[4]
'o'
>>>
```

A screenshot of the IDLE Shell 3.9.1 window. The title bar says "IDLE Shell 3.9.1". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The shell area shows the following text: "Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32", "Type 'help()', 'copyright()', 'credits()' or 'license()' for more information.", and the code execution output: ">>> a='python'", ">>> a[0]", "'p'", ">>> a[4]", "'o'", and ">>> |". The status bar at the bottom right shows "Ln: 8 Col: 4".

Here a is a variable, contains the string “python”, a[0] is used to access the first character of the string. If you consider negative index, a[-6] will output “p” and a[-1] will output “n”, you may try this in your python IDLE.

Positive index access the string from the beginning whereas the negative index access the string from the end. index value 0 or –n access the first element (n is length of the string).

String	P	Y	T	H	O	N
Positive index	0	1	2	3	4	5
Negative index	-6	-5	-4	-3	-2	-1

Although you can access each characters of the string using the index variables, they cannot be changed after creation. Hence strings are immutable.

```

Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> car="toyota"
>>> car[1]="p"
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    car[1]="p"
TypeError: 'str' object does not support item assignment
>>> |

```

Now let us see some of the string operations and manipulations.

3.1.1. Length of string

The length of the string is the total number of characters in that string and can be found using the “len” function. Let us see “len” operation on a simple string.

Example:

```

>>> car="toyota"
>>> len(car)
6
>>>

```

```

Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> car="toyota"
>>> len(car)
6
>>> |

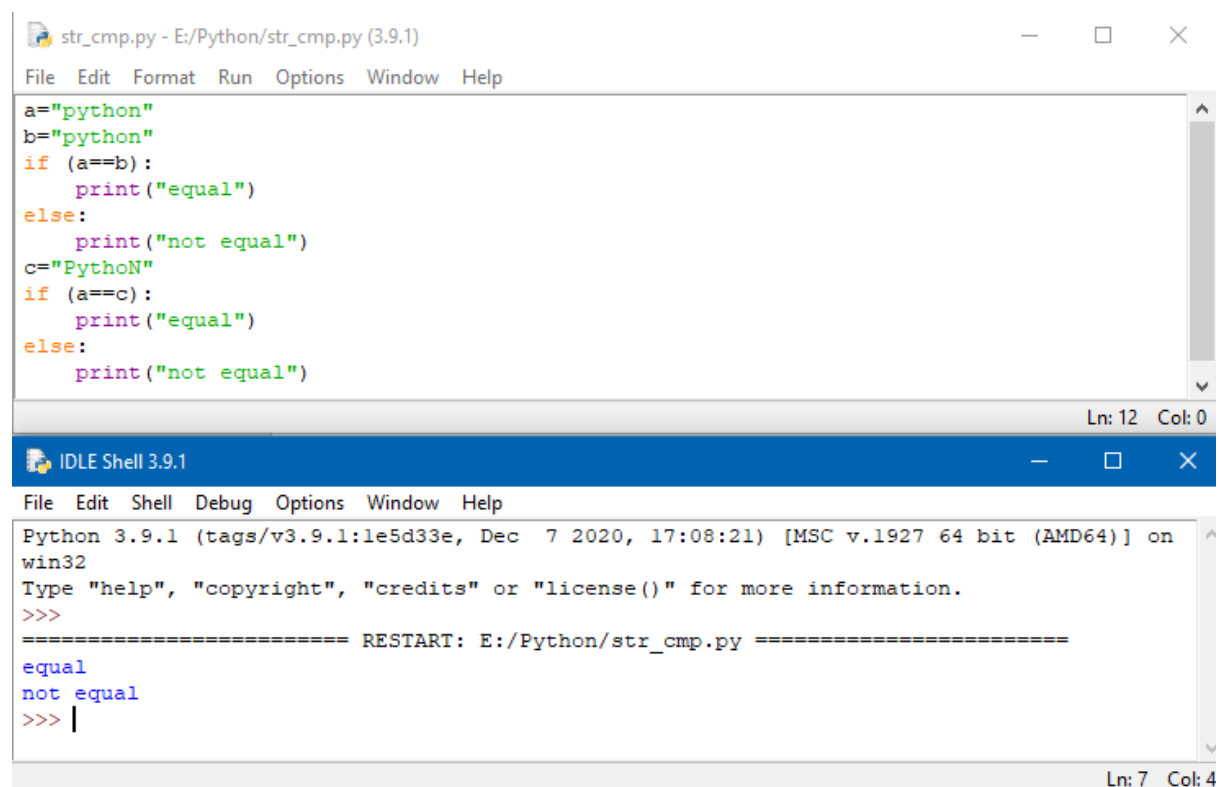
```

3.1.2. String Comparison

String comparison is used to compare two string to find if they are equal or not. You may use logical operators to compare strings. Since python is case sensitive, two string similar but differ in case will be considered as not equal. Let us see some simple example.

Example:

```
a="python"
b="python"
if (a==b):
    print("equal")
else:
    print("not equal")
c="PythoN"
if (a==c):
    print("equal")
else:
    print("not equal")
```



The screenshot displays two windows from an IDE. The top window, titled 'str_cmp.py - E:/Python/str_cmp.py (3.9.1)', contains the following Python code:

```
a="python"
b="python"
if (a==b):
    print("equal")
else:
    print("not equal")
c="PythoN"
if (a==c):
    print("equal")
else:
    print("not equal")
```

The bottom window, titled 'IDLE Shell 3.9.1', shows the execution output of the script. It displays the Python version and environment information, followed by the output of the script's print statements:

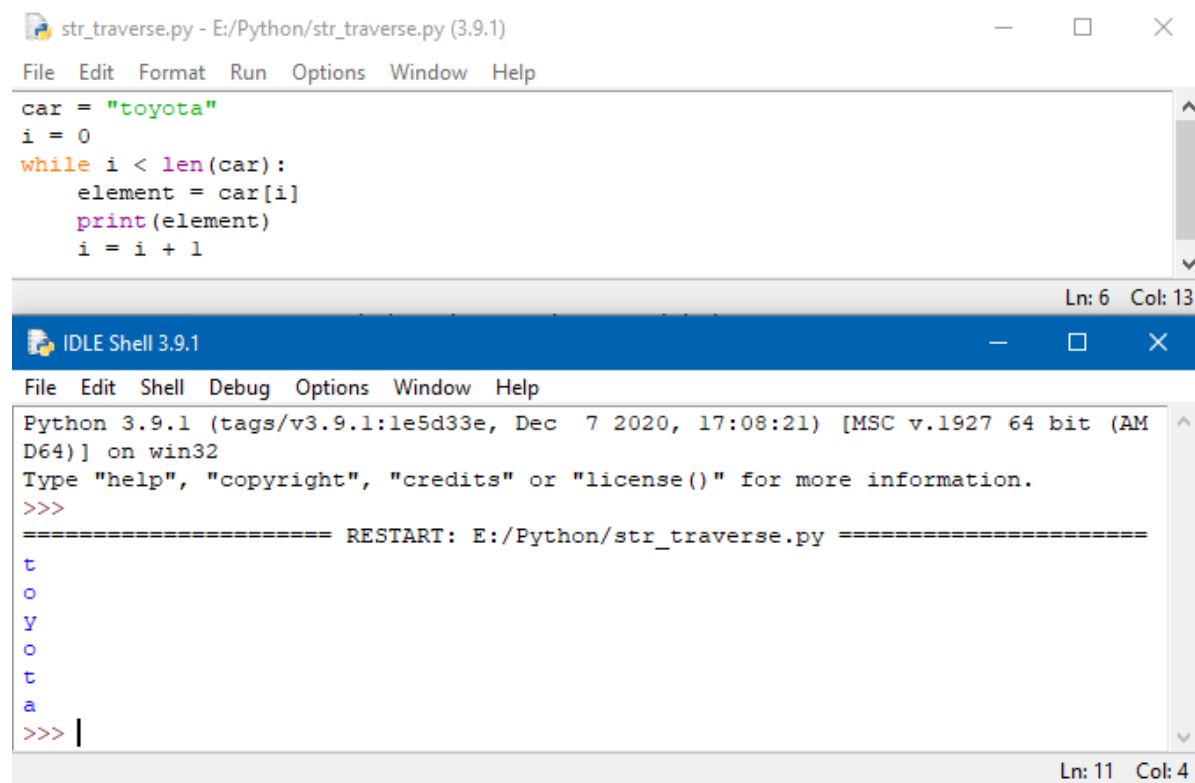
```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: E:/Python/str_cmp.py =====
equal
not equal
>>> |
```

3.1.3. String Traversal

String traversal refers to the act of accessing the string elements one by one, using the string index. You can use the idea of loop to traverse the string. Let us see one example of string traversal using while loop.

Example:

```
car = "toyota"
i = 0
while i < len(car):
    element = car[i]
    print(element)
    i = i + 1
```



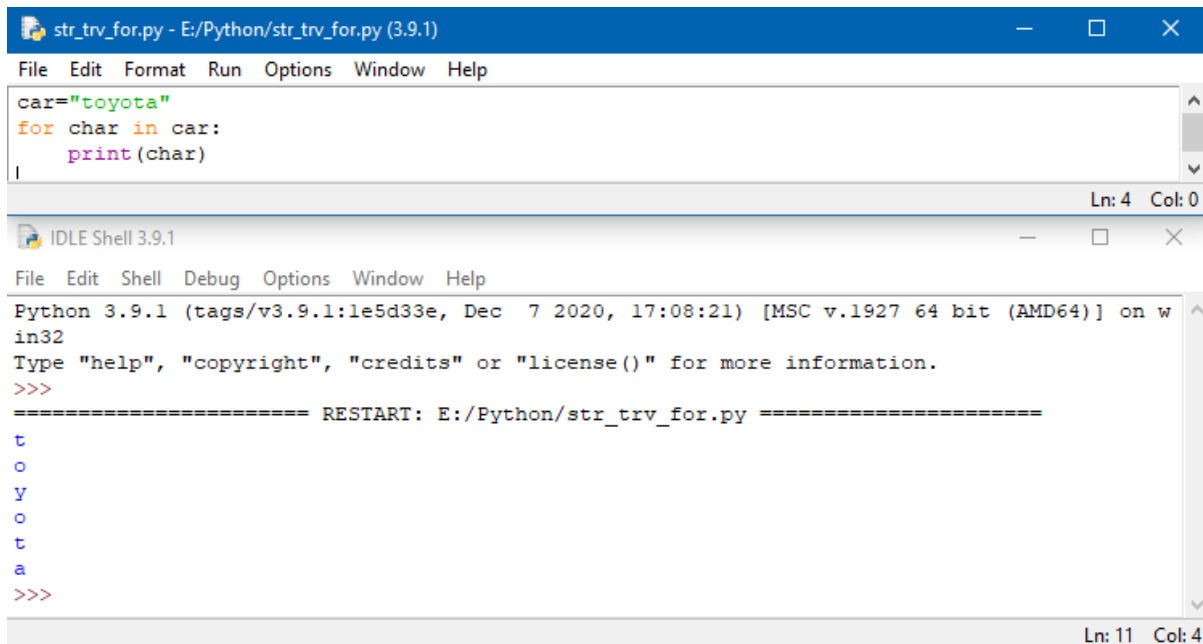
The screenshot displays a Python IDE window titled 'str_traverse.py - E:/Python/str_traverse.py (3.9.1)'. The code editor contains the following Python code:

```
car = "toyota"
i = 0
while i < len(car):
    element = car[i]
    print(element)
    i = i + 1
```

Below the code editor is the 'IDLE Shell 3.9.1' window. It shows the output of the program, which prints the characters of the string 'toyota' one by one on separate lines. The shell also displays the Python version (3.9.1) and the file path (E:/Python/str_traverse.py).

You may also consider a simpler way of traversing using for loop as given below.

```
car="toyota"
for char in car:
    print(char)
```



The screenshot shows the Python IDLE environment. The top window, titled 'str_trv_for.py - E:/Python/str_trv_for.py (3.9.1)', contains the following code:

```
car="toyota"
for char in car:
    print(char)
```

The bottom window, titled 'IDLE Shell 3.9.1', shows the execution output:

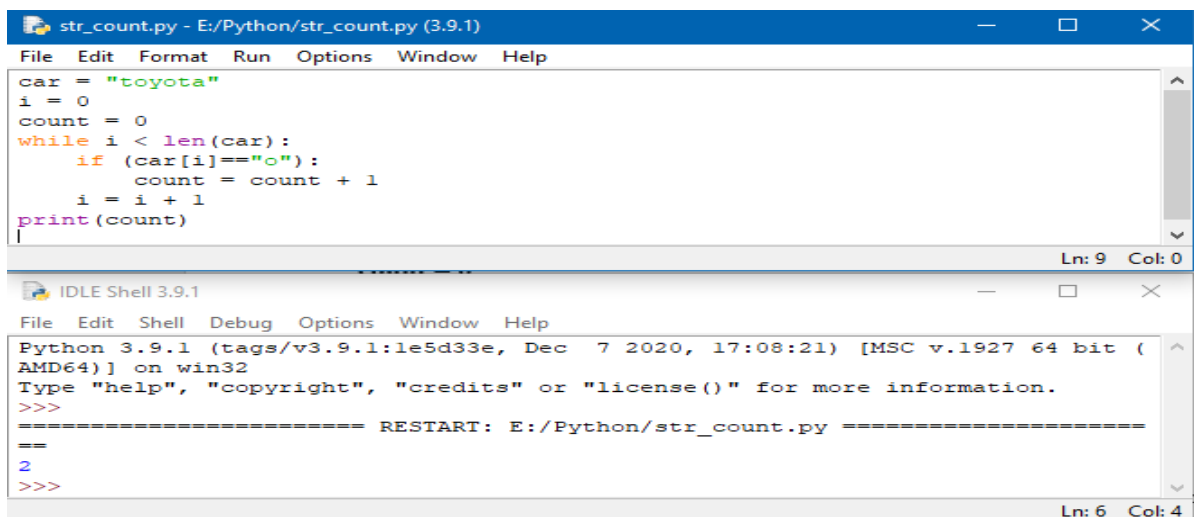
```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: E:/Python/str_trv_for.py =====
t
o
y
o
t
a
>>>
```

3.1.4. String traversal and counting

You can traverse the string and count the number of occurrence of specific characters in the string. Let us see one simple example.

Example:

```
car = "toyota"
i = 0
count = 0
while i < len(car):
    if (car[i]=="o"):
        count = count + 1
    i = i + 1
print(count)
```



The screenshot shows the Python IDLE environment. The top window, titled 'str_count.py - E:/Python/str_count.py (3.9.1)', contains the following code:

```
car = "toyota"
i = 0
count = 0
while i < len(car):
    if (car[i]=="o"):
        count = count + 1
    i = i + 1
print(count)
```

The bottom window, titled 'IDLE Shell 3.9.1', shows the execution output:

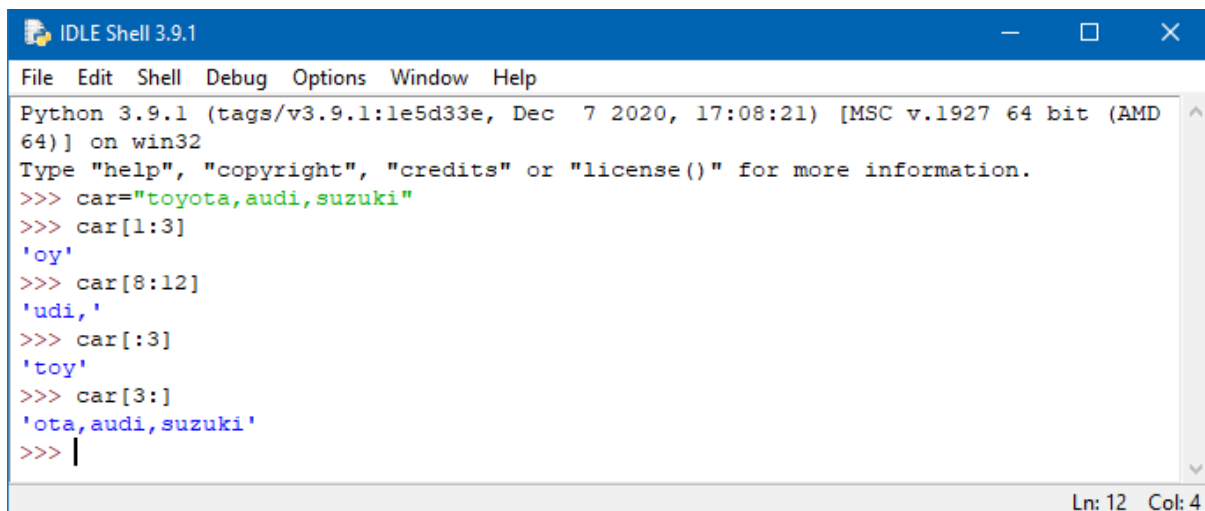
```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: E:/Python/str_count.py =====
2
>>>
```

This program checks for the occurrence of character “o” in the string while traversing and count them, finally the program prints the number of occurrence of character “o”

3.1.5. String Slices

String slices refers to substrings of s string. You can consider it as a segment of a given string. The substring between nth character to mth character can be found using the operator [n:m]. let us see this operation with the aid of some simple examples.

Example:

A screenshot of the IDLE Shell 3.9.1 window. The window has a blue title bar with the text 'IDLE Shell 3.9.1' and standard window controls. Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following Python code and its output:

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD 64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> car="toyota,audi,suzuki"
>>> car[1:3]
'oy'
>>> car[8:12]
'udi,'
>>> car[:3]
'toy'
>>> car[3:]
'ota,audi,suzuki'
>>> |
```

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

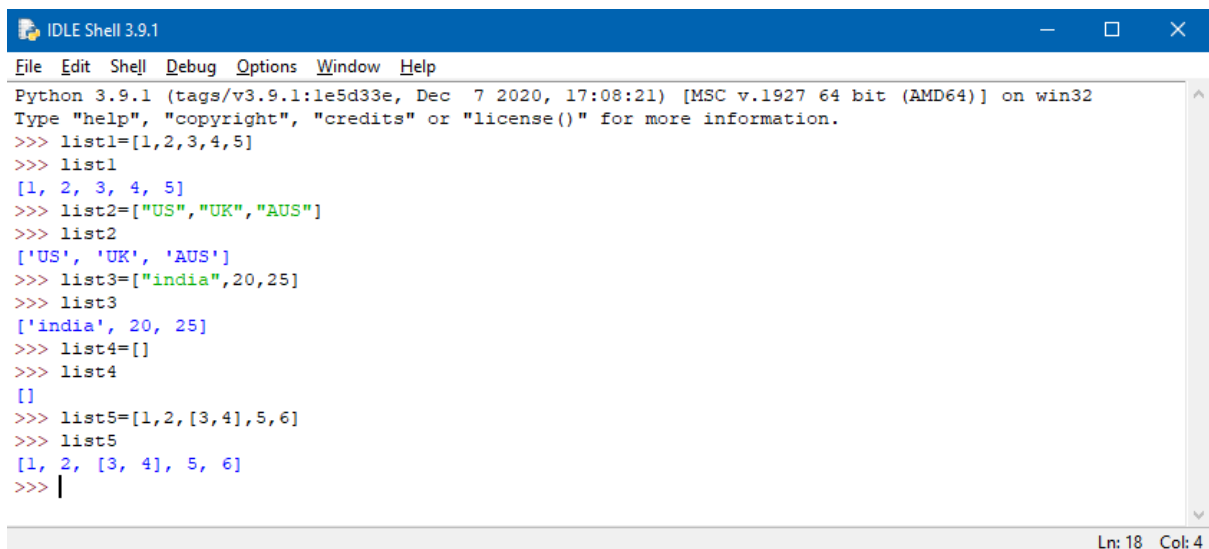
In the above example, you can see how the operator [n:m] produced substrings, you can see that avoiding the first operand produced a substring from the beginning of the string, and avoiding last operand produced the remaining substring till the end of string.

3.2. List

A list is a set of values enclosed in [], which are individually addressable by an index. Each items in the list is called as its elements. Strings are ordered set of characters, but the elements in the list can have any types. Now let us consider a simple example of list.

Example:

```
>>> list1=[1,2,3,4,5]
>>> list1
[1, 2, 3, 4, 5]
>>> list2=["US","UK","AUS"]
>>> list2
['US', 'UK', 'AUS']
>>> list3=["india",20,25]
>>> list3
['india', 20, 25]
>>> list4=[ ]
>>> list4
[]
>>> list5=[1,2,[3,4],5,6]
>>> list5
[1, 2, [3, 4], 5, 6]
```

A screenshot of the IDLE Shell 3.9.1 window. The window has a blue title bar with the text 'IDLE Shell 3.9.1' and standard window controls. Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the Python 3.9.1 prompt and the execution of the example code. The output matches the code shown in the previous block. At the bottom right of the window, the status bar shows 'Ln: 18 Col: 4'.

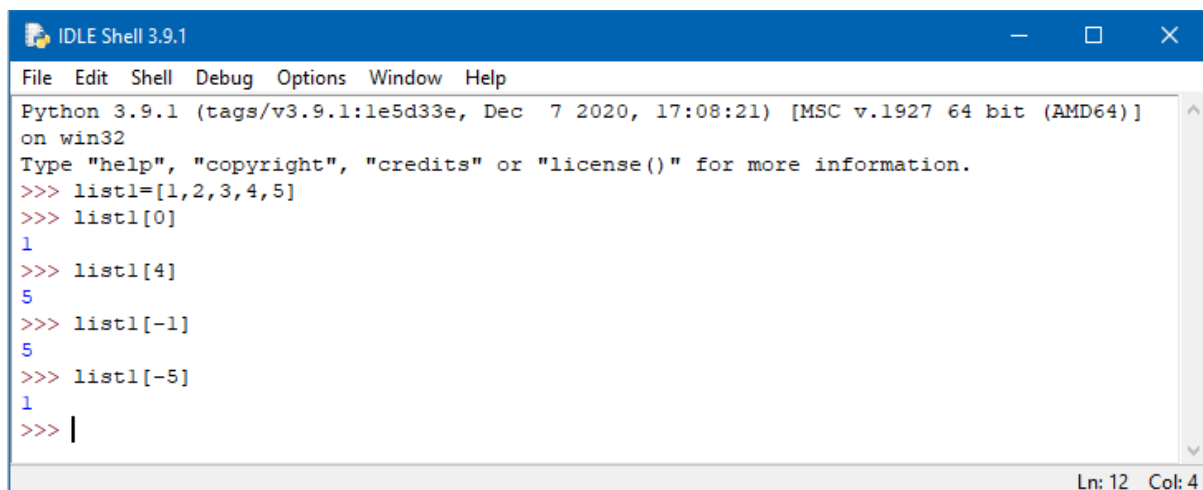
Here in this example, list 1 is an integer list, list2 is a list of strings, list3 has both strings and integers as elements. List 4 represents an empty list, with no elements and list5 is called nested list, that is a list with in a list.

