

PYTHON

Introduction

In order to tell the computer „what you want to do“, we write a program in a language which computer can understand. Though there are many different programming languages such as BASIC, Pascal, C, C++, Java, Haskell, Ruby, Python, etc. but we will study Python in this workshop.

Before learning the technicalities of Python, let's get familiar with it.

Python was created by Guido Van **Rossum** when he was working at CWI (Centrum Wiskunde&Informatica) which is a National Research Institute for Mathematics and Computer Science in Netherlands. The language was released in 1991. Python got its name from a BBC comedy series from seventies- "Monty Python"s Flying Circus". Python can be used to follow both Procedural approach and Object Oriented approach of programming. It is free to use.

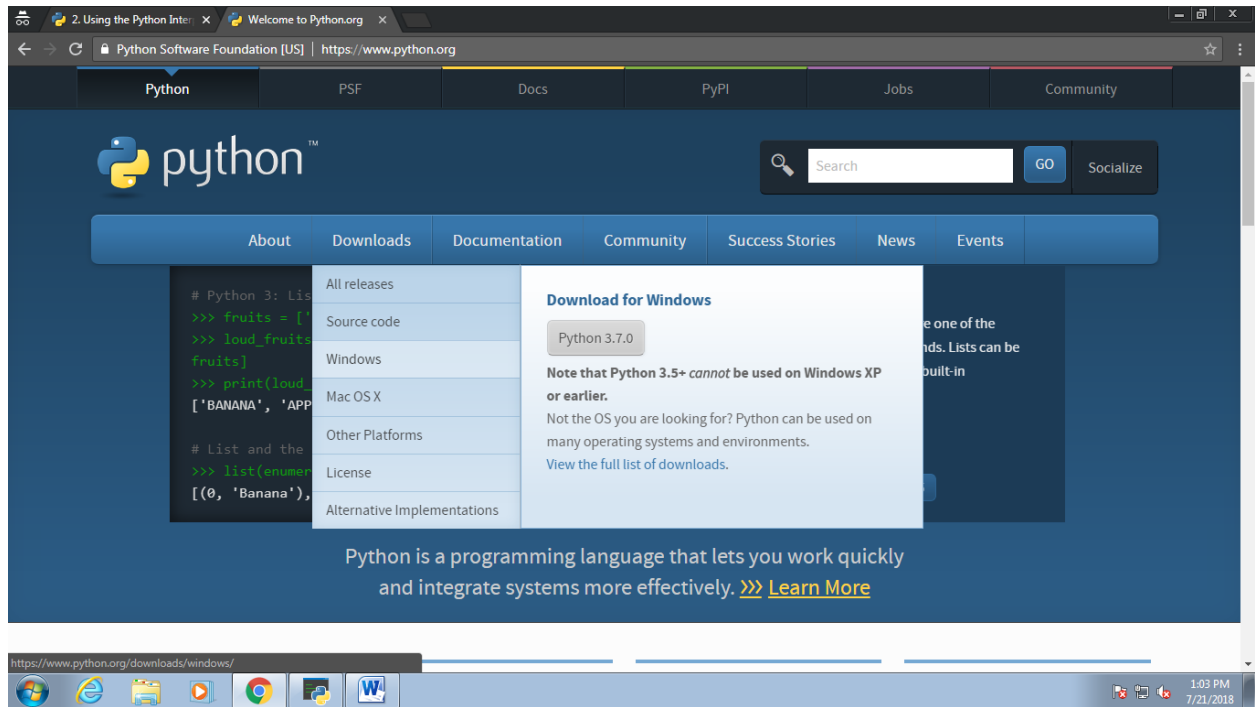
Python Features

Python's features include –

- **Easy-to-learn** – Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** – Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** – Python's source code is fairly easy-to-maintain.
- **Interactive Mode** – Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** – Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** – You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** – Python provides interfaces to all major commercial databases.
- **GUI Programming** – Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

Downloading Python:

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, <https://www.python.org/>, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.



First Step with Python

We are continuously saying that Python is a programming language but don't know what a program is? Therefore, let's start Python by understanding Program.

A program is a sequence of instructions that specifies how to perform a Computation. The Computation might be mathematical or working with text.

To write and run Python program, we need to have Python interpreter installed in our computer. **IDLE** (GUI integrated) is the standard, most popular Python development environment. IDLE is an acronym of Integrated Development Environment. It lets edit, run, browse and debug Python Programs from a single interface. This environment makes it easy to write programs.