3.2.1. Accessing List Elements

Now let us see how we can access the elements of a list, you can use the bracket operator with the index inside to access the list elements. This is very similar to that we have seen in the case of strings. Index of a list start from 0. If an index has a negative value, it counts backward from the end of the list. Let us see some examples.

Example:

```
>>> list1=[1,2,3,4,5]
>>> list1[0]
1
>>> list1[4]
5
>>> list1[-1]
5
>>> list1[-5]
1
>>>
```

A screenshot of the IDLE Shell 3.9.1 window. The window has a blue title bar with the text 'IDLE Shell 3.9.1' and standard window controls. Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following code and output:

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> list1=[1,2,3,4,5]
>>> list1[0]
1
>>> list1[4]
5
>>> list1[-1]
5
>>> list1[-5]
1
>>> |
```

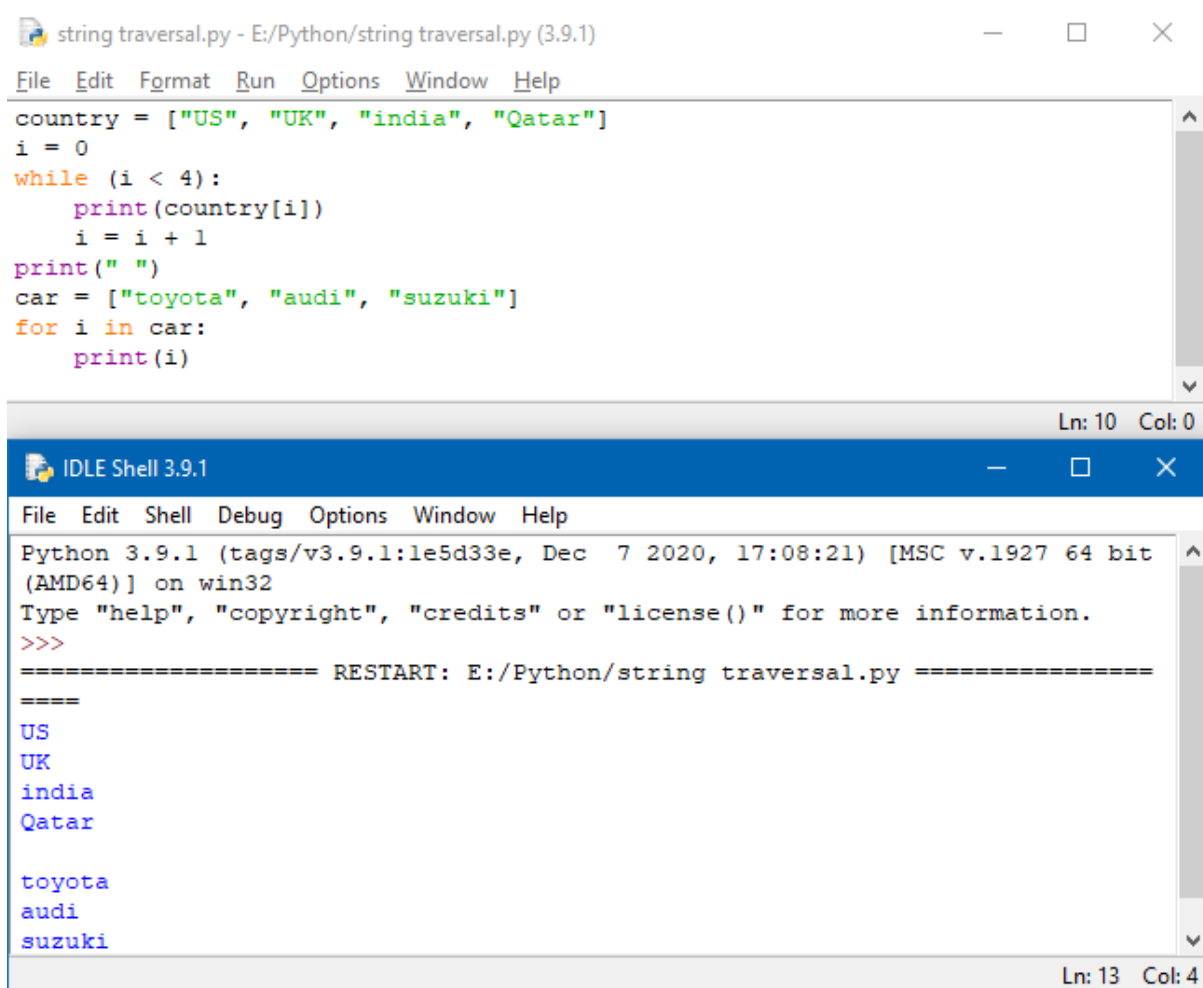
The status bar at the bottom right indicates 'Ln: 12 Col: 4'.

3.2.2. List Traversal

List traversal is very similar to that we have seen in strings, traversal refers to visiting list elements one by one. We use the help of loops to achieve this. Let us consider one example.

Example:

```
country = ["US", "UK", "india", "Qatar"]
i = 0
while (i < 4):
    print(country[i])
    i = i + 1
print(" ")
car = ["toyota", "audi", "suzuki"]
for i in car:
    print(i)
```



The screenshot displays a Python IDE with two windows. The top window, titled 'string traversal.py - E:/Python/string traversal.py (3.9.1)', contains the following code:

```
country = ["US", "UK", "india", "Qatar"]
i = 0
while (i < 4):
    print(country[i])
    i = i + 1
print(" ")
car = ["toyota", "audi", "suzuki"]
for i in car:
    print(i)
```

The bottom window, titled 'IDLE Shell 3.9.1', shows the output of the program after execution. It displays the elements of the 'country' list followed by a blank line, and then the elements of the 'car' list.

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: E:/Python/string traversal.py =====
=====
US
UK
india
Qatar

toyota
audi
suzuki
```

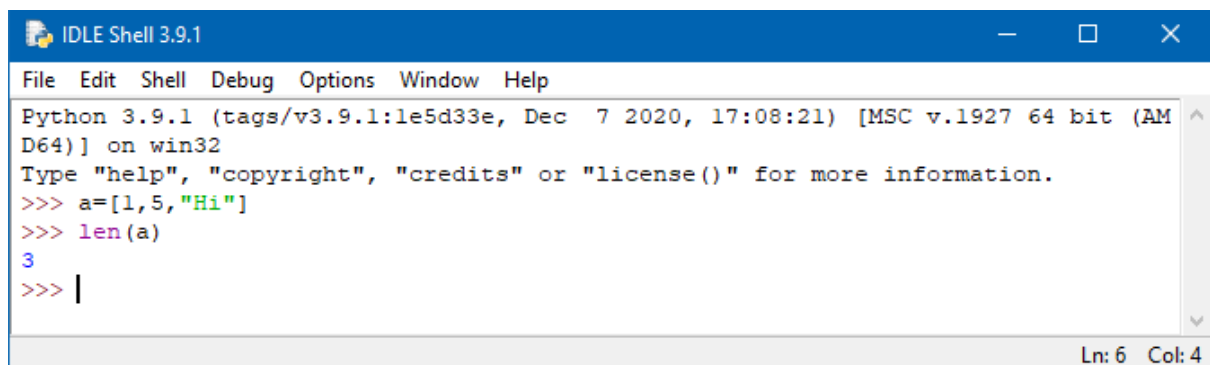
The above program shows the usage of while and for loop to traverse through the list elements and print them.

3.2.3. Length of a list

Very similar to what we have seen in strings, the “len” function applied on a list will output the number of elements in the list.

Example:

```
>>> a=[1,5,"Hi"]
>>> len(a)
3
>>>
```

A screenshot of the IDLE Shell 3.9.1 window. The title bar is blue with the text 'IDLE Shell 3.9.1' and standard window controls. The menu bar includes 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the Python 3.9.1 version information and a prompt. The user has entered the following code:

```
>>> a=[1,5,"Hi"]
>>> len(a)
3
>>> |
```

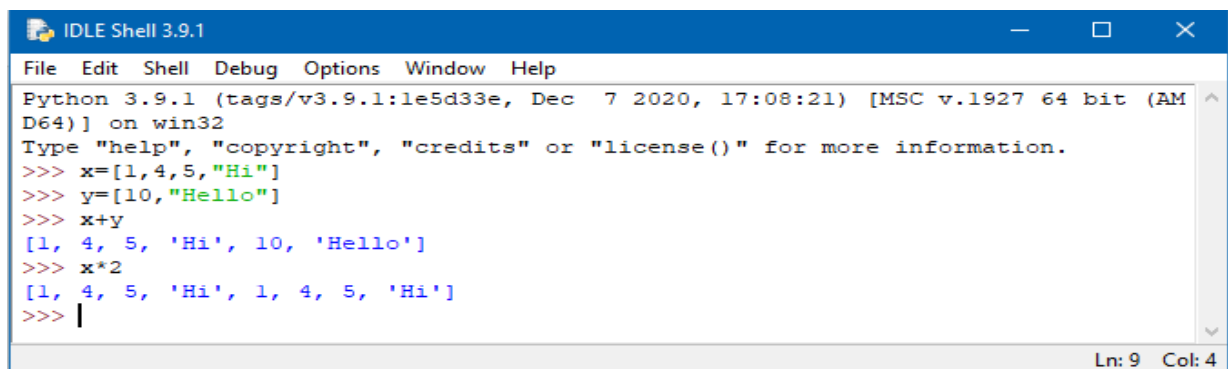
 The status bar at the bottom right indicates 'Ln: 6 Col: 4'.

3.2.4. Operations on a List

For concatenating two list, one can use + operator. The * operator will repeats a list a specified number of times. Let us see both these in action

Example:

```
>>> x=[1,4,5,"Hi"]
>>> y=[10,"Hello"]
>>> x+y
[1, 4, 5, 'Hi', 10, 'Hello']
>>> x*2
[1, 4, 5, 'Hi', 1, 4, 5, 'Hi']
```

A screenshot of the IDLE Shell 3.9.1 window. The title bar is blue with the text 'IDLE Shell 3.9.1' and standard window controls. The menu bar includes 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the Python 3.9.1 version information and a prompt. The user has entered the following code:

```
>>> x=[1,4,5,"Hi"]
>>> y=[10,"Hello"]
>>> x+y
[1, 4, 5, 'Hi', 10, 'Hello']
>>> x*2
[1, 4, 5, 'Hi', 1, 4, 5, 'Hi']
>>> |
```

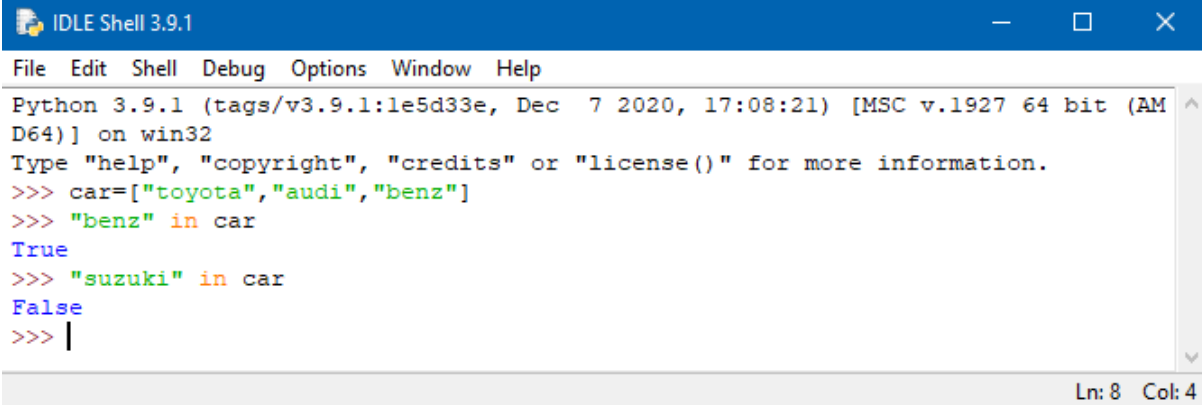
 The status bar at the bottom right indicates 'Ln: 9 Col: 4'.

3.2.5. List Membership

List membership refer to the presence of an element in the list. "in", is a Boolean operator that test the membership of an element in the given list. It will return TRUE if the element is present in the list. Let us see a simple example

Example:

```
>>> car=["toyota","audi","benz"]
>>> "benz" in car
True
>>> "suzuki" in car
False
>>>
```

A screenshot of the IDLE Shell 3.9.1 window. The title bar is blue with the text 'IDLE Shell 3.9.1' and standard window controls. The menu bar includes 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following Python code and its output:

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> car=["toyota","audi","benz"]
>>> "benz" in car
True
>>> "suzuki" in car
False
>>> |
```

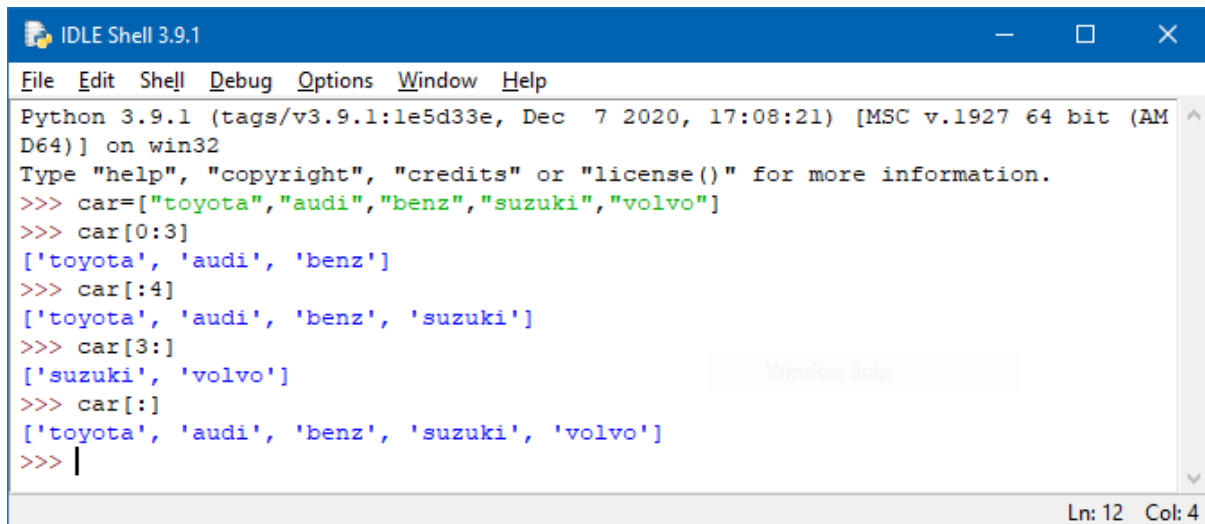
 The status bar at the bottom right indicates 'Ln: 8 Col: 4'.

3.2.6. List Slices

List slices are very similar to what we have seen in string slices, list slices refers to the sub-lists. The slice starts at the beginning, when you omit the first index. If you omit the second, the slice goes to the end and when you omit both, the slice is becomes a copy of the original list.

Example:

```
>>> car=["toyota","audi","benz","suzuki","volvo"]
>>> car[0:3]
['toyota', 'audi', 'benz']
>>> car[:4]
['toyota', 'audi', 'benz', 'suzuki']
>>> car[3:]
['suzuki', 'volvo']
>>> car[:]
['toyota', 'audi', 'benz', 'suzuki', 'volvo']
```



```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> car=["toyota","audi","benz","suzuki","volvo"]
>>> car[0:3]
['toyota', 'audi', 'benz']
>>> car[:4]
['toyota', 'audi', 'benz', 'suzuki']
>>> car[3:]
['suzuki', 'volvo']
>>> car[:]
['toyota', 'audi', 'benz', 'suzuki', 'volvo']
>>> |
```

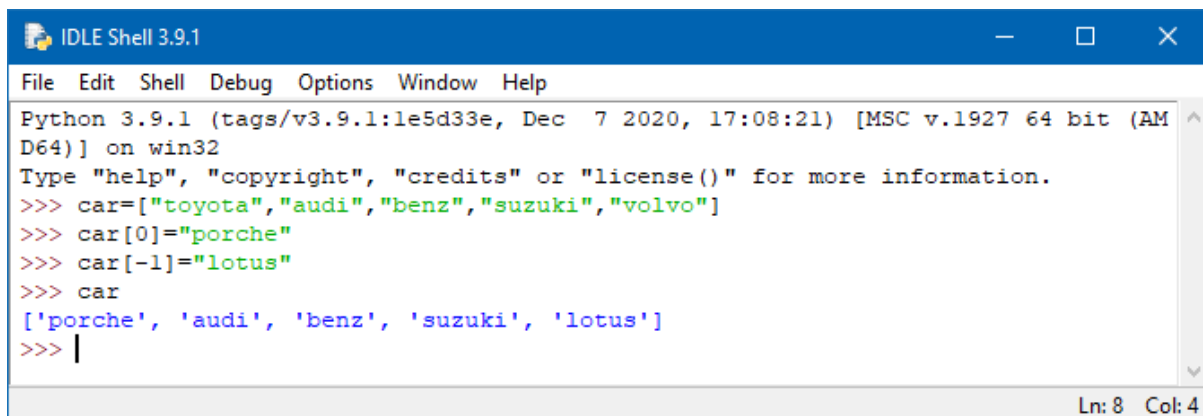
Ln: 12 Col: 4

3.2.7. Are Lists Mutable?

Yes!, lists are mutable unlike strings, you can alter an element of the list after its creation. You can use the brackets operator along with index value to make the changes to the index pointing element. Let us see one example.

Example:

```
>>> car=["toyota","audi","benz","suzuki","volvo"]
>>> car[0]="porche"
>>> car[-1]="lotus"
>>> car
['porche', 'audi', 'benz', 'suzuki', 'lotus']
>>>
```



```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> car=["toyota","audi","benz","suzuki","volvo"]
>>> car[0]="porche"
>>> car[-1]="lotus"
>>> car
['porche', 'audi', 'benz', 'suzuki', 'lotus']
>>> |
```

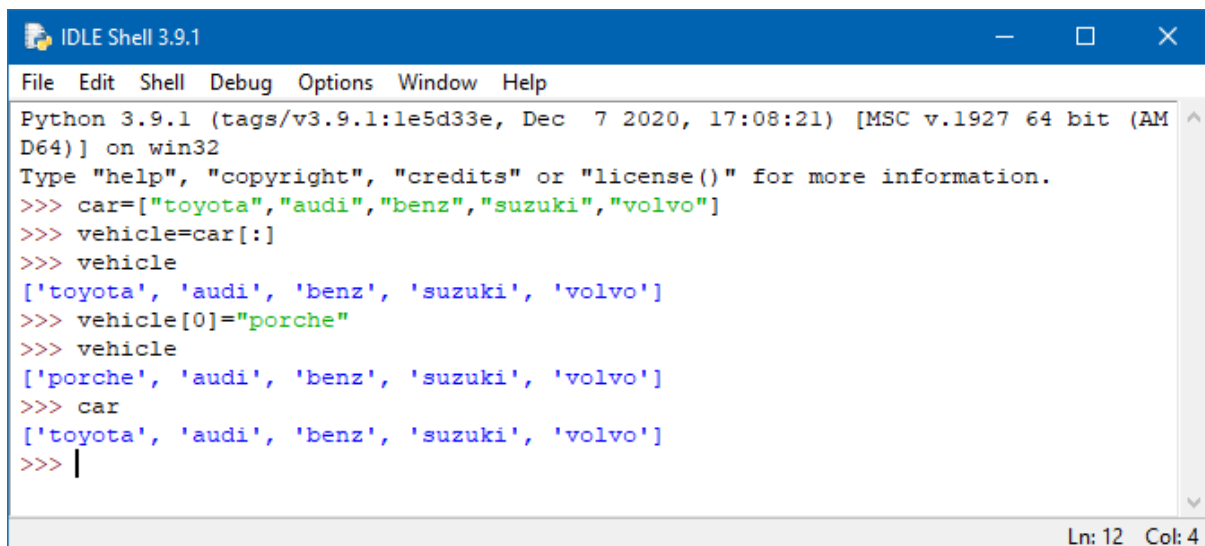
Ln: 8 Col: 4

3.2.8. Clone List

Cloning a list refers to the act of duplicating a list, let us see one example. This usually helps when we need to keep a copy of a list and to modify it.

Example:

```
>>> car=["toyota","audi","benz","suzuki","volvo"]
>>> vehicle=car[:]
>>> vehicle
['toyota', 'audi', 'benz', 'suzuki', 'volvo']
>>> vehicle[0]="porche"
>>> vehicle
['porche', 'audi', 'benz', 'suzuki', 'volvo']
>>> car
['toyota', 'audi', 'benz', 'suzuki', 'volvo']
>>>
```

A screenshot of the IDLE Shell 3.9.1 window. The window has a blue title bar with the text 'IDLE Shell 3.9.1' and standard window controls. Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following code and its output:

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> car=["toyota","audi","benz","suzuki","volvo"]
>>> vehicle=car[:]
>>> vehicle
['toyota', 'audi', 'benz', 'suzuki', 'volvo']
>>> vehicle[0]="porche"
>>> vehicle
['porche', 'audi', 'benz', 'suzuki', 'volvo']
>>> car
['toyota', 'audi', 'benz', 'suzuki', 'volvo']
>>> |
```

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

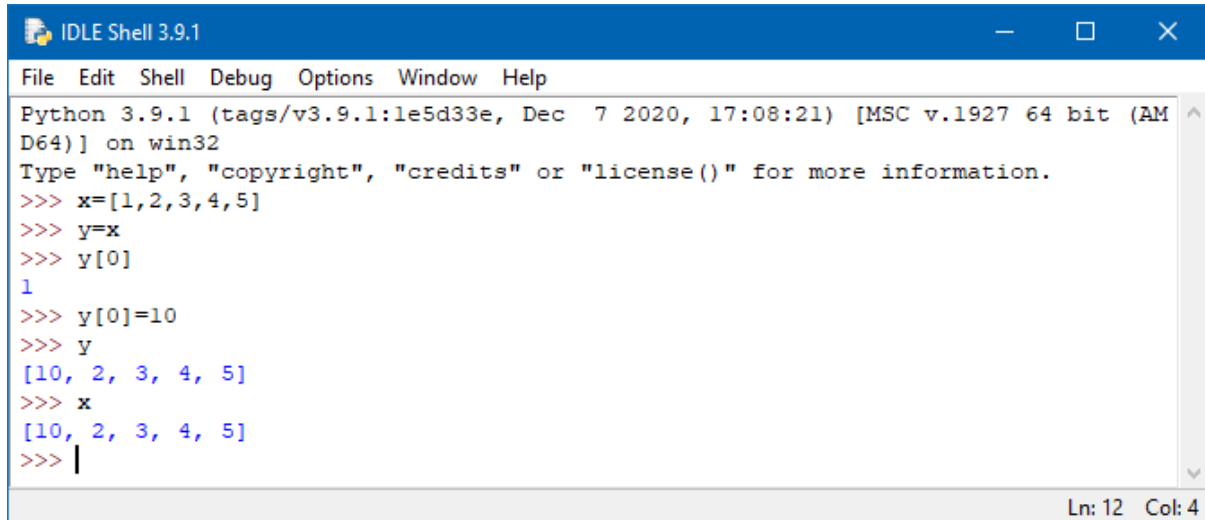
3.2.9. List Aliasing

Aliasing refers to assigning another variable to a list, both variable access same address location. Let us see an example.

Example:

```
>>> x=[1,2,3,4,5]
>>> y=x
>>> y[0]
1
>>> y[0]=10
```

```
>>> y
[10, 2, 3, 4, 5]
>>> x
[10, 2, 3, 4, 5]
>>>
```



```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> x=[1,2,3,4,5]
>>> y=x
>>> y[0]
1
>>> y[0]=10
>>> y
[10, 2, 3, 4, 5]
>>> x
[10, 2, 3, 4, 5]
>>> |
```

Ln: 12 Col: 4

Since the aliased variables access the same location, changes made to one variable affects the other.

3.2.10. Nested List

A list within a list is called nested list. Let us see an example.

Example:

```
>>> x=['audi','benz',[1,2,3]]
>>> x[0]
'audi'
>>> x[2]
[1, 2, 3]
>>> x[2][0]
1
>>> x[2][1]
2
>>> x[2][2]
3
>>>
```

In order to access the list within the list, a double bracket operation can be used (`[n][m]`), where `n` represents the index of the outer list and `m` represents the index of the list inside. It's clearly shown in the given example.


```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> x=["audi","benz",[1,2,3]]
>>> x[0]
'audi'
>>> x[2]
[1, 2, 3]
>>> x[2][0]
1
>>> x[2][1]
2
>>> x[2][2]
3
>>> |
```

Ln: 14 Col: 4

Nested lists are usually used to represent matrices

Example:

```
>>> a=[[1,2,3],[4,5,6],[7,8,9]]
>>> a
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>>
```

```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> a=[[1,2,3],[4,5,6],[7,8,9]]
>>> a
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> |
```

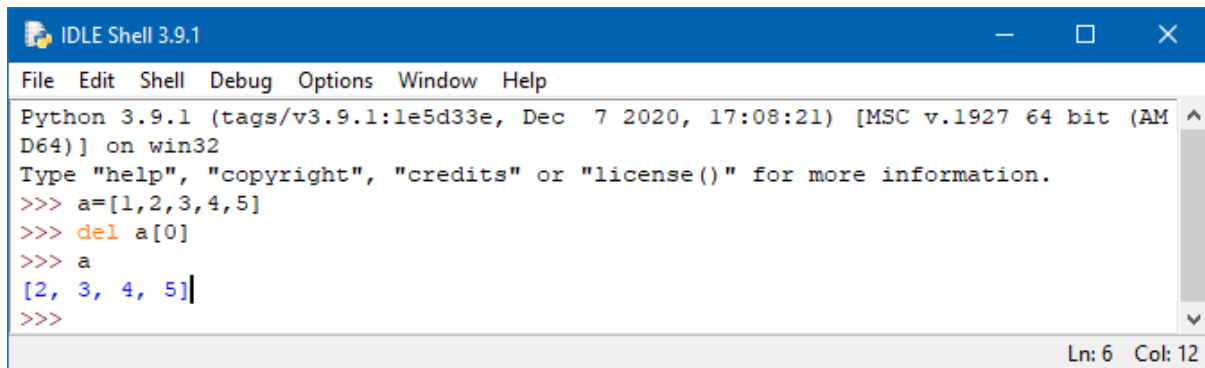
Ln: 6 Col: 4

3.2.11. Deleting a List

“del” can be used to remove an element from a list, “del variable [index]” is the syntax. Let us see an example.

Example:

```
>>> a=[1,2,3,4,5]
>>> del a[0]
>>> a
[2, 3, 4, 5]
>>>
```

A screenshot of the IDLE Shell 3.9.1 window. The title bar is blue with the text 'IDLE Shell 3.9.1' and standard window controls. The menu bar includes 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following Python code:

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> a=[1,2,3,4,5]
>>> del a[0]
>>> a
[2, 3, 4, 5]
```

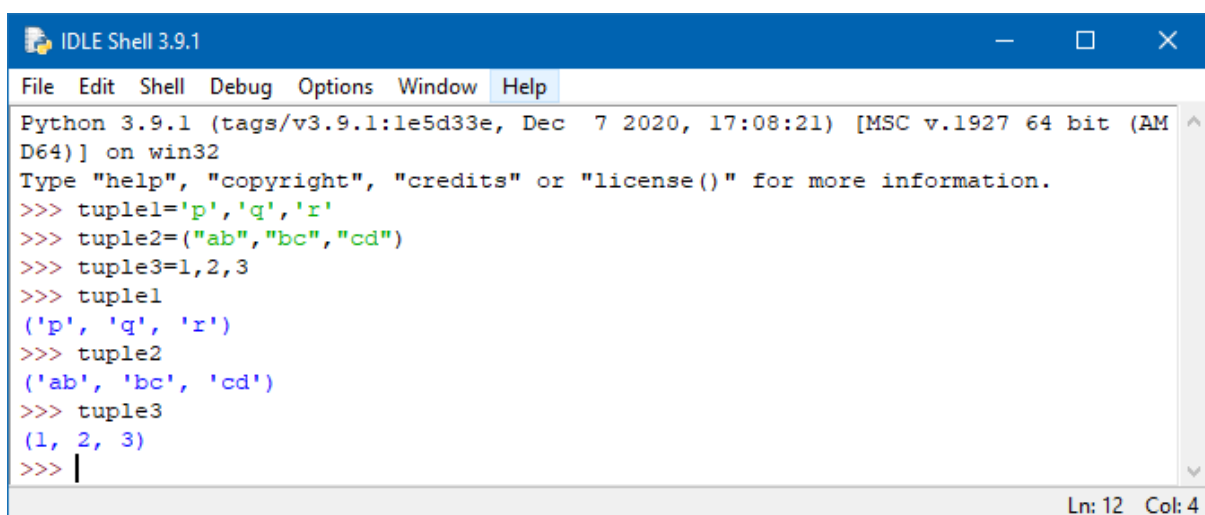
 The status bar at the bottom right indicates 'Ln: 6 Col: 12'.

3.3. TUPLES

Tuples are very similar to list except that it is immutable. You can say a list is a comma separated list of values. You may enclose tuples in parenthesis, which is a conventional approach. Let us see some examples

Example:

```
>>> tuple1='p','q','r'
>>> tuple2=('ab',"bc","cd")
>>> tuple3=1,2,3
>>> tuple1
('p', 'q', 'r')
>>> tuple2
('ab', 'bc', 'cd')
>>> tuple3
(1, 2, 3)
>>>
```

A screenshot of the IDLE Shell 3.9.1 window. The title bar is blue with the text 'IDLE Shell 3.9.1' and standard window controls. The menu bar includes 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following Python code:

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> tuple1='p','q','r'
>>> tuple2=('ab',"bc","cd")
>>> tuple3=1,2,3
>>> tuple1
('p', 'q', 'r')
>>> tuple2
('ab', 'bc', 'cd')
>>> tuple3
(1, 2, 3)
>>> |
```

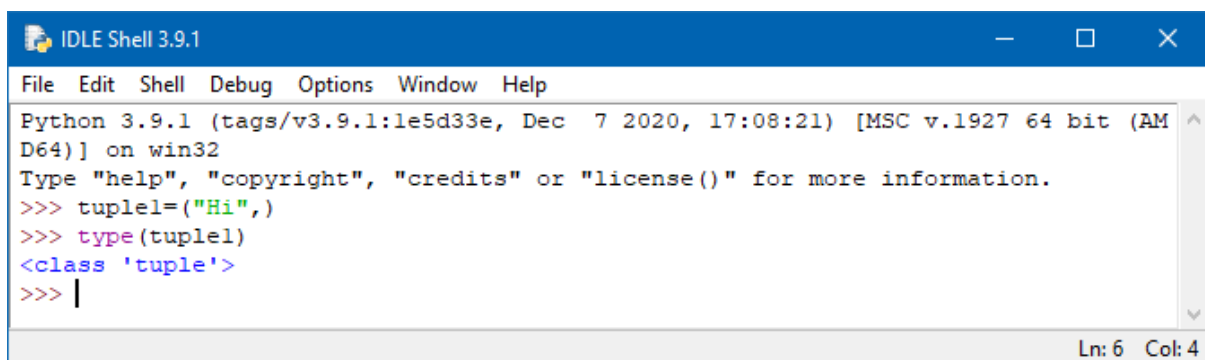
 The status bar at the bottom right indicates 'Ln: 12 Col: 4'.

If you want to check the type of the variable tuple 1, use type function.

```
>>> tuple1=1,2,3
>>> type(tuple1)
<class 'tuple'>
>>>
```

When you are creating a tuple with a single element, include a final comma as given below.

```
>>> tuple1=('Hi',)
>>> type(tuple1)
<class 'tuple'>
>>>
```

A screenshot of the IDLE Shell 3.9.1 window. The title bar says 'IDLE Shell 3.9.1'. The menu bar includes 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The shell area shows the following text: 'Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32', 'Type "help", "copyright", "credits" or "license()" for more information.', and the interactive session: '>>> tuple1=('Hi',)', '>>> type(tuple1)', '<class \'tuple\'>', and '>>> |'. The status bar at the bottom right shows 'Ln: 6 Col: 4'.

The reason why you have to add a comma is that, python consider ("Hi") without comma as a string.

3.3.1. Selecting Elements of Tuple

You can select the individual elements in a tuple using index value, just like we saw in strings and list. For example

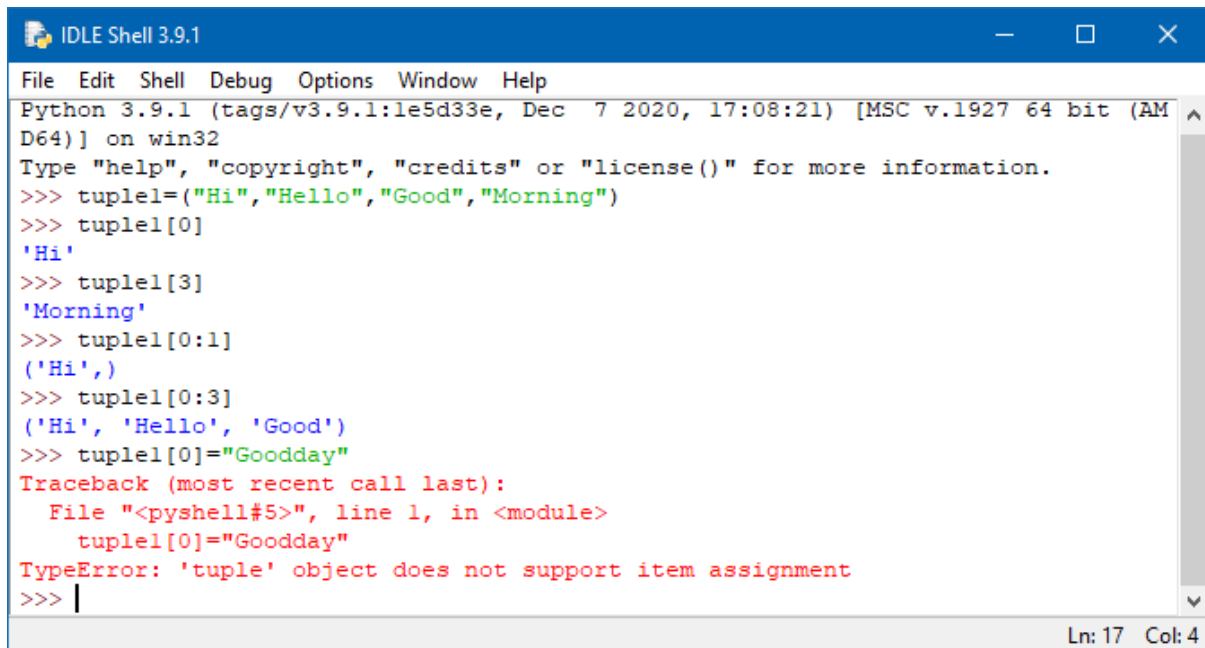
```
>>> tuple1=('Hi','Hello','Good','Morning')
>>> tuple1[0]
'Hi'
>>> tuple1[3]
'Morning'
>>>
```

3.3.2. Slice of a Tuple

The slice operator takes a slice of the tuple, same as what we have seen in case of list. See the below given example.

```
>>> tuple1[0:1]
('Hi',)
>>> tuple1[0:3]
('Hi', 'Hello', 'Good')
>>>
```

We get an error message if we try to modify an element of tuple after creation because tuples are immutable just like strings. See the below screenshot



```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> tuple1=("Hi","Hello","Good","Morning")
>>> tuple1[0]
'Hi'
>>> tuple1[3]
'Morning'
>>> tuple1[0:1]
('Hi',)
>>> tuple1[0:3]
('Hi', 'Hello', 'Good')
>>> tuple1[0]="Goodday"
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    tuple1[0]="Goodday"
TypeError: 'tuple' object does not support item assignment
>>> |
```

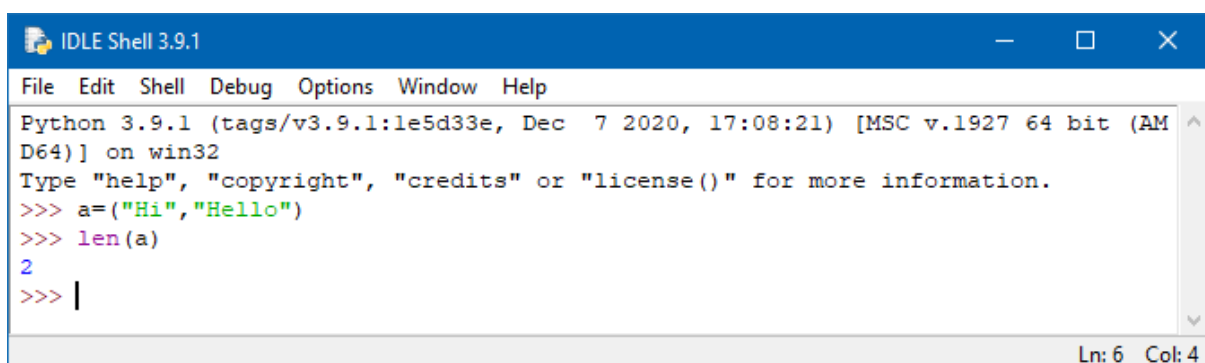
Ln: 17 Col: 4

3.3.3. Length of a Tuple

Length of a tuple can be found using the “len” function, and the operation is very similar to that of string and list.

Example:

```
>>> a=("Hi","Hello")
>>> len(a)
2
>>> |
```



```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> a=("Hi","Hello")
>>> len(a)
2
>>> |
```

Ln: 6 Col: 4

3.3.4. Tuple Assignment

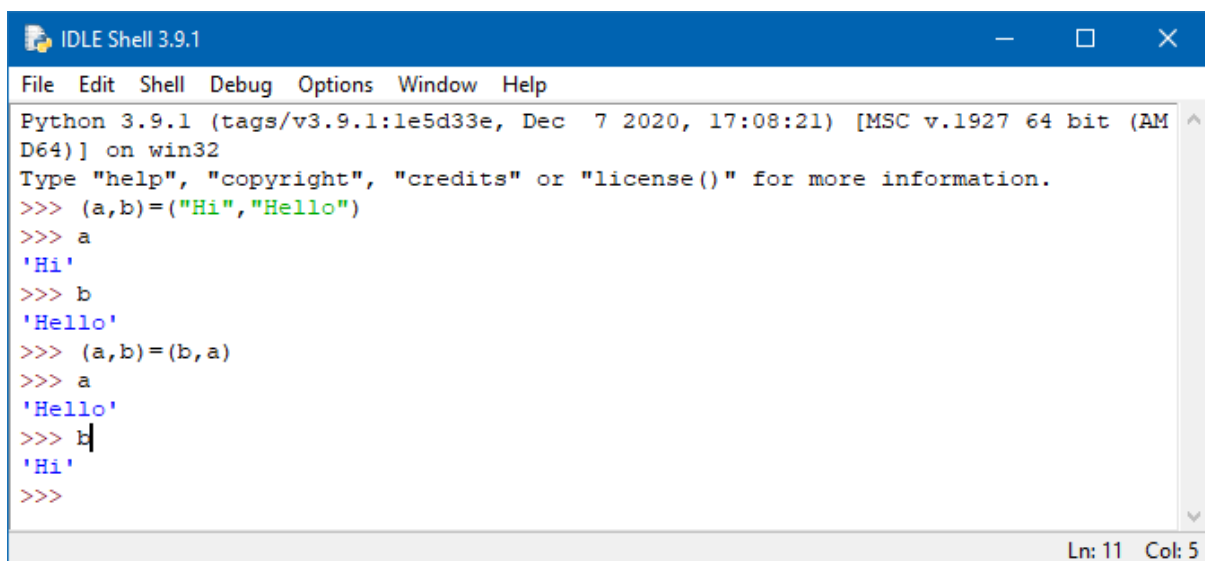
Python provides a very high level option to assign values to elements of tuple. For example

Example:

```
>>> (a,b)=("Hi","Hello")
>>> a
'Hi'
>>> b
'Hello'
>>>
```

This feature allows to swap values of two variables very easily. For example, if we have to swap values of a and b

```
>>> (a,b)=(b,a)
>>> a
'Hello'
>>> b
'Hi'
>>>
```

A screenshot of the IDLE Shell 3.9.1 window. The window has a blue title bar with the text 'IDLE Shell 3.9.1' and standard window controls. Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following Python code and its output:

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> (a,b)=("Hi","Hello")
>>> a
'Hi'
>>> b
'Hello'
>>> (a,b)=(b,a)
>>> a
'Hello'
>>> b
'Hi'
>>>
```

The status bar at the bottom right indicates 'Ln: 11 Col: 5'.

3.4. DICTIONARIES

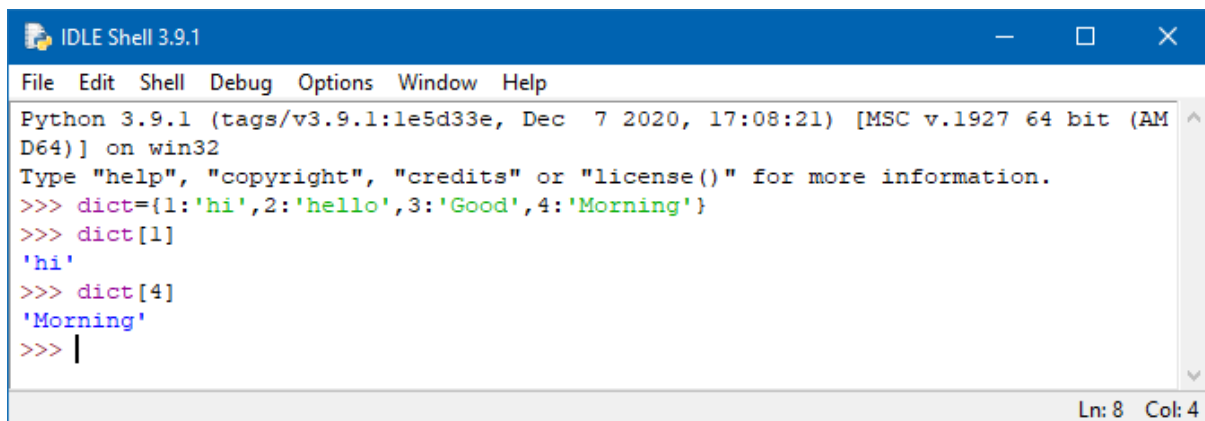
Dictionaries are similar to other compound types such as string, list and tuple except that they can use and immutable type as an index. You can create a dictionary by creating an empty dictionary using { }, and adding elements to it. You may consider the dictionary as a mapping between keys and values, here keys refers to indices. The association of a key and value is known as a key value pair. Let us see the syntax of a dictionary.

Syntax:

```
dict = {'key1': 'val1','key2': 'val2','key3': 'val3'...'keyn': 'valn'}
```

Example:

```
>>> dict={1:'hi',2:'hello',3:'Good',4:'Morning'}
>>> dict[1]
'hi'
>>> dict[4]
'Morning'
>>>
```

A screenshot of the IDLE Shell 3.9.1 window. The window has a blue title bar with the text 'IDLE Shell 3.9.1' and standard window controls. Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following code and output:

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> dict={1:'hi',2:'hello',3:'Good',4:'Morning'}
>>> dict[1]
'hi'
>>> dict[4]
'Morning'
>>> |
```

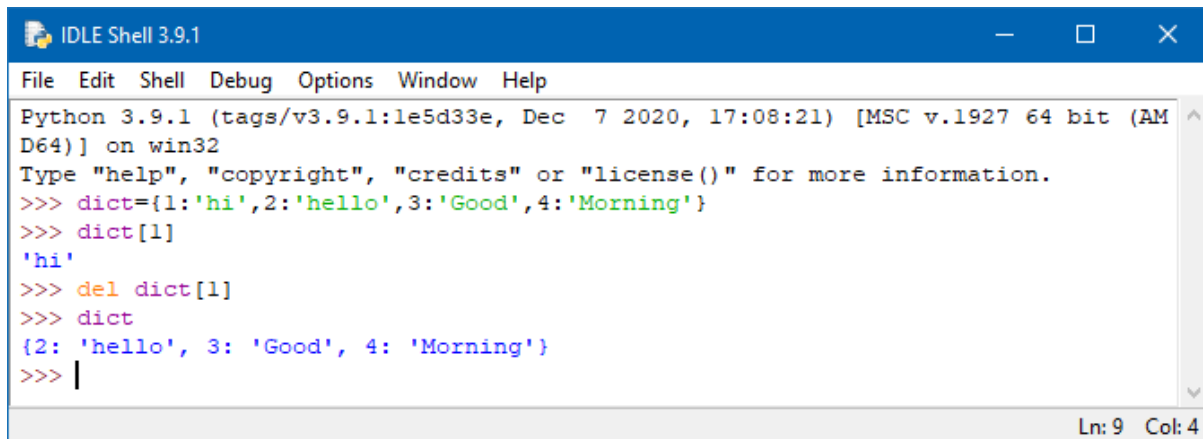
The status bar at the bottom right indicates 'Ln: 8 Col: 4'.