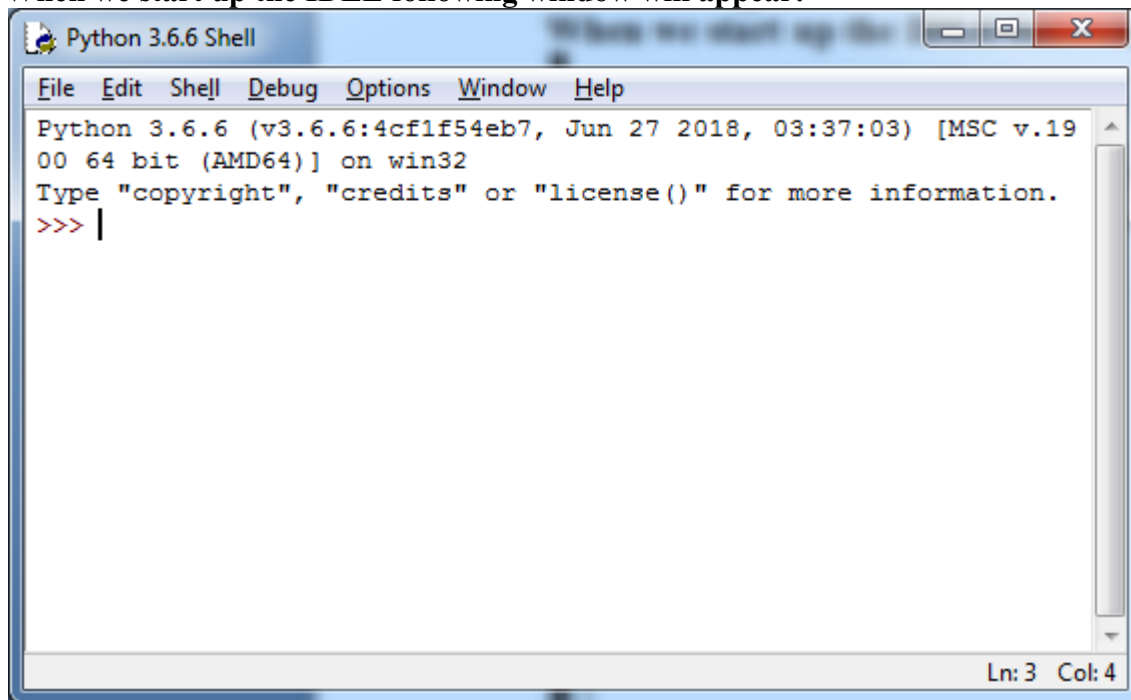
Python shell can be used in two ways, viz., interactive mode and script mode. Where

Interactive Mode, as the name suggests, allows us to interact with OS; script mode let us create and edit python source file. Now, we will first start with interactive mode. Here, we type a Python statement and the interpreter displays the result(s) immediately.

➤ Interactive Mode

For working in the interactive mode, we will start Python on our computer.

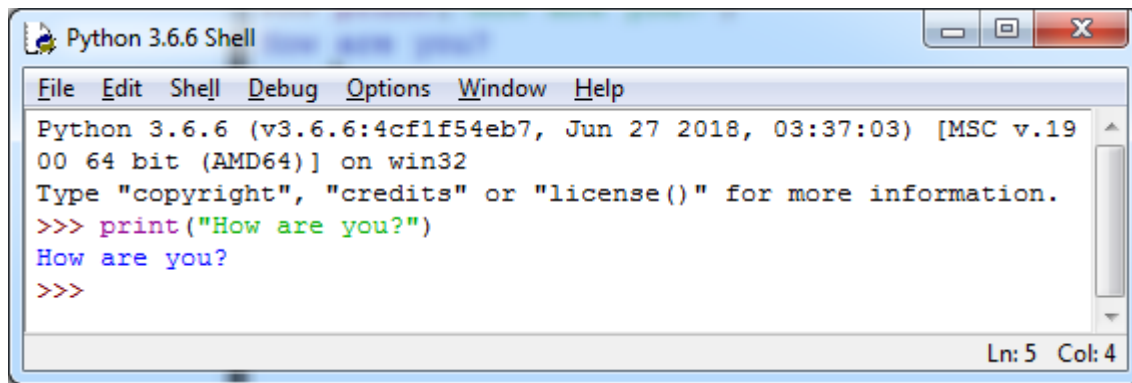
When we start up the IDLE following window will appear:



What we see is a welcome message of Python interpreter with revision details and the Python prompt, i.e., „>>>“. This is a primary prompt indicating that the interpreter is expecting a python command. There is secondary prompt also which is „...“ indicating that interpreter is waiting for additional input to complete the current statement.