Since the keys here are number, we haven't used quotes.

3.4.1. Dictionary Operations

You can use "del" to delete the elements in a dictionary, you can refer to the key of element to be deleted. For example.

```
>>> dict={1:'hi',2:'hello',3:'Good',4:'Morning'}
>>> del dict[1]
```

A screenshot of the IDLE Shell 3.9.1 window. The title bar says "IDLE Shell 3.9.1". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The shell area shows the following code:

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> dict={1:'hi',2:'hello',3:'Good',4:'Morning'}
>>> dict[1]
'hi'
>>> del dict[1]
>>> dict
{2: 'hello', 3: 'Good', 4: 'Morning'}
>>> |
```

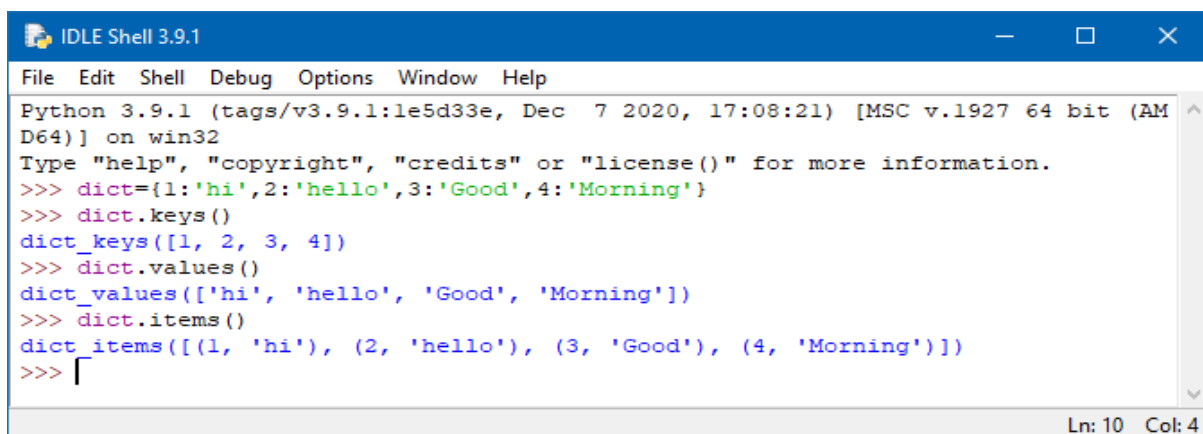
 The status bar at the bottom right shows "Ln: 9 Col: 4".

The “len” function works similar to list and tuples, it returns the number of key-value pairs in the dictionary.

```
>>> dict={1:'hi',2:'hello',3:'Good',4:'Morning'}
>>> len(dict)
4
```

One can use the keys method to return all the keys in a given dictionary, whereas the value method returns the values in the dictionary. The items method returns both keys and values. For example

```
>>> dict={1:'hi',2:'hello',3:'Good',4:'Morning'}
>>> dict.keys()
dict_keys([1, 2, 3, 4])
>>> dict.values()
dict_values(['hi', 'hello', 'Good', 'Morning'])
>>> dict.items()
dict_items([(1, 'hi'), (2, 'hello'), (3, 'Good'), (4, 'Morning')])
```

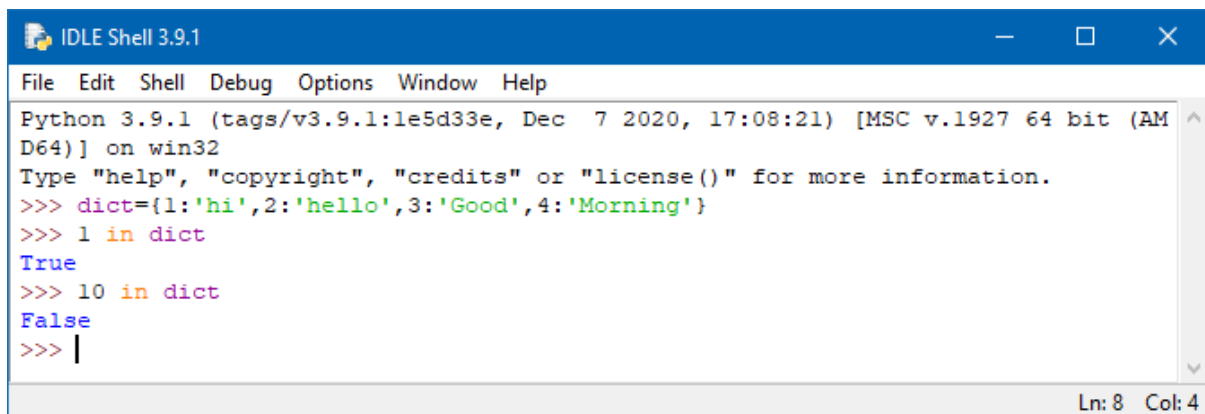
A screenshot of the IDLE Shell 3.9.1 window. The title bar says "IDLE Shell 3.9.1". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The shell area shows the following code:

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> dict={1:'hi',2:'hello',3:'Good',4:'Morning'}
>>> dict.keys()
dict_keys([1, 2, 3, 4])
>>> dict.values()
dict_values(['hi', 'hello', 'Good', 'Morning'])
>>> dict.items()
dict_items([(1, 'hi'), (2, 'hello'), (3, 'Good'), (4, 'Morning')])
>>> |
```

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

The `in` keyword returns a true value if the passed key value is present in the dictionary.
For example

```
>>> dict={1:'hi',2:'hello',3:'Good',4:'Morning'}
>>> 1 in dict
True
>>> 10 in dict
False
>>>
```

A screenshot of the IDLE Shell 3.9.1 window. The window has a blue title bar with the text 'IDLE Shell 3.9.1' and standard window controls. Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following code and output:

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> dict={1:'hi',2:'hello',3:'Good',4:'Morning'}
>>> 1 in dict
True
>>> 10 in dict
False
>>> |
```

The status bar at the bottom right indicates 'Ln: 8 Col: 4'.

CHAPTER 4

FUNCTIONS

4.1. FUNCTIONS

A function is a statement or a group of statement that performs a particular task. A python function is named and it has a body which contains a sequence of instructions that are executed one by one up on function call.

There are different advantages while using functions in a program, such as

- Function can be called whenever they are to be executed, without duplicating required line of code. Hence functions are reusable.
- Functions helps in reducing the complexity of program, by decomposing a large program to small segments. Which make program readable and easy to understand.
- Programmers can share work load among team by modulating the requirement.

Functions are categorized three

- Functions in Modules
- Built in Functions
- User Defined Functions

4.1.1. Functions in Module

Python module is a file that contains definitions such as functions and statements. You can consider modules as standard library files. The functions in a module can be used in the program, by importing the required module. Now let us see how we can import a module and use definitions in it.

1. Import

Using import is the simplest and most widely used way to import modules to a python program. Let us see the syntax of import.

Syntax:

import modulename₁, modulename₂,, modulename_n

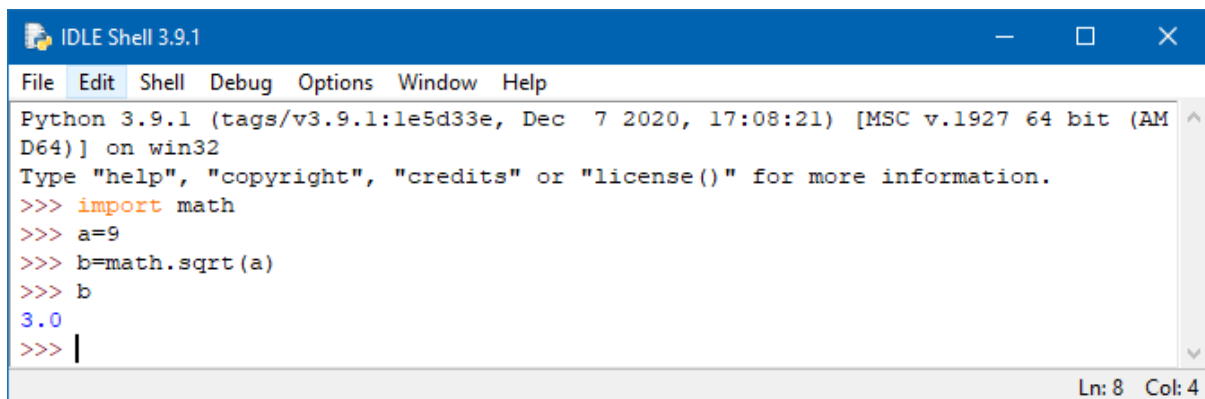
Example:

```
>>> import math, string, io
```

The above example import math, string and io modules. In order to use a function in a module, you have to use dot notation, that is the “**module name.function_name**”

Example:

```
>>> import math
>>> a=9
>>> b=math.sqrt(a)
>>> b
3.0
```

A screenshot of the IDLE Shell 3.9.1 window. The title bar says 'IDLE Shell 3.9.1'. The menu bar includes 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The shell area shows the following text: 'Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32', 'Type "help", "copyright", "credits" or "license()" for more information.', and the code execution: '>>> import math', '>>> a=9', '>>> b=math.sqrt(a)', '>>> b', '3.0', and '>>> |'. The status bar at the bottom right shows 'Ln: 8 Col: 4'.

Here in this program the sqrt() function with in the math module, calculate the square root of the given value.

2. From

From is another way of using definitions in a module, the import feature will import the modules fully. If we know the specific functions within a module to be included, then we can use from. It is recommended to use from to import the functions if the corresponding module is fairly big.

Syntax:

from modulename import function 1 , function 2 , ... , function n

Example:

```
>>> from math import sqrt, log, pow
```

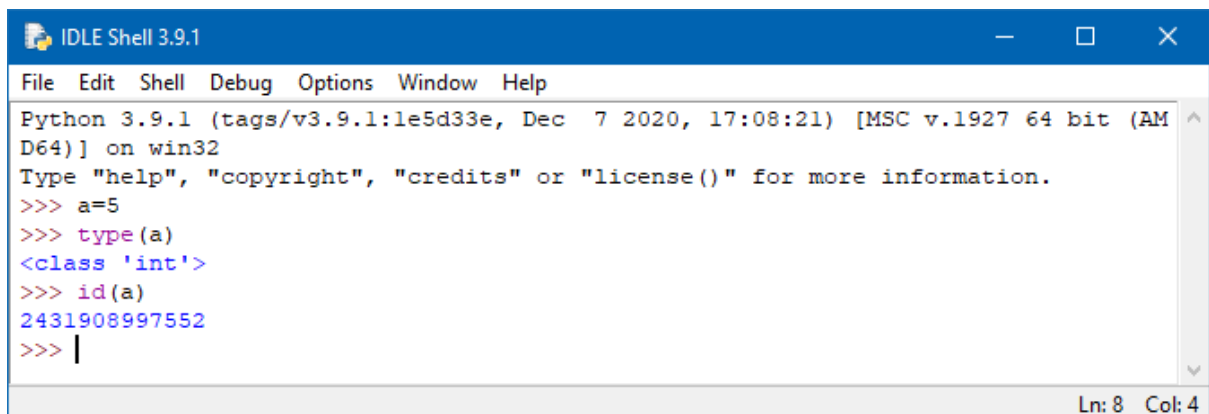
4.1.2. Built in Functions

These are the function that are available by default in python, and can be used directly without adding any module files. Built in functions are very small in number compared to that available via modules.

Let us see some examples of built in functions below.

Example:

```
>>> a=5
>>> type(a)
<class 'int'>
>>> id(a)
2431908997552
>>>
```

A screenshot of the IDLE Shell 3.9.1 window. The window has a blue title bar with the text 'IDLE Shell 3.9.1' and standard window controls. Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following text: 'Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32', 'Type "help", "copyright", "credits" or "license()" for more information.', and the interactive session code: '>>> a=5', '>>> type(a)', '<class \'int\'>', '>>> id(a)', '2431908997552', and '>>> |'. The status bar at the bottom right shows 'Ln: 8 Col: 4'.

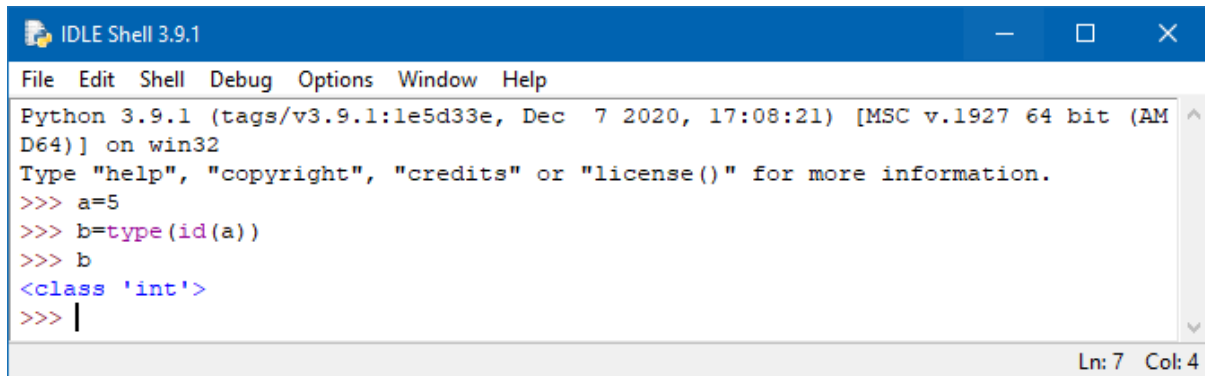
Here id() and type() are the built functions and used without adding any modules.

4.1.3. Composition

Python allows you to combine simple functions, that is the output of one function will be the input to the other function. Let us see an example.

Example:

```
>>> a=5
>>> b=type(id(a))
>>> b
<class 'int'>
>>>
```

A screenshot of the IDLE Shell 3.9.1 window. The window has a blue title bar with the text 'IDLE Shell 3.9.1' and standard window controls. Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following code: 'Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32', 'Type "help", "copyright", "credits" or "license()" for more information.', '>>> a=5', '>>> b=type(id(a))', '>>> b', '<class \'int\'>', and '>>> |'. The status bar at the bottom right shows 'Ln: 7 Col: 4'.

Here the result of function `id(a)` used as an argument for the `type()` function. This way you can combine simple python functions.

4.1.4. User Defined Functions

In python programmers have the freedom to write their own functions and use them, such user written functions are called as user defined functions. These functions can be combined to modules and they can be used later by importing the module. “def” keyword is used to define a function, and a function is named by an identifier following “def”. The identifier is then followed by parenthesis with in which the parameters are specified. The function then ends with a colon, and next follows a block of statement(s) that are the body of the function. Let us see the syntax of a function.

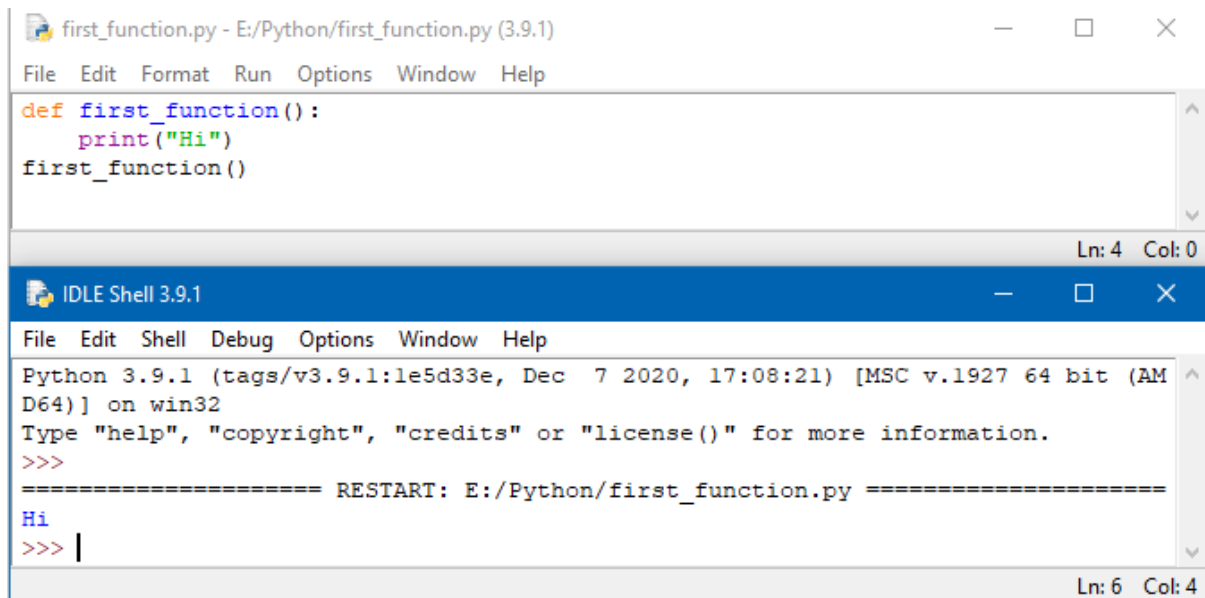
Syntax:

```
def function_name (parameter 1, parameter 2, ...):
    statement(s)
```

Now let us write a very simple function to print “Hi”, let us name the function as first_function

Example:

```
def first_function():  
    print("Hi")  
first_function()
```



The screenshot displays the Python IDLE 3.9.1 environment. The top window, titled 'first_function.py - E:/Python/first_function.py (3.9.1)', contains the following code:

```
def first_function():  
    print("Hi")  
first_function()
```

The bottom window, titled 'IDLE Shell 3.9.1', shows the execution output. It includes the Python version and system information, followed by a restart message for the script. The output of the function call is 'Hi'.

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: E:/Python/first_function.py =====  
Hi  
>>> |
```

Function Header

Function header begins with def and ends in colon. Function header has function identification details such as function name and parameters. Here in our example the first line is the function header.

Function Body

Function header is followed by function body, which contains set of statements to be executed up on the function call. Here in our example line number 2 is function body.

Function Call

Function call is performed to execute the contents of a body of a function. Function call contains the name of the function followed by the list of arguments in parenthesis.

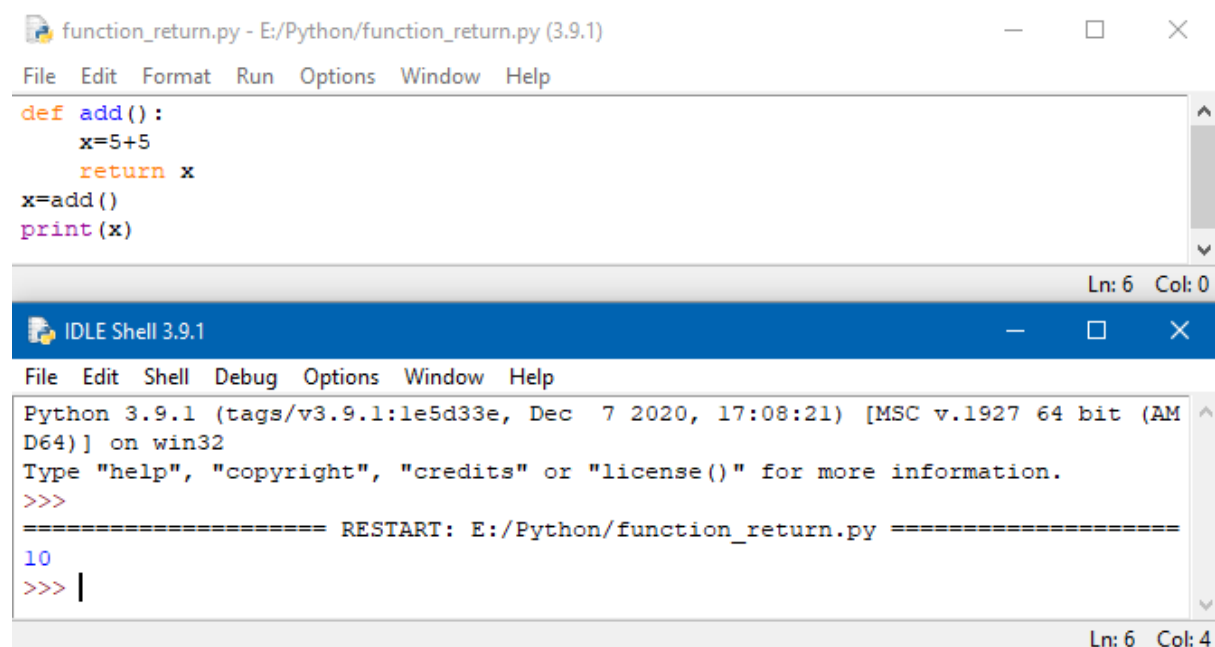
These arguments are assigned to parameters from Left to right. Here in our example line number 3 is the function call. Since our example function does not accept any arguments function call is very simple as function name followed by parenthesis.

Return Statement

Return statement in a function is used to return values from functions. Return statements return values to its caller. Return statement may contain a variable, a constant, an expression or function, if return is used without anything, it will return None. A function having return values are called as fruitful function and those function that does not return anything is called as void functions. Let us consider a simple example.

Example:

```
def add():  
    x=5+5  
    return x  
x=add()  
print(x)
```



The screenshot displays the Python IDLE 3.9.1 environment. The top window, titled 'function_return.py - E:/Python/function_return.py (3.9.1)', contains the following Python code:

```
def add():  
    x=5+5  
    return x  
x=add()  
print(x)
```

The bottom window, titled 'IDLE Shell 3.9.1', shows the execution of the code. It displays the Python version and system information, followed by a restart message for the script. The output of the program is '10'.

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: E:/Python/function_return.py =====  
10  
>>> |
```

Here in this program, the function add computes 5+5 and save to x, this x is returned using "return x" to the caller. Later this x is printed from outside the function.

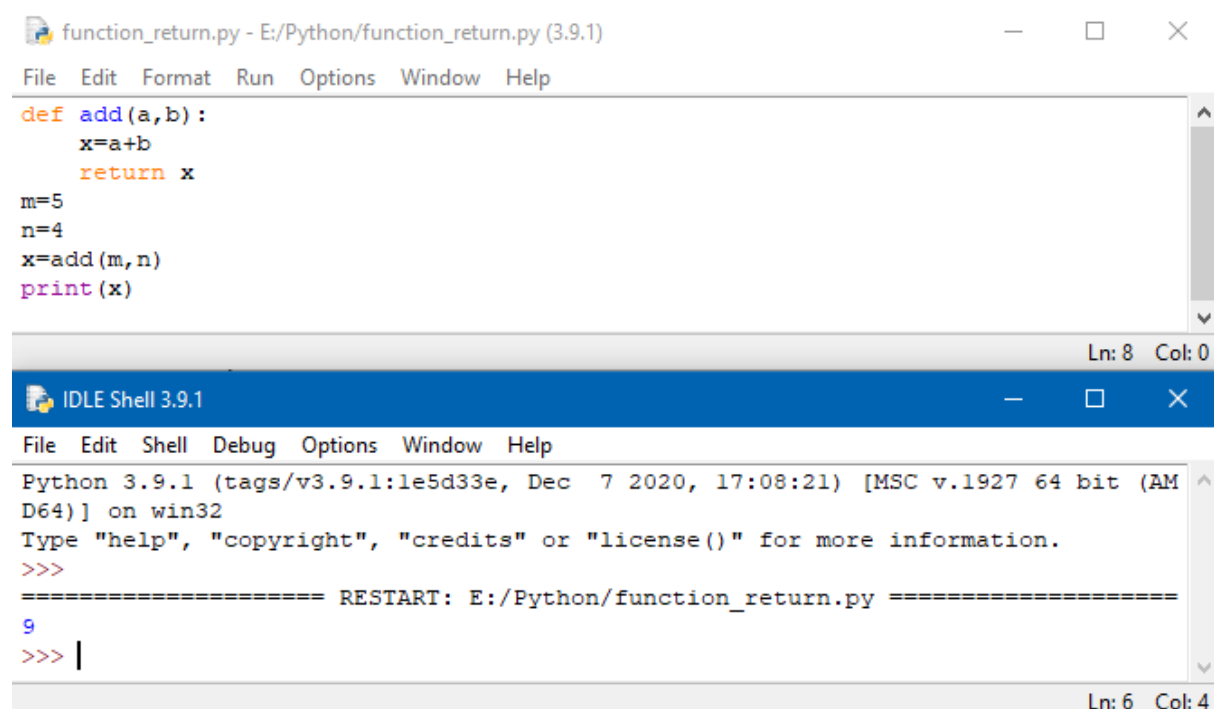
Now let us see the parameters and arguments in detail and discuss examples of functions that has parameters.

Parameters and Arguments

Parameters are otherwise called as **formal parameters**, they are the value(s) provided in the parenthesis of function header. These values are required to operate the function body. There can be a single parameter or multiple parameters in a function. **Arguments or actual parameters** are the value(s) provided along with the function call. The order actual parameters must match the order of formal parameters. Mapping of parameters to arguments is done one a one is to one manner. Hence the number order and type of actual parameter should match that of formal parameters. Now let us see a simple example.

Example:

```
def add(a,b):  
    x=a+b  
    return x  
m=5  
n=4  
x=add(m,n)  
print(x)
```



The screenshot displays the Python IDLE 3.9.1 environment. The top window, titled 'function_return.py - E:/Python/function_return.py (3.9.1)', contains the following Python code:

```
def add(a,b):  
    x=a+b  
    return x  
m=5  
n=4  
x=add(m,n)  
print(x)
```

The bottom window, titled 'IDLE Shell 3.9.1', shows the execution of the code. It displays the Python version and architecture information, followed by a restart message and the output of the program:

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: E:/Python/function_return.py =====  
9  
>>> |
```

The output of the program is the number 9, which is the result of adding 5 and 4.

Here in this program m, n are the actual parameters (arguments) and a, b are the formal parameters (Parameters). Up on function call the values of m and n are assigned to a and b respectively, then the body of the function is executed and the result is returned as x. Later x is printed.

You can also pass arguments directly. That is add(5,4)

```
def add(a,b):  
    x=a+b  
    return x  
x=add(5,4)  
print(x)
```

Here in the above program, the arguments are constant and directly passed to the function. This will also produce an output of 9.

4.1.5. Scope of Variables

Scope of variable refers to the portion of program where the variable is visible or usable. We can say that scope holds current variables and its values. There are two different type of scopes, global scope and local scope. Let us see each of them with simple examples.

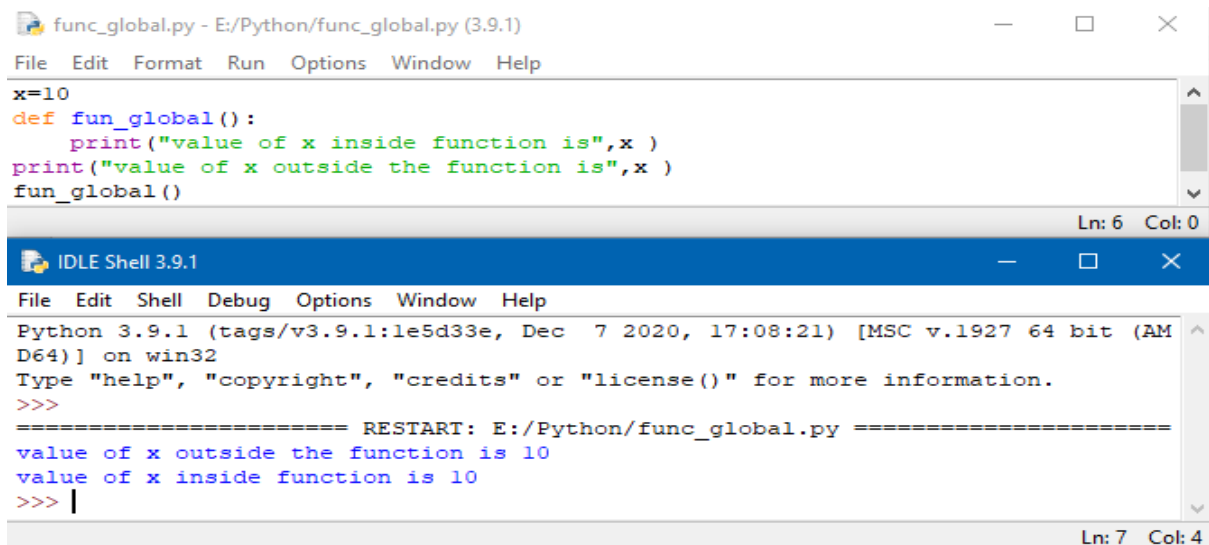
Global Scope

A variable having global scope can be used anywhere in the program. Such variables are called as global variables and are defined outside the scope of any function or module. Let us see an example.

Example

```
x=10  
def fun_global():  
    print("value of x inside function is",x )  
print("value of x outside the function is",x )  
fun_global()
```

The program will output as shown in screen shot, the values of x inside and outside x will be 10.



```
func_global.py - E:/Python/func_global.py (3.9.1)
File Edit Format Run Options Window Help
x=10
def fun_global():
    print("value of x inside function is",x )
print("value of x outside the function is",x )
fun_global()
Ln: 6 Col: 0

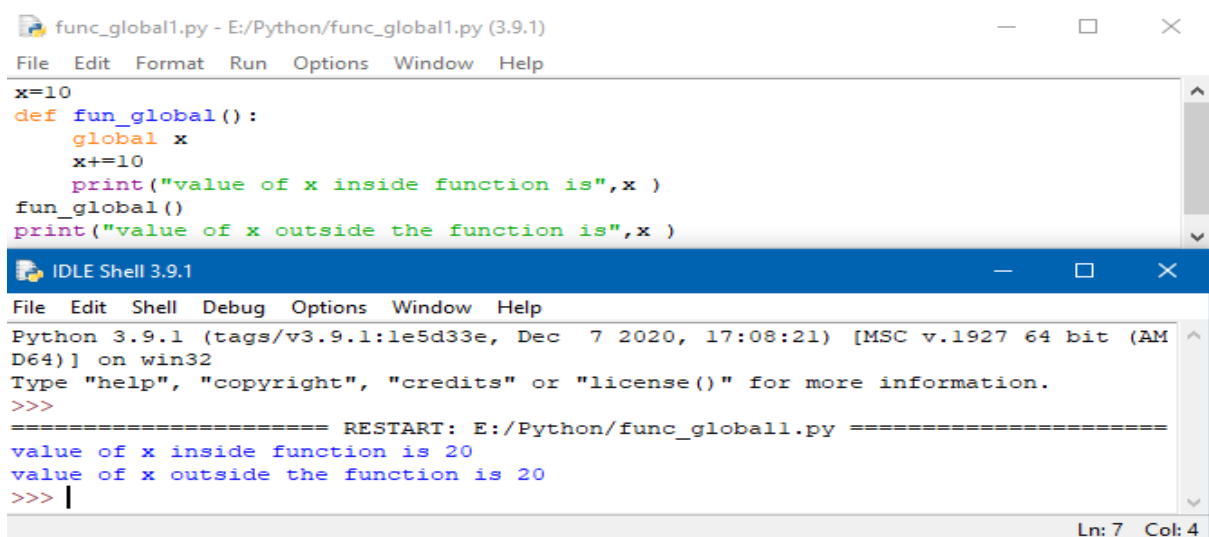
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: E:/Python/func_global.py =====
value of x outside the function is 10
value of x inside function is 10
>>> |
Ln: 7 Col: 4
```

Any modification done to the global variable is permanent and visible to all the functions written in the program. Note: If you want to refer global variable inside a function, you have to declare the variable using keyword global inside the function. Let us see an example

Example:

```
x=10
def fun_global():
    global x
    x+=10
    print("value of x inside function is",x )
fun_global()
print("value of x outside the function is",x )
```

This program print values of x insider and outside as 20. See the screenshot.



```
func_global1.py - E:/Python/func_global1.py (3.9.1)
File Edit Format Run Options Window Help
x=10
def fun_global():
    global x
    x+=10
    print("value of x inside function is",x )
fun_global()
print("value of x outside the function is",x )

IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: E:/Python/func_global1.py =====
value of x inside function is 20
value of x outside the function is 20
>>> |
Ln: 7 Col: 4
```

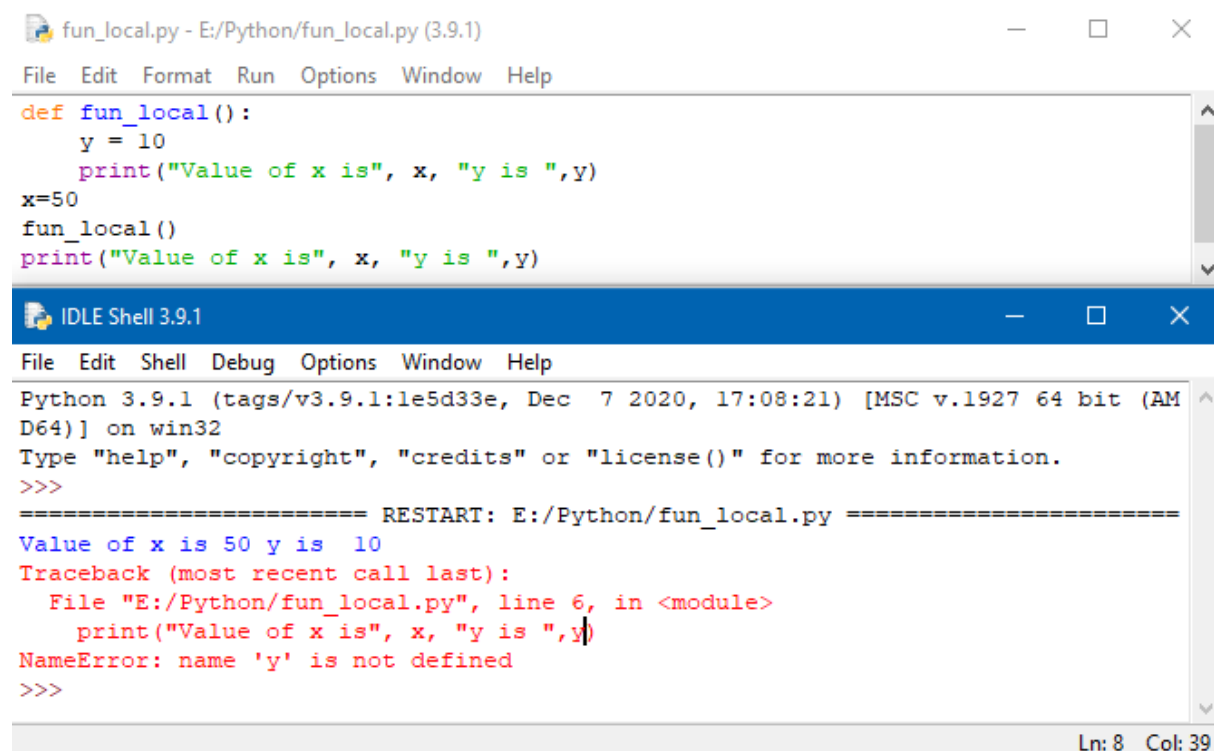
Local Scope

A variable having local scope can only be accessed within the function that it is created in. When a variable is created inside the function, the variable becomes local to it. A local variable only exists while the function is executing. Let us see an example.

Example:

```
def fun_local():  
    y = 10  
    print("Value of x is", x, "y is ",y)  
x=50  
fun_local()  
print("Value of x is", x, "y is ",y)
```

The program will output as , Value of x is 50 y is 10 followed by an error, because y is a local variable whose scope is limited in the function fun_local(). See the screenshot.



The screenshot shows the Python IDLE Shell 3.9.1 interface. The top window displays the code from the example. The bottom window shows the execution output: "Value of x is 50 y is 10", followed by a "NameError: name 'y' is not defined" message. The error message includes a traceback pointing to line 6 of the file "E:/Python/fun_local.py".

```
def fun_local():  
    y = 10  
    print("Value of x is", x, "y is ",y)  
x=50  
fun_local()  
print("Value of x is", x, "y is ",y)
```

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: E:/Python/fun_local.py =====  
Value of x is 50 y is 10  
Traceback (most recent call last):  
  File "E:/Python/fun_local.py", line 6, in <module>  
    print("Value of x is", x, "y is ",y)  
NameError: name 'y' is not defined  
>>>
```

Ln: 8 Col: 39

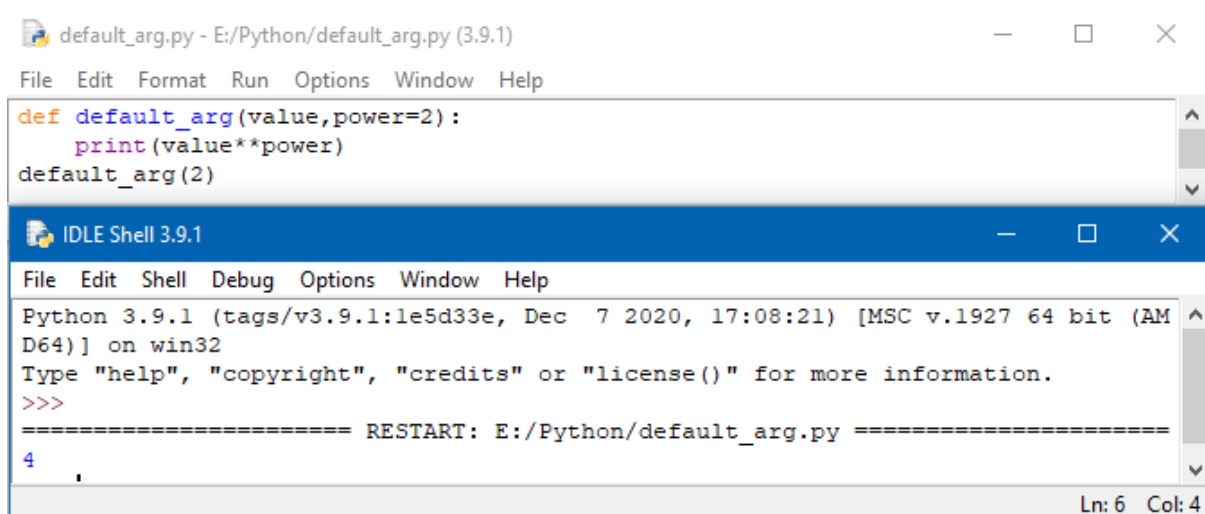
4.1.6. Default arguments

It is possible to initialize the parameters in the definition of a function itself, such arguments are called as default arguments. This is useful when the programmers does not want to pass the values during the function call. Let us see an example.

Example

```
def default_arg(value,power=2):  
    print(value**power)  
default_arg(2)
```

The user specifies only the first argument in the function call, the second argument is taken from function definition by default. The output of the program is 4 and shown in screenshot.



The screenshot shows the Python IDLE 3.9.1 interface. The top window, titled 'default_arg.py - E:/Python/default_arg.py (3.9.1)', contains the following code:

```
def default_arg(value,power=2):  
    print(value**power)  
default_arg(2)
```

The bottom window, titled 'IDLE Shell 3.9.1', shows the output of the program. It displays the Python version and environment information, followed by a prompt 'Type "help", "copyright", "credits" or "license()" for more information.' and a 'RESTART' message. The output of the function call is '4'.

If the user decides to provide both the argument, it will override the default arguments. That is if the user calls `default_arg(2,10)` the output will be 1024.

4.1.7. Recursions

A function calling itself is called recursion. In a recursive function the function call is made within the body of the function. A recursive function has to terminate, in order to terminate a recursive function there should be a terminating criteria. This terminating criteria is otherwise called as base condition. A recursive function terminates, if with every recursive call the solution of the problem is downsized and moves towards a

base case. A base case is a case, where the problem can be solved without further recursion. If the base case is not met, the recursive function leads to an infinite loop.

Let us see a simple example of recursion, Factorial can be solved using a recursive function.

Example:

$4! = 4 * 3!$

$3! = 3 * 2!$

$2! = 2 * 1$

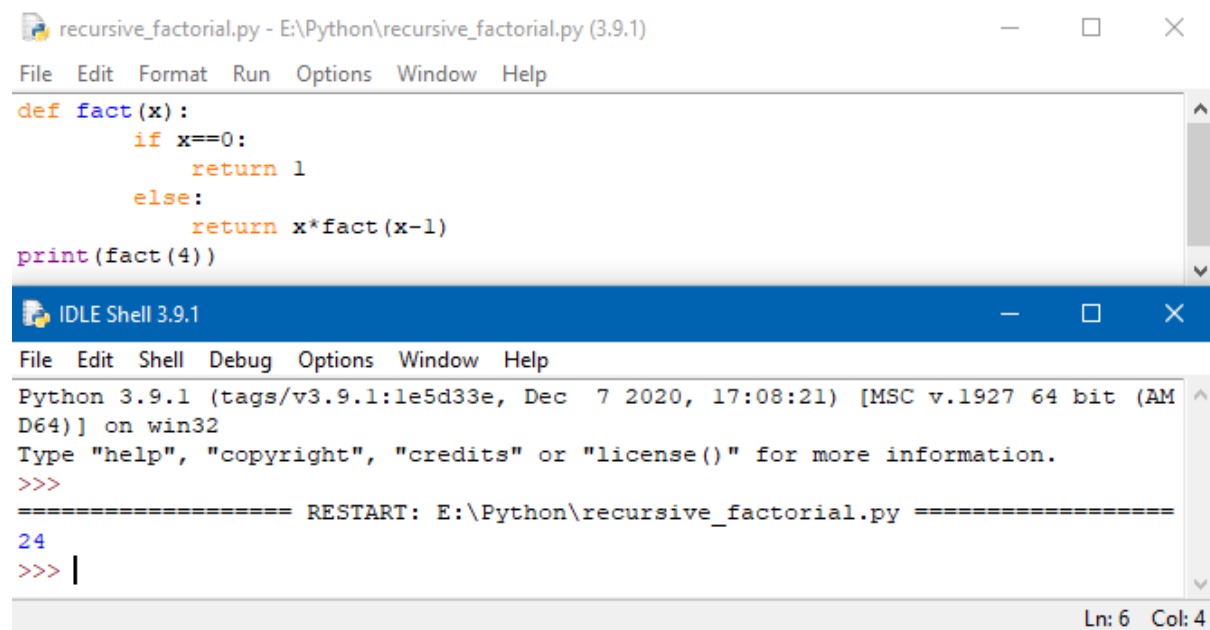
That is $4! = 4 * 3 * 2 * 1$

Now let us see the python recursive code for factorial:

Program:

```
def fact(x):  
    if x==0:  
        return 1  
    else:  
        return x*fact(x-1)  
print(fact(4))
```

The output is given in the below screenshot.



The screenshot displays the Python IDLE Shell 3.9.1 interface. The top window shows the code for a recursive factorial function. The bottom window shows the execution output, which includes a restart message and the result of the function call.