Interpreter uses prompt to indicate that it is ready for instruction. Therefore, we can say, if there is prompt on screen, it means IDLE is working in interactive mode. We type Python expression / statement / command after the prompt and Python immediately responds with the output of it. Let's start with typing print **“How are you”** after the prompt.

```
>>>print ("How are you?")  
How are you?
```

A screenshot of a Windows-style application window titled "Python 3.6.6 Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main text area contains the following text: "Python 3.6.6 (v3.6.6:4cf1f54eb7, Jun 27 2018, 03:37:03) [MSC v.1900 64 bit (AMD64)] on win32", "Type 'copyright', 'credits' or 'license()' for more information.", ">>> print('How are you?')", "How are you?", and ">>>". The status bar at the bottom right shows "Ln: 5 Col: 4".

```
Python 3.6.6 (v3.6.6:4cf1f54eb7, Jun 27 2018, 03:37:03) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print('How are you?')
How are you?
>>>
```

➤ Script Mode

In script mode, we type Python program in a file and then use the interpreter to execute the content from the file. Working in interactive mode is convenient for beginners and for testing small pieces of code, as we can test them immediately. But for coding more than few lines, we should always save our code so that we may modify and reuse the code.

Python, in interactive mode, is good enough to learn, experiment or explore, but its only drawback is that we cannot save the statements for further use and we have to retype all the statements to re-run them.

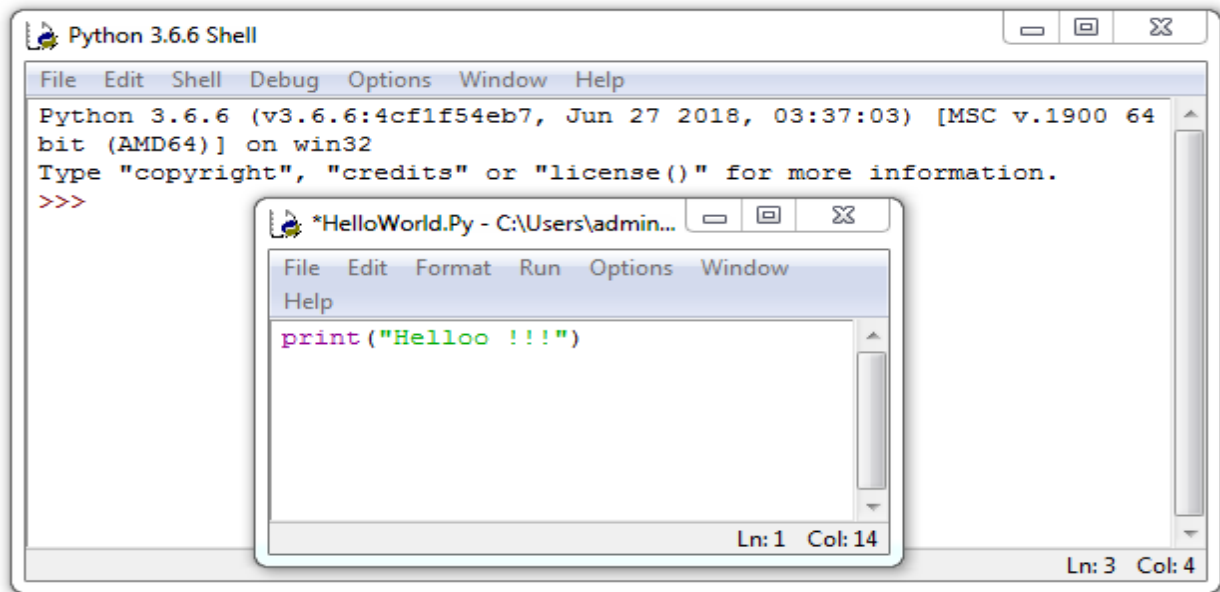
To create and run a Python script, we will use following steps in IDLE, if the script mode is not made available by default with IDLE environment.