```
recursive_factorial.py - E:\Python\recursive_factorial.py (3.9.1)  
File Edit Format Run Options Window Help  
def fact(x):  
    if x==0:  
        return 1  
    else:  
        return x*fact(x-1)  
print(fact(4))  
IDLE Shell 3.9.1  
File Edit Shell Debug Options Window Help  
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: E:\Python\recursive_factorial.py =====  
24  
>>> |
```

Ln: 6 Col: 4

Now let us see how to display Fibonacci series recursively.

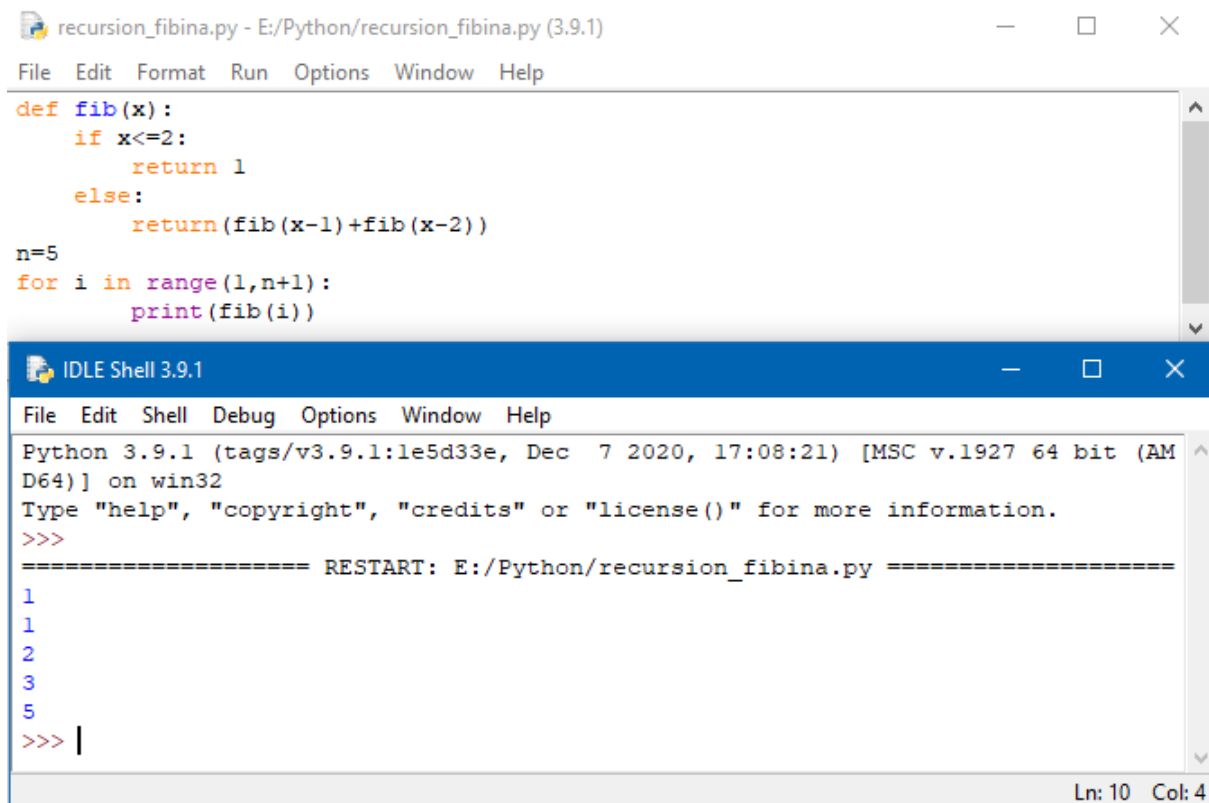
The Fibonacci numbers are : 0,1,1,2,3,5,8,13,21,34,55,89,... First two values of Fibonacci series are 0 and 1, then next values are found by adding previous two values. That is

$$f_n = f(n-1) + f(n-2) \text{ where } f_0 = 1 \text{ and } f_1 = 0$$

Program:

```
def fib(x):
    if x<=2:
        return 1
    else:
        return(fib(x-1)+fib(x-2))
n=5
for i in range(1,n+1):
    print(fib(i))
```

This program recursively calls fib() function and produces Fibonacci series for number 5. See the below screen shot to see the output.



The screenshot displays a Python IDE window titled 'recursion_fibina.py - E:/Python/recursion_fibina.py (3.9.1)'. The code editor shows the following Python code:

```
def fib(x):
    if x<=2:
        return 1
    else:
        return(fib(x-1)+fib(x-2))
n=5
for i in range(1,n+1):
    print(fib(i))
```

Below the code editor is the 'IDLE Shell 3.9.1' window. It shows the output of the program as the Fibonacci series for n=5: 1, 1, 2, 3, 5. The shell also displays the Python version (3.9.1) and the file path (E:/Python/recursion_fibina.py).

4.1.8. Type Conversion

Type conversion (type casting) is a way of changing an entity of one data type into another. Let us see some of type conversions via examples.

Function	Conversion	Example
int()	string, floating point → integer	>>> int('2014') 2014 >>> int(2.1418) 2
float()	string, integer → floating point number	>>> float('2.98') 2.98 >>> float(5) 5.0
str()	integer, float, list, tuple, dictionary → string	>>> str(2.1418) '2.1418' >>> str([1,2,3,4,5]) '[1, 2, 3, 4, 5]'

Type coercion:

Coercion is the implicit conversion of an instance of one type to another during an operation which involves two arguments of the same type.

Eg: **x= 7/2.0**

Here 7 is integer and 2.0 is a float number. But during division 7 is implicitly converted to 7.0(float) and result x= 3.5

```
>>> x=7/2.0
```

```
>>> x
```

```
3.5
```

```
>>>
```

CHAPTER 5

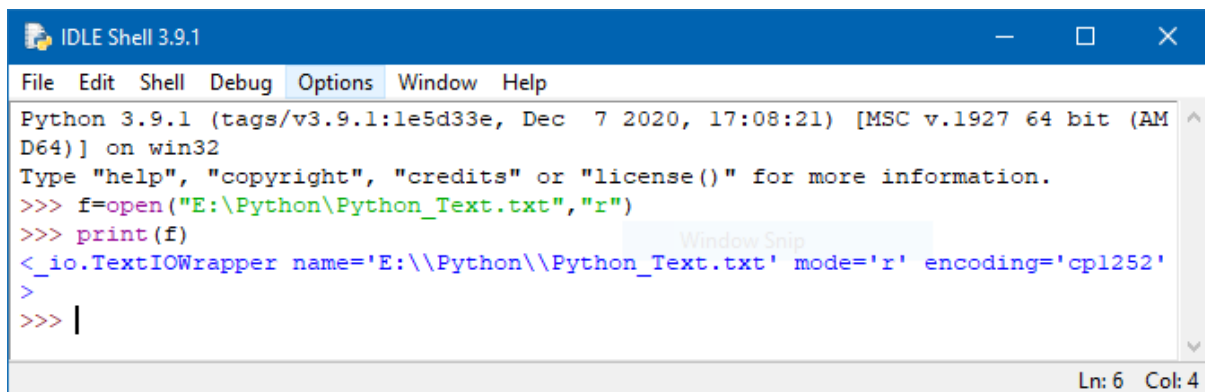
FILES AND EXCEPTIONS

Python support a wide variety of operations on file. In this chapter we will discuss various file operations in python with simplified examples.

5.1. FILE OPENING AND WRITING

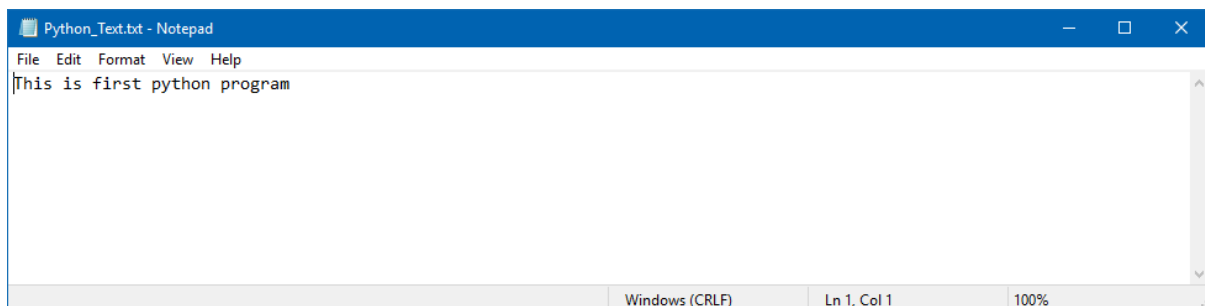
Opening a file in python creates a file object, let us see how you can open a file in python. You can open a file using open function with file name and file open mode as arguments.

```
>>> f=open("E:\Python\Python_Text.txt","r")
>>> print(f)
<_io.TextIOWrapper name='E:\\Python\\Python_Text.txt' mode='r'
encoding='cp1252'>
```

A screenshot of the IDLE Shell 3.9.1 window. The title bar says "IDLE Shell 3.9.1". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The shell area shows the following text: "Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32", "Type 'help', 'copyright', 'credits' or 'license()' for more information.", and the code execution output: ">>> f=open('E:\Python\Python_Text.txt','r')", ">>> print(f)", and the result: "<_io.TextIOWrapper name='E:\\Python\\Python_Text.txt' mode='r' encoding='cp1252'". The status bar at the bottom right shows "Ln: 6 Col: 4".

```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> f=open("E:\Python\Python_Text.txt","r")
>>> print(f)
<_io.TextIOWrapper name='E:\\Python\\Python_Text.txt' mode='r' encoding='cp1252'
>
>>> |
```

The content of Python_Text.txt is given below

A screenshot of a Notepad window titled "Python_Text.txt - Notepad". The menu bar includes File, Edit, Format, View, and Help. The text area contains the text "This is first python program". The status bar at the bottom shows "Windows (CRLF)", "Ln 1, Col 1", and "100%".

```
Python_Text.txt - Notepad
File Edit Format View Help
This is first python program
```

The open function return a file pointer , here in our example f is the file pointer. The first argument is the function name and here we have given filename with full path, the second argument is mode, this depends on the kind of operation one need to do on the file. “r” for reading file and “w” for writing file. In write mode if there is no file named Python_Text.txt, it will be created. If there already is one, it will be replaced

by the file we are writing. If we try to open one file which is not existing in read mode, python will return an error saying the file is not existing.

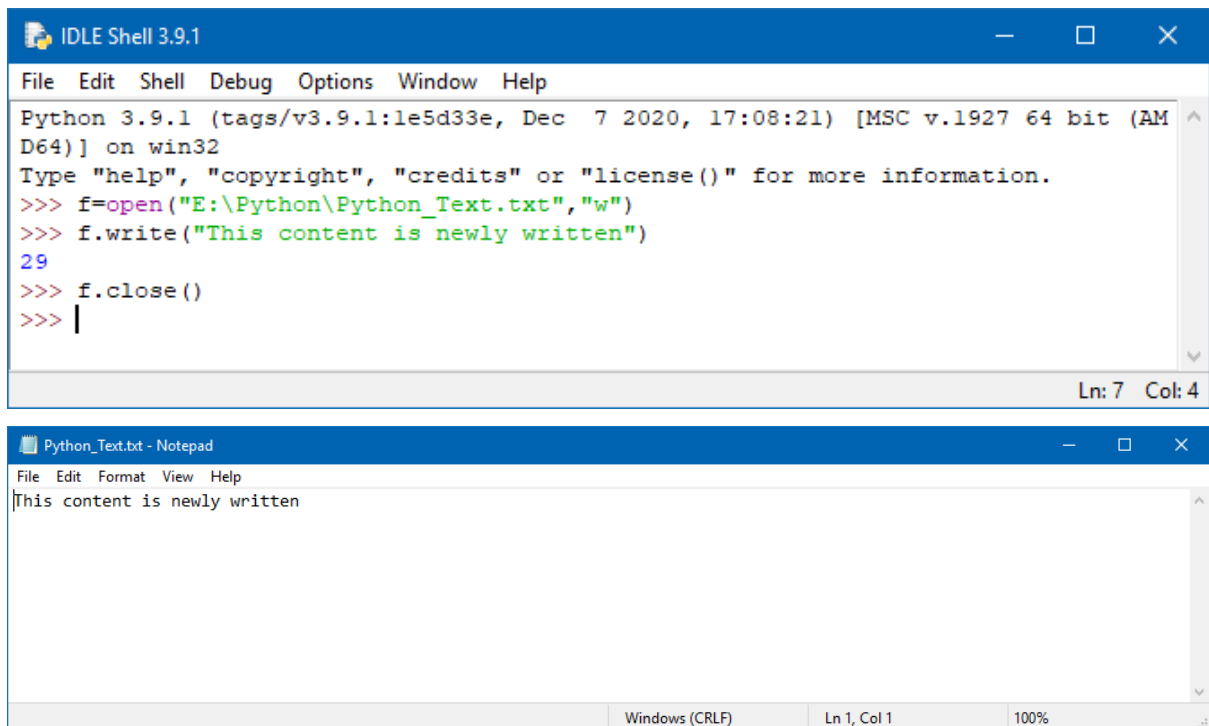
Now let us see how to write contents to the open file. Let us open the file in write mode. And add some content to Python_Text.txt

```
>>> f=open("E:\Python\Python_Text.txt","w")
```

```
>>> f.write("This content is newly written")
```

29

```
>>> f.close()
```



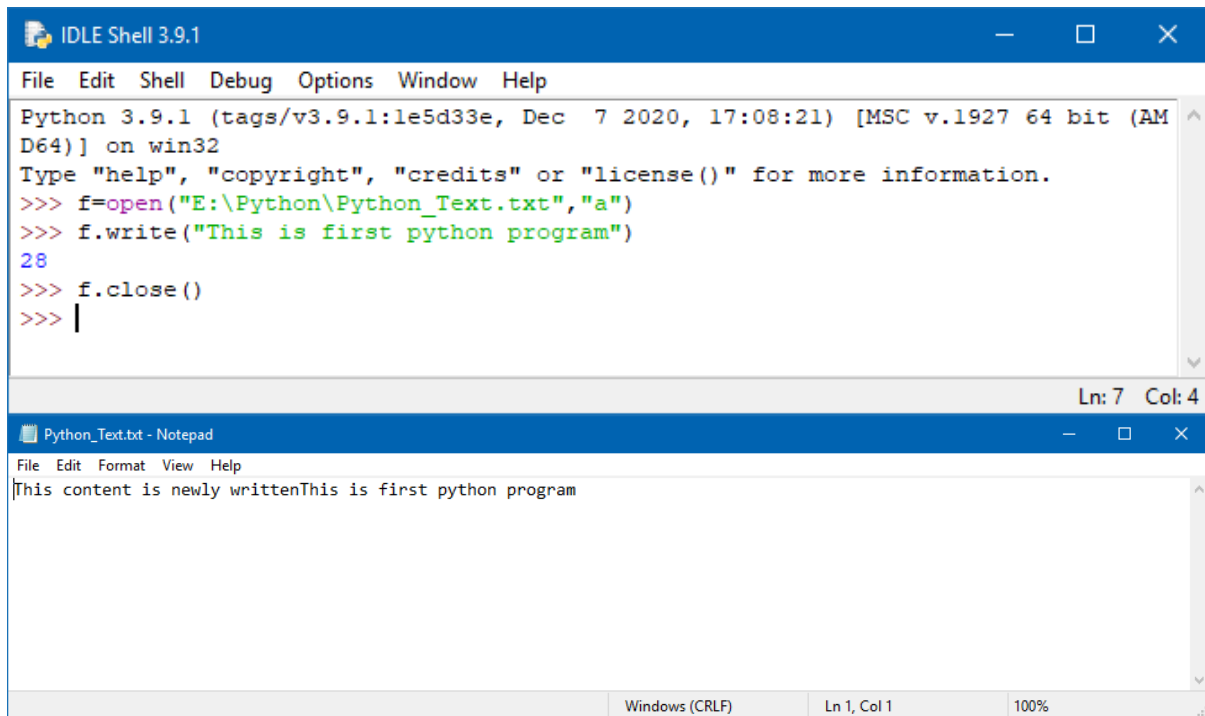
Here you can see that the content of the file is changed, that means the file is overwritten in write mode. `f.close()` is used to close an open file, closing a file informs the system that the writing operation is done and makes the file available for reading. Now let us see how to append contents to a file, in order to append contents to a file, one must open the file in append mode and add contents to it. Mode parameter "a" can be used to open a file in append mode. See below.

```
>>> f=open("E:\Python\Python_Text.txt","a")
```

```
>>> f.write("This is first python program")
```

28

```
>>> f.close()
```

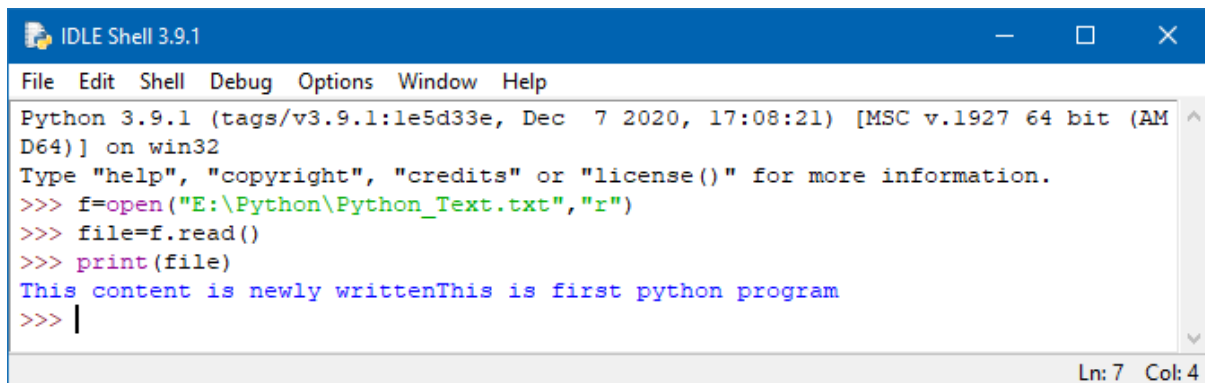
5.2. READING AND DISPLAYING FILE CONTENTS

One can use read method to read the contents of the file, and then to display the contents of the file. Simple read method does not have any arguments. Let us see an example. We will open the file in read mode and display contents.

```
>>> f=open("E:\Python\Python_Text.txt","r")
>>> file=f.read()
>>> print(file)
```

This content is newly writtenThis is first python program

```
>>>
```



Up on reading you must close the file using `f.close()`

You can pass the number of characters to be read, via read method. Let us see.

```
>>> f=open("E:\Python\Python_Text.txt","r")
```

```
>>> file=f.read(6)
```

```
>>> print(file)
```

This c

```
>>>
```

Now let us see how to copy contents of one file to another. We have to open one file in read mode and other in write mode.