1. File>Open OR File>New Window (for creating a new script file)
2. Write the Python code as function i.e. script
3. Save it (^S)
4. Execute it in interactive mode- by using RUN option (^F5)

If we write Example 1 in script mode, it will be written in the following way:

Step 1: File> New Window

Step 2:



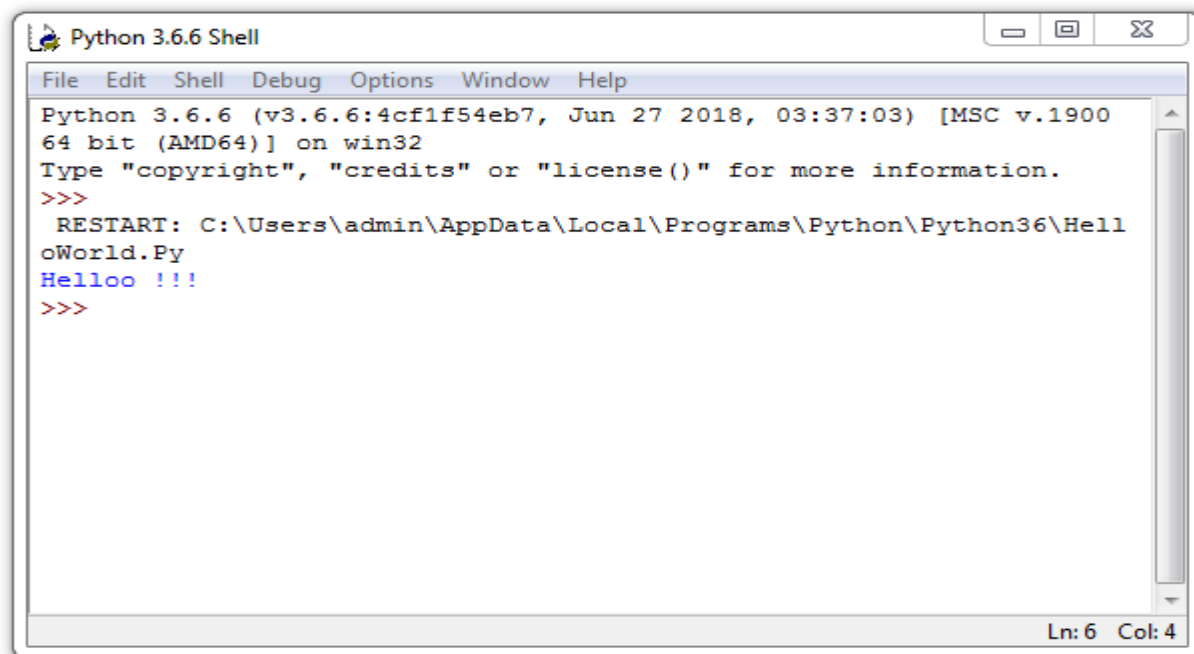
Step 3:

Use File > Save or File > Save As - option for saving the file

Step 4:

For execution, press ^F5,

OUTPUT:



Variables and Assignment:

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

➤ Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:

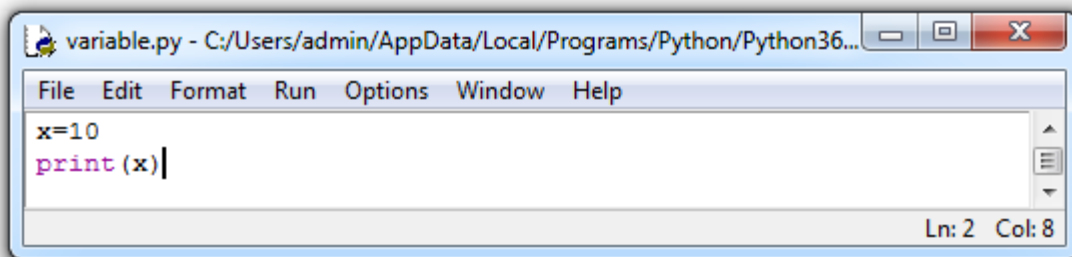
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

➤ Assigning Values to Variables

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

Example

Following example (variable.py) uses a variable to store an integer value and then print value of variable.



The screenshot shows a window titled 'variable.py - C:/Users/admin/AppData/Local/Programs/Python/Python36...'. The window contains a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The main text area contains two lines of code: 'x=10' and 'print(x)'. The status bar at the bottom right indicates 'Ln: 2 Col: 8'.

```
x=10
print(x)
```

➤ Multiple Assignments

Python allows you to assign a single value to several variables simultaneously.

For example –

```
a = b = c = 1
```

Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location.

Data Types

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has standard data types –

- Numbers
- Sequence
 - String
 - List
 - Tuple
- Dictionary

NOTE: To verify the type of any object in Python, use the `type()` function:

1. Number

Number data type stores Numerical Values. This data type is immutable i.e. value of its object cannot be changed (we will talk about this aspect later). These are of three different types:

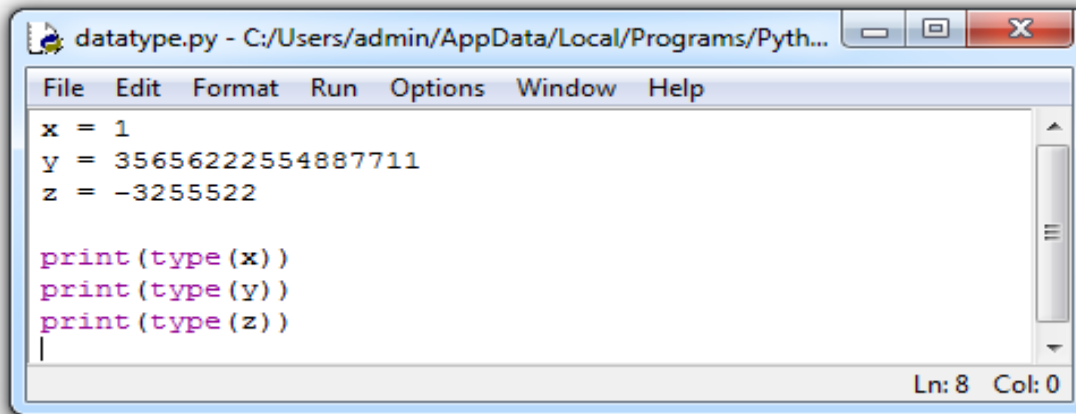
- a) Integer
- b) Float/floating point
- c) Complex

➤ int

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited

length.

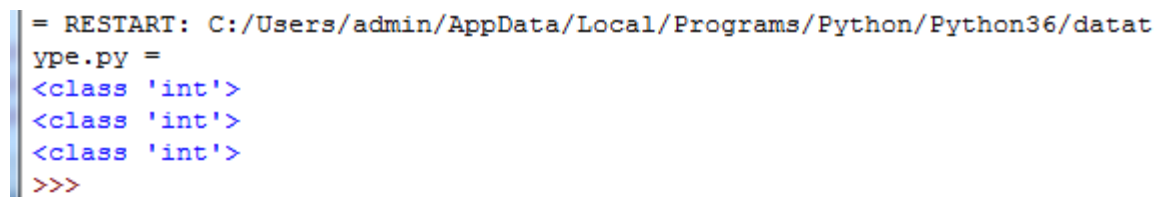
EXAMPLE:



```
datatype.py - C:/Users/admin/AppData/Local/Programs/Pyth...
File Edit Format Run Options Window Help
x = 1
y = 35656222554887711
z = -3255522

print(type(x))
print(type(y))
print(type(z))
|
Ln: 8 Col: 0
```

OUTPUT:

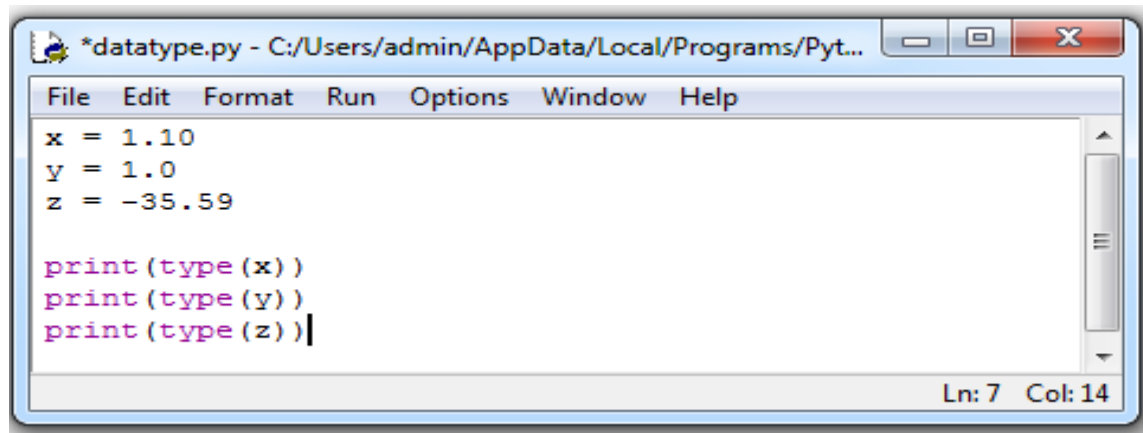


```
= RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python36/datat
ype.py =
<class 'int'>
<class 'int'>
<class 'int'>
>>>
```

➤ Float

Float, or "floating point number" is a number, positive or negative, containing one or More decimal.