```
def filecopy(file1,file2):
```

```
    f1 = open(file1, "r")
```

```
    f2 = open(file2, "w")
```

```
    while True:
```

```
        text = f1.read()
```

```
        if text == "":
```

```
            break
```

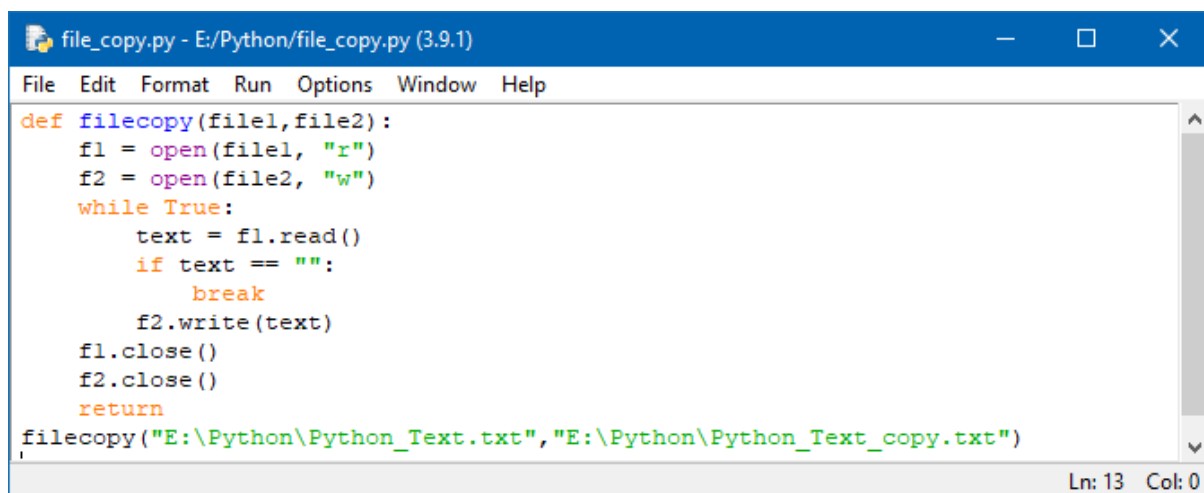
```
        f2.write(text)
```

```
    f1.close()
```

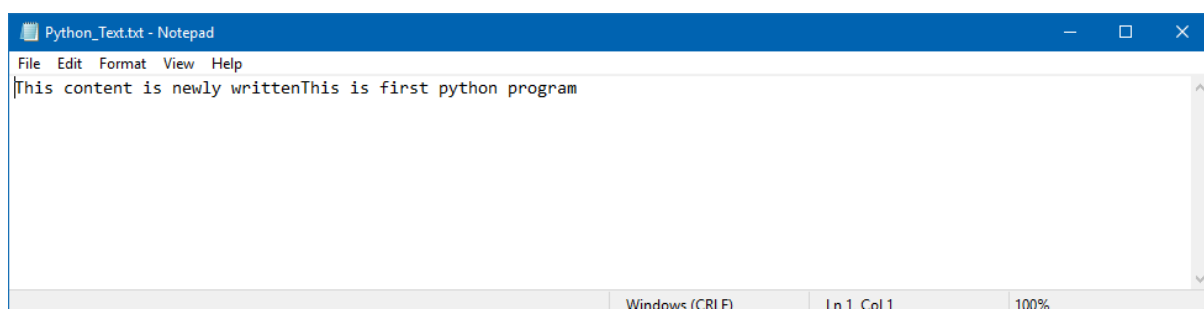
```
    f2.close()
```

```
    return
```

```
filecopy("E:\Python\Python_Text.txt","E:\Python\Python_Text_copy.txt")
```

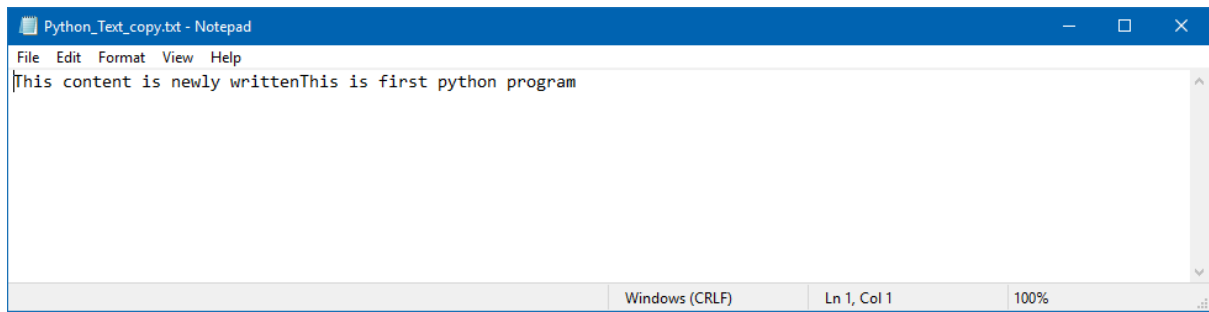
A screenshot of a Python IDE window titled 'file_copy.py - E:/Python/file_copy.py (3.9.1)'. The window has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The code editor contains the following Python code:

```
def filecopy(file1,file2):  
    f1 = open(file1, "r")  
    f2 = open(file2, "w")  
    while True:  
        text = f1.read()  
        if text == "":  
            break  
        f2.write(text)  
    f1.close()  
    f2.close()  
    return  
filecopy("E:\Python\Python_Text.txt","E:\Python\Python_Text_copy.txt")
```

The status bar at the bottom right shows 'Ln: 13 Col: 0'.A screenshot of a Notepad window titled 'Python_Text.txt - Notepad'. The window has a menu bar with 'File', 'Edit', 'Format', 'View', and 'Help'. The text area contains the following text:

This content is newly writtenThis is first python program

The status bar at the bottom shows 'Windows (CRLF)', 'Ln 1, Col 1', and '100%'.



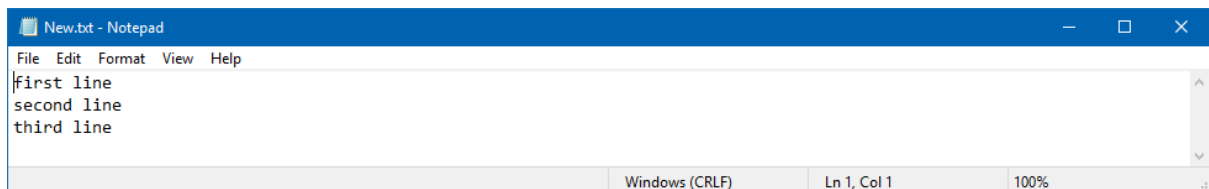
A text file can be read line by line using `readline()` method. Let us create text file with multiple lines. In order to create files with multiple lines, the new line character can be used.

```
>>> f.close()
>>> f=open("E://Python/New.txt","w")
>>> f.write("first line\nsecond line\nthird line")
```

33

```
>>> f.close()
>>> f=open("E://Python/New.txt","r")
>>> f.read()
'first line\nsecond line\nthird line'
>>>
```

```
>>> f.close()
>>> f=open("E://Python/New.txt","w")
>>> f.write("first line\nsecond line\nthird line")
33
>>> f.close()
>>> f=open("E://Python/New.txt","r")
>>> f.read()
'first line\nsecond line\nthird line'
>>>
```



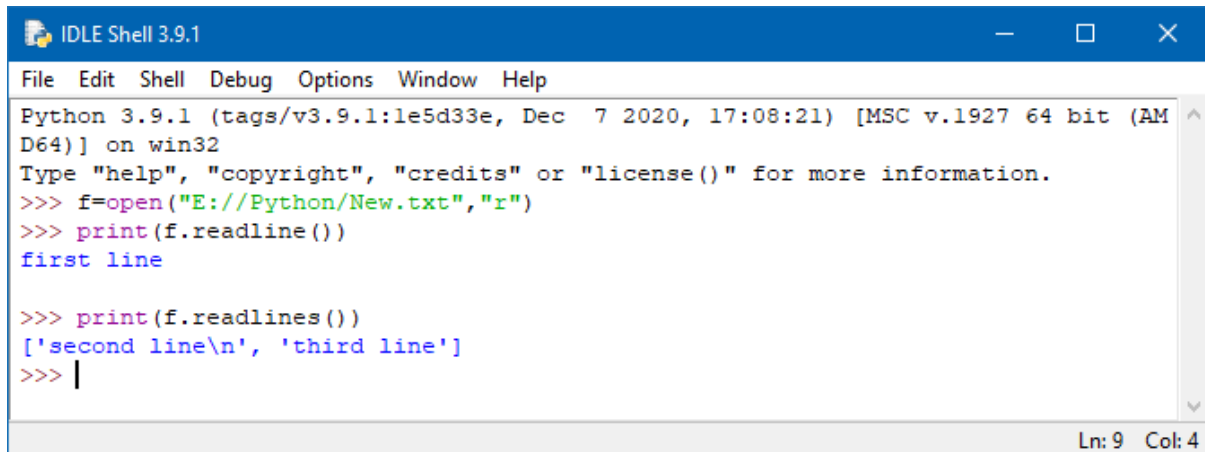
`readline` method prints the first line of the text file.

```
>>> f=open("E://Python/New.txt","r")
>>> print(f.readline())
first line
```

readlines returns all of the remaining lines as a list of strings

```
>>> print(f.readlines())
```

```
['second line\n', 'third line']
```

A screenshot of the IDLE Shell 3.9.1 window. The window has a blue title bar with the text 'IDLE Shell 3.9.1' and standard window controls. Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following Python code:

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> f=open("E://Python/New.txt","r")
>>> print(f.readline())
first line

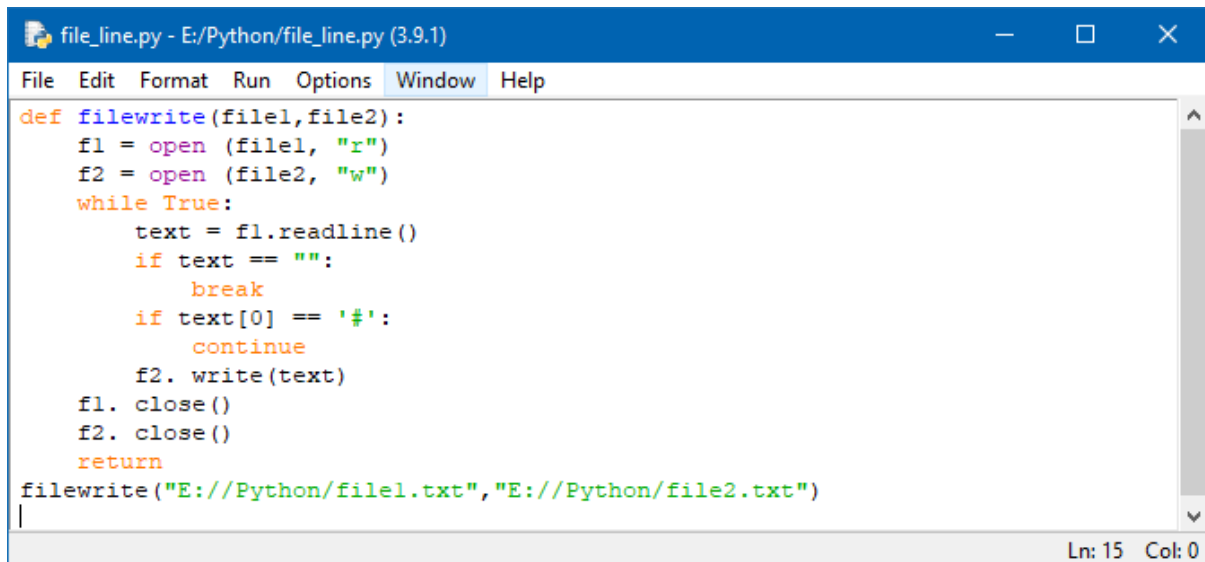
>>> print(f.readlines())
['second line\n', 'third line']
>>> |
```

The status bar at the bottom right indicates 'Ln: 9 Col: 4'.

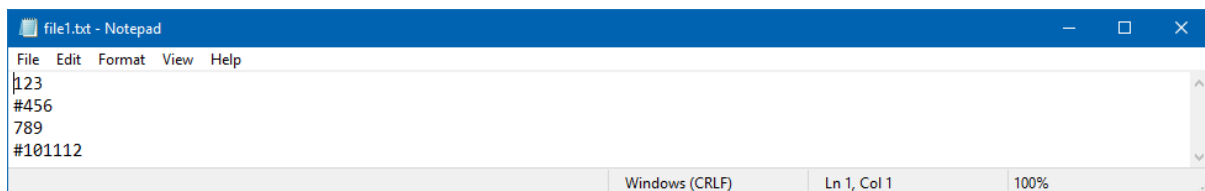
Excercise: Write a program to copy one file to another, by omitting the lines starting from #

Solution:

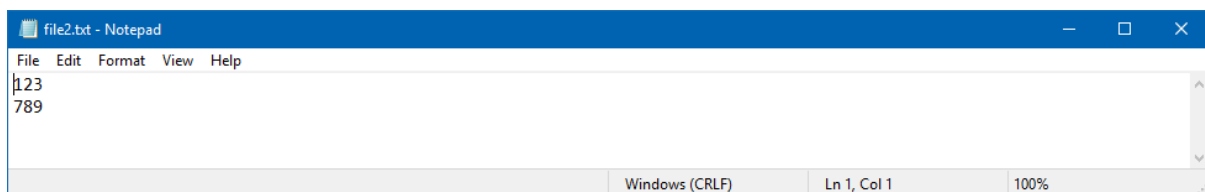
```
def filewrite(file1,file2):
    f1 = open (file1, "r")
    f2 = open (file2, "w")
    while True:
        text = f1.readline()
        if text == '':
            break
        if text[0] == '#':
            continue
        f2. write(text)
    f1. close()
    f2. close()
    return
filewrite("E://Python/file1.txt","E://Python/file2.txt")
```



```
def filewrite(file1,file2):
    f1 = open (file1, "r")
    f2 = open (file2, "w")
    while True:
        text = f1.readline()
        if text == "":
            break
        if text[0] == '#':
            continue
        f2. write(text)
    f1. close()
    f2. close()
    return
filewrite("E://Python/file1.txt","E://Python/file2.txt")
```



```
123
456
789
101112
```



```
123
789
```

5.3. PICKLING

We have seen the values are stored to a text file as strings, The problem is that when you read the file back you get strings. The problem is that when you read the value back, you get a string. The original type information has been lost. The solution is pickling, it “preserves” data structures. The pickle module contains the necessary commands to dumb and read values preserving data structures. To use it, import pickle and then open the file in the usual way:

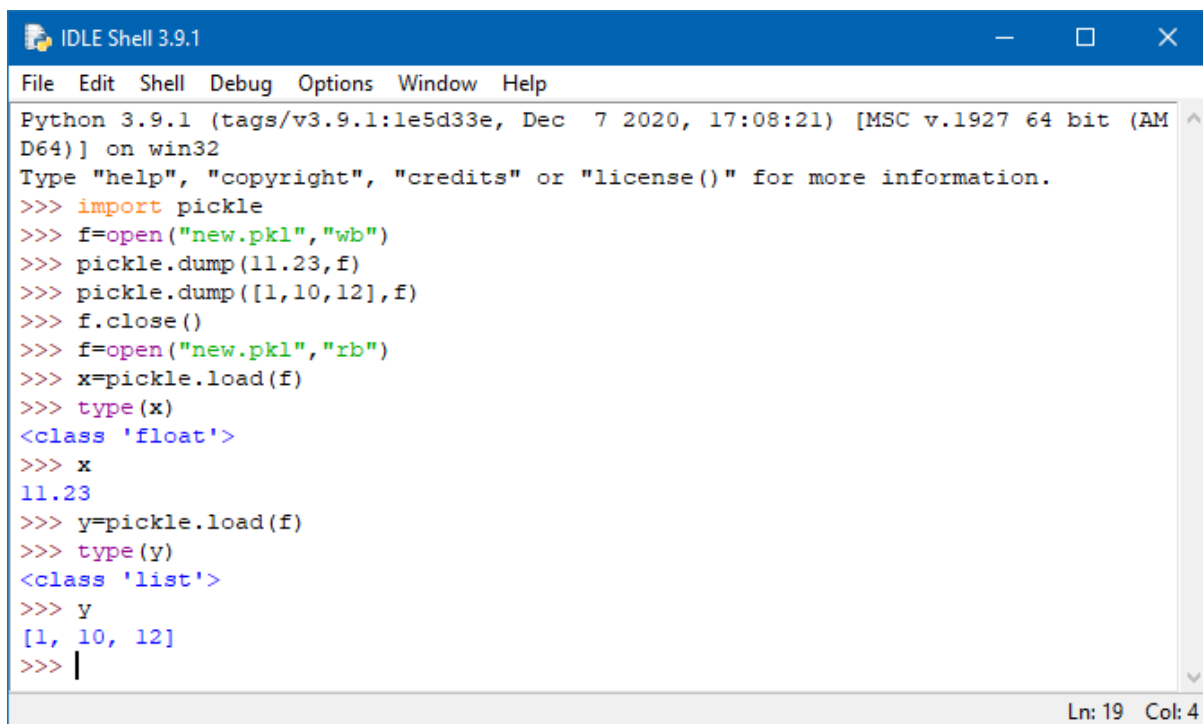
```
>>> import pickle
>>> f=open("new.pkl","wb")
```

Use write byte mode here. To store a data structure, use the dump method and then close the file in the usual way:

```
>>> pickle.dump(11.23,f)
>>> pickle.dump([1,10,12],f)
>>> f.close()
```

Now open the file in read mode and use pickle. Load method for loading the values.

```
>>> f=open("new.pkl","rb")
>>> x=pickle.load(f)
>>> type(x)
<class 'float'>
>>> x
11.23
>>> y=pickle.load(f)
>>> type(y)
<class 'list'>
>>> y
[1, 10, 12]
```

A screenshot of the IDLE Shell 3.9.1 window. The window has a blue title bar with the text "IDLE Shell 3.9.1" and standard window controls. Below the title bar is a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main text area shows the following code being executed:

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import pickle
>>> f=open("new.pkl","wb")
>>> pickle.dump(11.23,f)
>>> pickle.dump([1,10,12],f)
>>> f.close()
>>> f=open("new.pkl","rb")
>>> x=pickle.load(f)
>>> type(x)
<class 'float'>
>>> x
11.23
>>> y=pickle.load(f)
>>> type(y)
<class 'list'>
>>> y
[1, 10, 12]
>>> |
```

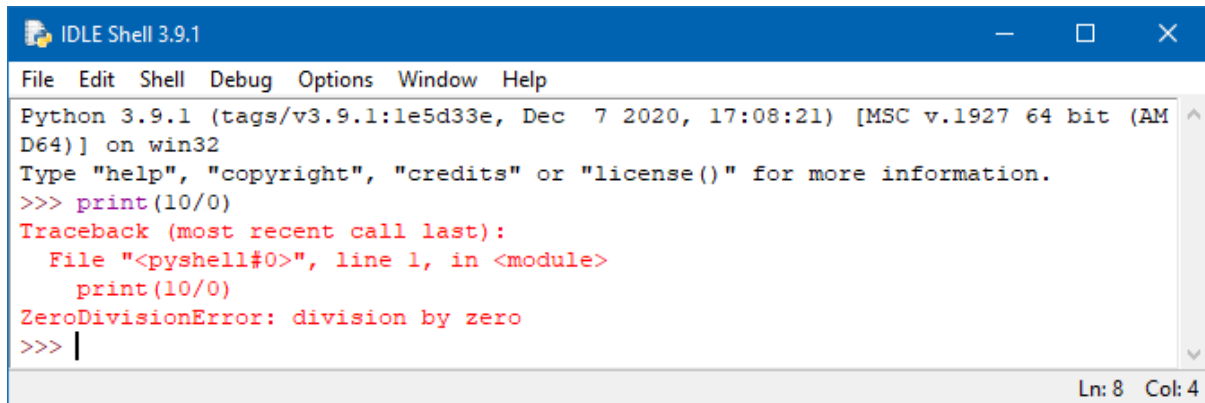
 The status bar at the bottom right shows "Ln: 19 Col: 4".

5.4. EXCEPTIONS

Run time errors are called exceptions. Whenever a run time error occurs, the program stop its execution and prints an error message.

Divides by zero is an example exception

```
>>> print(10/0)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print(10/0)
ZeroDivisionError: division by zero
>>>
```

A screenshot of the IDLE Shell 3.9.1 window. The window has a blue title bar with the text 'IDLE Shell 3.9.1' and standard window controls. Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following text: 'Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32', 'Type "help", "copyright", "credits" or "license()" for more information.', '>>> print(10/0)', 'Traceback (most recent call last):', ' File "<pyshell#0>", line 1, in <module>', ' print(10/0)', 'ZeroDivisionError: division by zero', and '>>> |'. The status bar at the bottom right shows 'Ln: 8 Col: 4'.

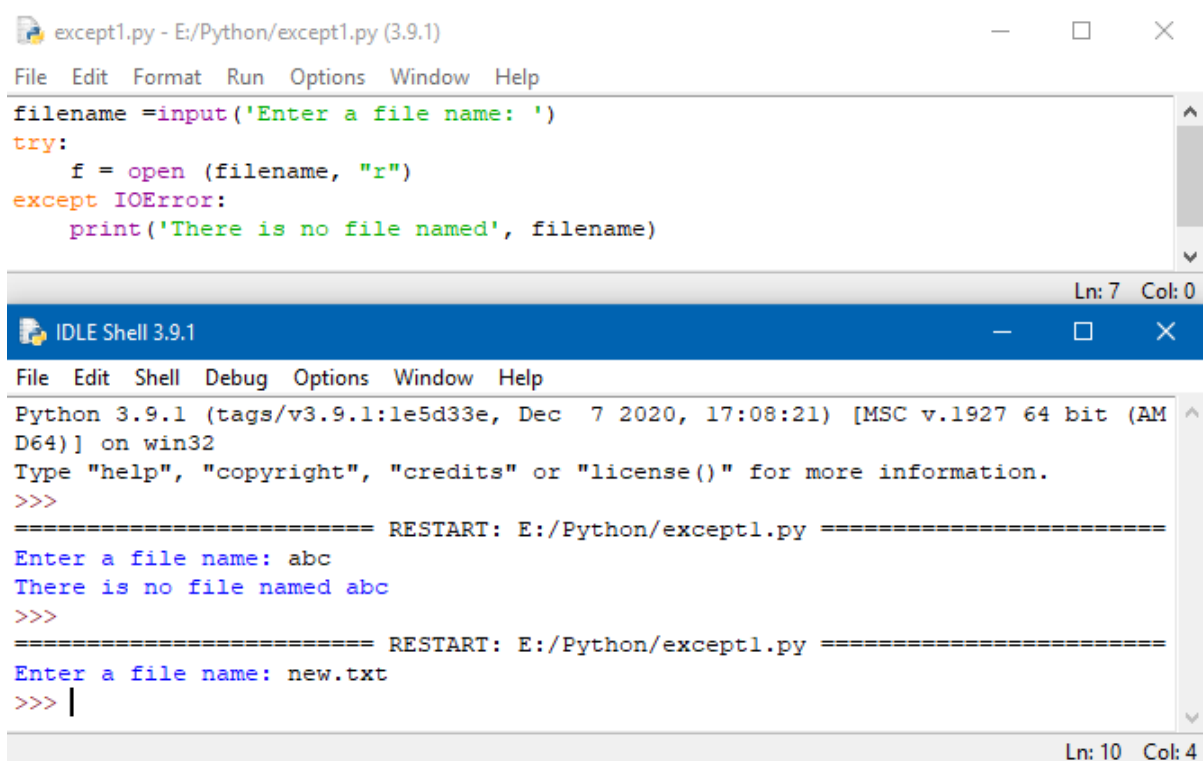
The error message has two parts separated by a colon. The first part of error message is the exception name and second part specifies the details of error. If we want to execute an operation that leads to an exception without stopping the program we can handle the exception using **try** and **except** statements.

Consider the following example for IO exception

Example:

```
filename =input('Enter a file name: ')
try:
    f = open (filename, "r")
except IOError:
    print('There is no file named', filename)
```

The **try** statement executes the statements in the first block. If no exceptions occur, it ignores the **except** statement. If an exception of type **IOError** occurs, it executes the statements in the except branch and then continues.



The screenshot shows a Python IDE window titled 'except1.py - E:/Python/except1.py (3.9.1)'. The code in the editor is:

```
filename =input('Enter a file name: ')
try:
    f = open (filename, "r")
except IOError:
    print('There is no file named', filename)
```

The IDLE Shell 3.9.1 window below shows the execution output:

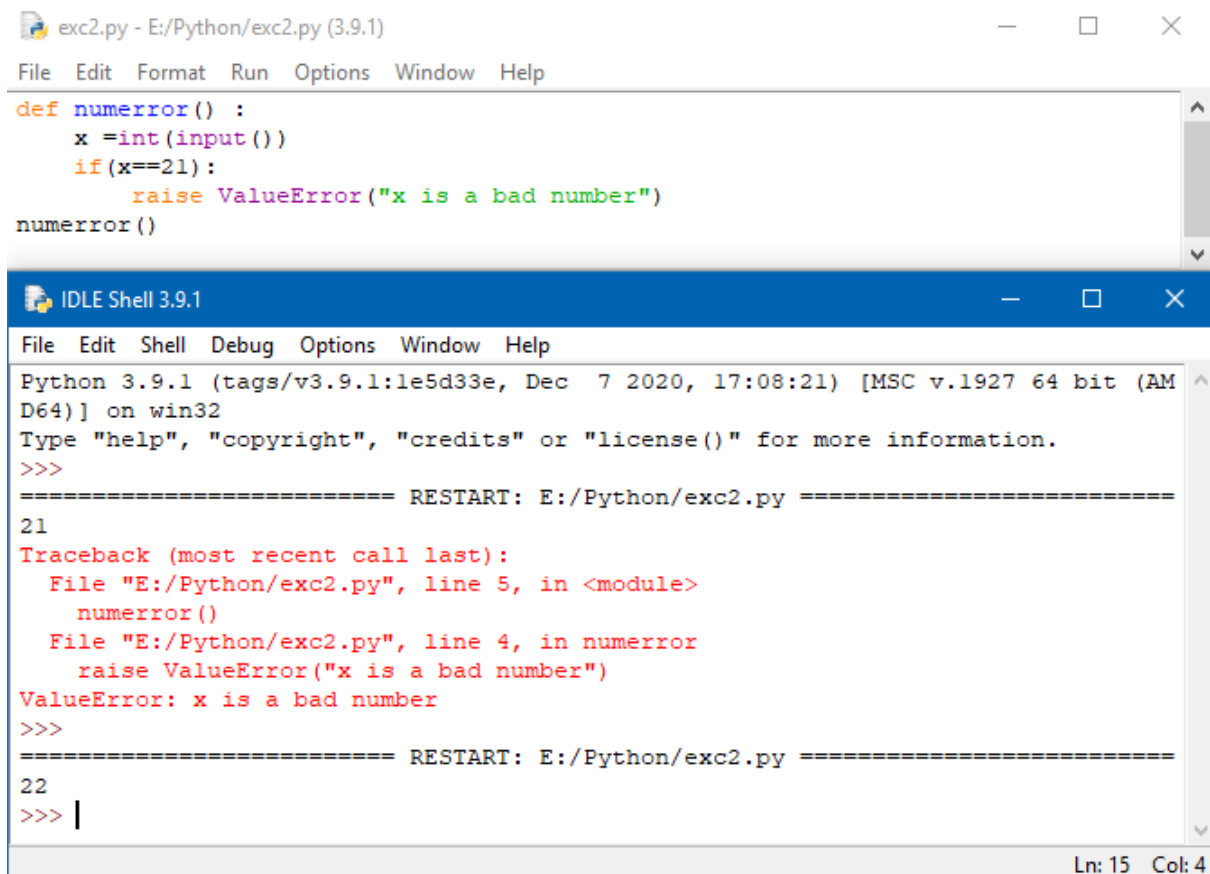
```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: E:/Python/except1.py =====
Enter a file name: abc
There is no file named abc
>>>
===== RESTART: E:/Python/except1.py =====
Enter a file name: new.txt
>>> |
```

If your program detects an error condition, you can make it **raise** an exception. Here is an example that gets input from the user and checks for the value 21.

Assuming that 21 is not valid input for some reason, we raise an exception. The **raise** statement takes two arguments: the exception type and specific information about the error.

Example:

```
def numerror() :
    x=int(input())
    if(x==21):
        raise ValueError("x is a bad number")
numerror()
```

The image shows a screenshot of a Python IDE window titled "exc2.py - E:/Python/exc2.py (3.9.1)". The window has a menu bar with "File", "Edit", "Format", "Run", "Options", "Window", and "Help". The code in the editor is as follows:

```
def numerror() :  
    x =int(input())  
    if(x==21):  
        raise ValueError("x is a bad number")  
numerror()
```

Below the editor is the "IDLE Shell 3.9.1" window, which has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The shell displays the following output:

```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: E:/Python/exc2.py =====  
21  
Traceback (most recent call last):  
  File "E:/Python/exc2.py", line 5, in <module>  
    numerror()  
  File "E:/Python/exc2.py", line 4, in numerror  
    raise ValueError("x is a bad number")  
ValueError: x is a bad number  
>>>  
===== RESTART: E:/Python/exc2.py =====  
22  
>>> |
```

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

CHAPTER 6

CLASSES & OBJECTS

6.1. CLASSES

Classes can be considered as user defined compound type. Since python is an object oriented programming methodology, almost everything in python programming is considered as an object. You can consider a class as an object creator. Now let us see the syntax of a class definition.

Syntax:

```
class class_name:  
    Methods and Variables
```

Although class definitions can be placed anywhere in a program, but it's convenient to place them near the beginning (after import statements) . There should be at least one statement or method inside class definition because, a compound statement must have something in its body . Now let us create a class named "classA"

```
class classA:  
    x=10
```

This definition creates a new class called classA. And x is a member variable. Now let us create an object and access the members inside the class. Here a class object can be created using

```
obj=classA()
```

This process is called as object instantiation . Now let us see how to access the member x using the object obj, you can use dot operation to access member functions or variables. See below.

```
class classA:  
    x=10  
obj=classA()  
print(obj.x)
```

```
class.py - E:/Python/class.py (3.9.1)
File Edit Format Run Options Window Help
class classA:
    x=10
obj=classA()
print(obj.x)
|
Ln: 5 Col: 0
```

```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: E:/Python/class.py =====
10
>>> |
Ln: 6 Col: 4
```

You can add new data items to objects using dot notation. That is

obj.y=20

print(obj.y) will produce 20

```
>>> print(obj)
<__main__.classA object at 0x000002443BAE3F10>
```

The result indicates that obj is an instance of the classA class and it was defined in main . 0x000002443BAE3F10 is the unique identifier for this object

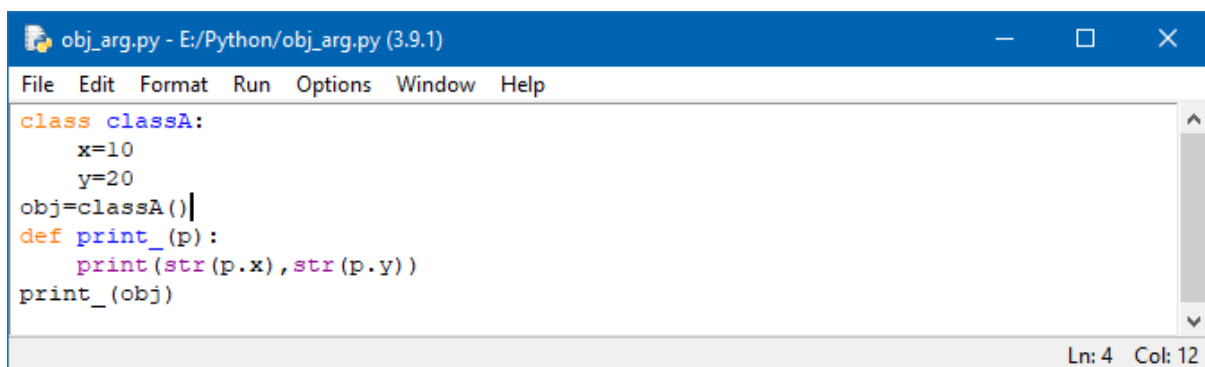
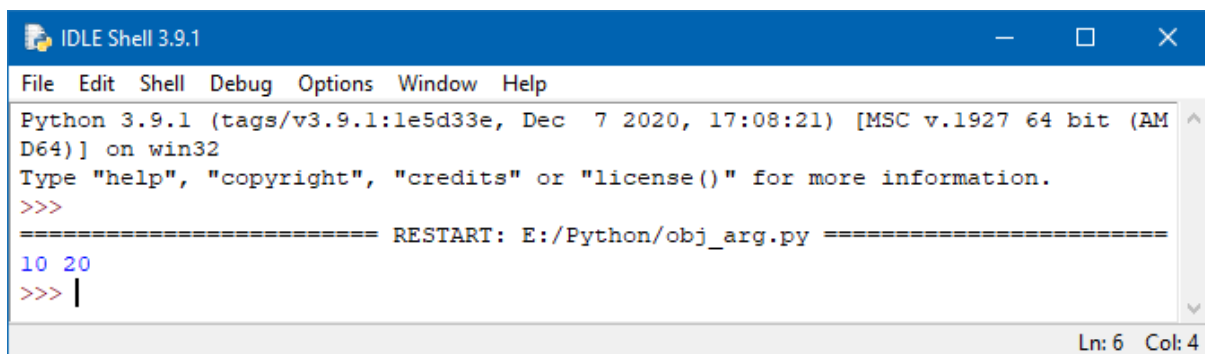
```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> class classA:
    x=10

>>> obj=classA()
>>> print(obj)
<__main__.classA object at 0x000002443BAE3F10>
>>>
Ln: 8 Col: 12
```

6.1.1. Passing Objects as Arguments in Function

You can pass an instance/object as an argument in functions. Let us see an example.

```
class classA:
    x=10
    y=20
obj=classA()
def print_(p):
    print(str(p.x),str(p.y))
print_(obj)
```

A screenshot of a Python IDE window titled 'obj_arg.py - E:/Python/obj_arg.py (3.9.1)'. The window contains the same Python code as shown in the previous block. The status bar at the bottom right indicates 'Ln: 4 Col: 12'.A screenshot of the IDLE Shell window titled 'IDLE Shell 3.9.1'. It shows the output of the program: '10 20'. The status bar at the bottom right indicates 'Ln: 6 Col: 4'.

Here in this program, the class object obj is passed as parameter to the function print_

6.1.2. Shallow Equality and Deep Equality

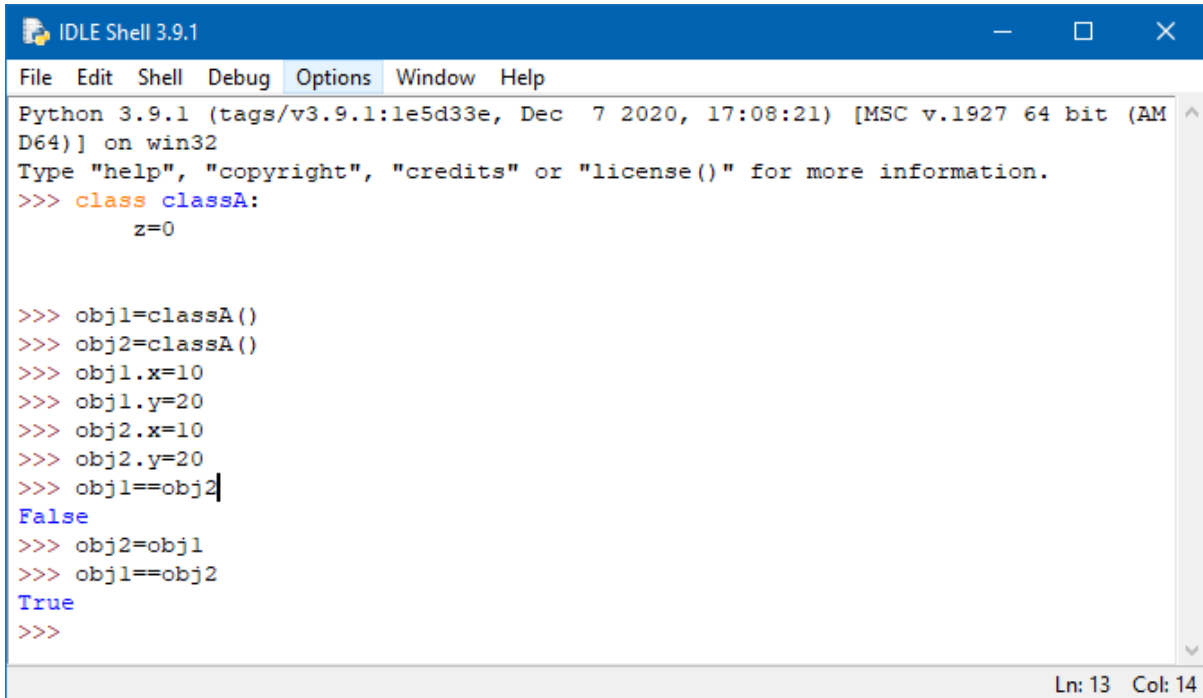
Even though obj1 and obj2 contain the same variables and values, they are not the same object and produce false while comparison. If we assign obj1 to obj2, then the two variables are aliases of the same object. See the below implementation.

```
>>> class classA:
    z=0
>>> obj1=classA()
>>> obj2=classA()
>>> obj1.x=10
>>> obj1.y=20
```

```

>>> obj2.x=10
>>> obj2.y=20
>>> obj1==obj2
False
>>> obj2=obj1
>>> obj1==obj2
True

```



```

IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> class classA:
    z=0

>>> obj1=classA()
>>> obj2=classA()
>>> obj1.x=10
>>> obj1.y=20
>>> obj2.x=10
>>> obj2.y=20
>>> obj1==obj2
False
>>> obj2=obj1
>>> obj1==obj2
True
>>>
Ln: 13 Col: 14

```

This type of equality is called shallow equality because it compares only the references, not the contents of the objects. To compare the contents of the objects deep equality need to be used. We can write a function to perform comparison of contents of the object. Let's call the function as obj_comp

```

def obj_comp (obj1, obj2) :
    return (obj1.x == obj2.x) and (obj1.y == obj2.y)

```

Now if we create two different objects that contain the same data, we can use obj_comp to find out if they represent the same data.

```

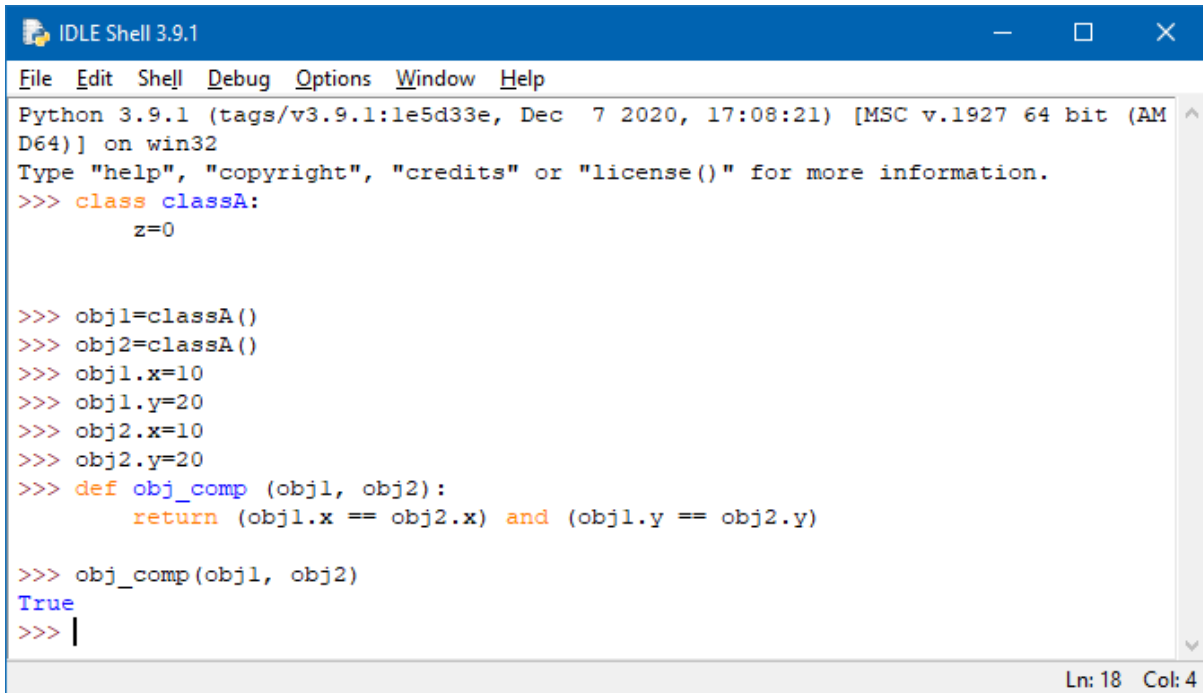
>>> p1 >>> class classA:
    z=0
>>> obj1=classA()
>>> obj2=classA()
>>> obj1.x=10
>>> obj1.y=20
>>> obj2.x=10
>>> obj2.y=20

```

```
>>> def obj_comp (obj1, obj2):
    return (obj1.x == obj2.x) and (obj1.y == obj2.y)

>>> obj_comp(obj1, obj2)
True
>>>
```

The comparison returned true after checking the contents of the object.



```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> class classA:
    z=0

>>> obj1=classA()
>>> obj2=classA()
>>> obj1.x=10
>>> obj1.y=20
>>> obj2.x=10
>>> obj2.y=20
>>> def obj_comp (obj1, obj2):
    return (obj1.x == obj2.x) and (obj1.y == obj2.y)

>>> obj_comp(obj1, obj2)
True
>>> |
```

Ln: 18 Col: 4

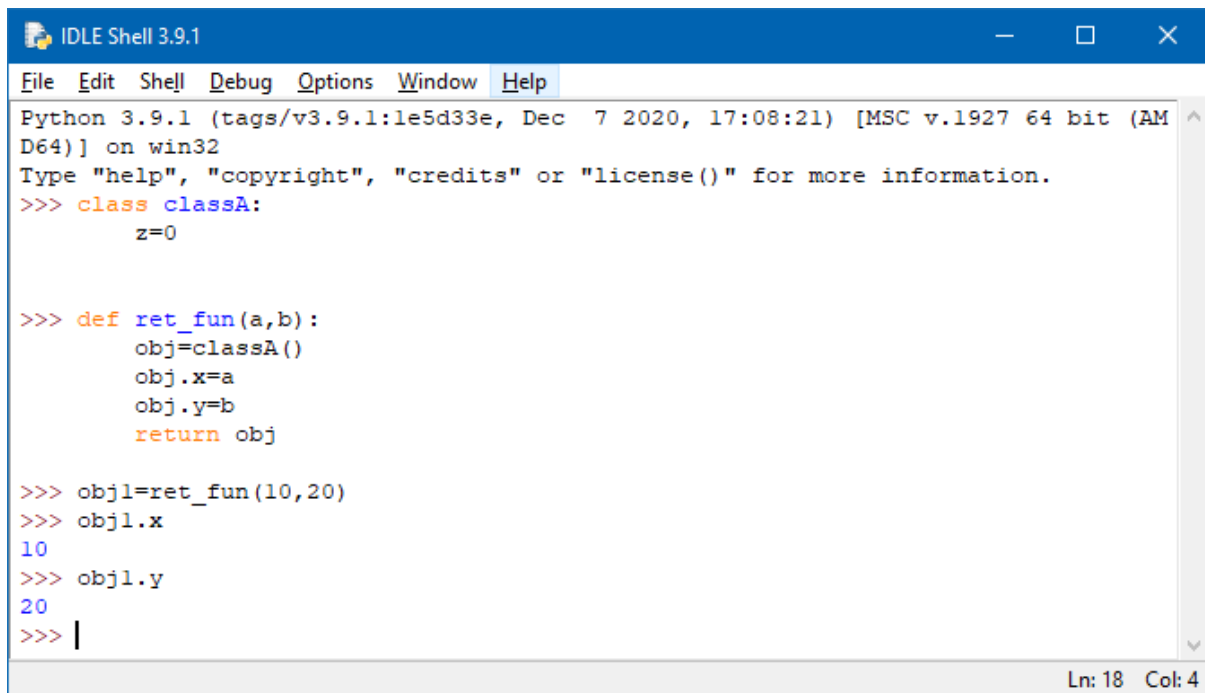
6.1.3. Objects as Return Values

Functions can return instances or objects. See the below example, where the function creates an object and return the object.

```
>>> class classA:
    z=0
>>> def ret_fun(a,b):
    obj=classA()
    obj.x=a
    obj.y=b
    return obj

>>> obj1=ret_fun(10,20)
>>> obj1.x
10
>>> obj1.y
20
```

Here classA object is created inside the function ret_fun and returned the object and its assigned values. And printed outside the function.



```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> class classA:
    z=0

>>> def ret_fun(a,b):
    obj=classA()
    obj.x=a
    obj.y=b
    return obj

>>> obj1=ret_fun(10,20)
>>> obj1.x
10
>>> obj1.y
20
>>> |
```

Ln: 18 Col: 4