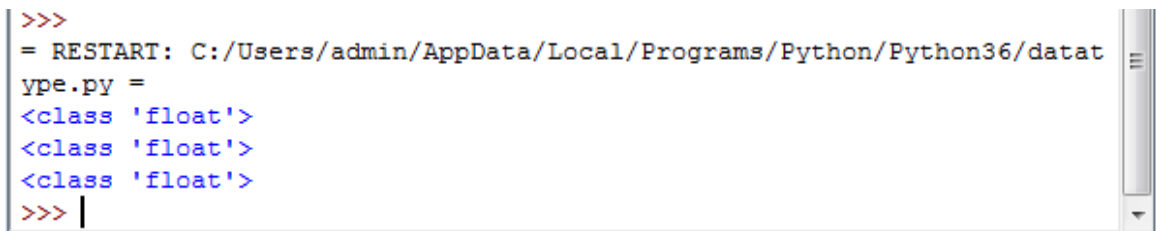
EXAMPLE:



```
*datatype.py - C:/Users/admin/AppData/Local/Programs/Pyt...
File Edit Format Run Options Window Help
x = 1.10
y = 1.0
z = -35.59

print(type(x))
print(type(y))
print(type(z))
Ln: 7 Col: 14
```

OUTPUT:

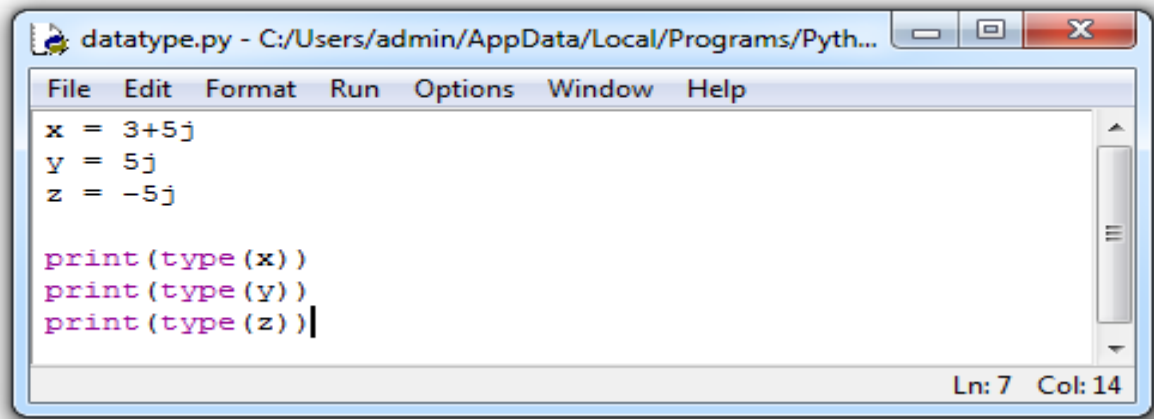


```
>>>
= RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python36/datatype.py =
<class 'float'>
<class 'float'>
<class 'float'>
>>> |
```

➤ Complex

Complex numbers are written with a "j" as the imaginary part:

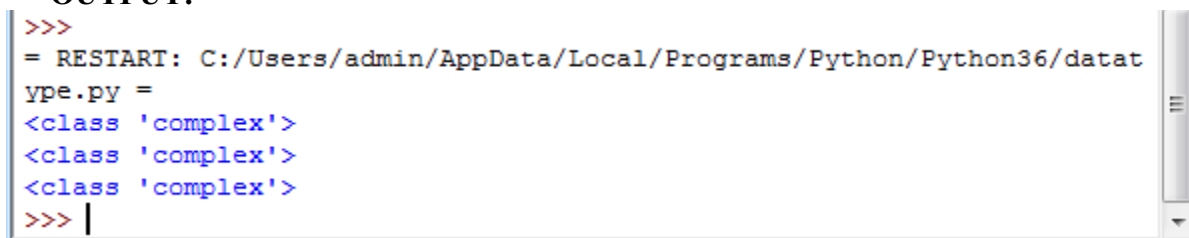
EXAMPLE:



```
datatype.py - C:/Users/admin/AppData/Local/Programs/Pyth...
File Edit Format Run Options Window Help
x = 3+5j
y = 5j
z = -5j

print(type(x))
print(type(y))
print(type(z))
Ln: 7 Col: 14
```

OUTPUT:



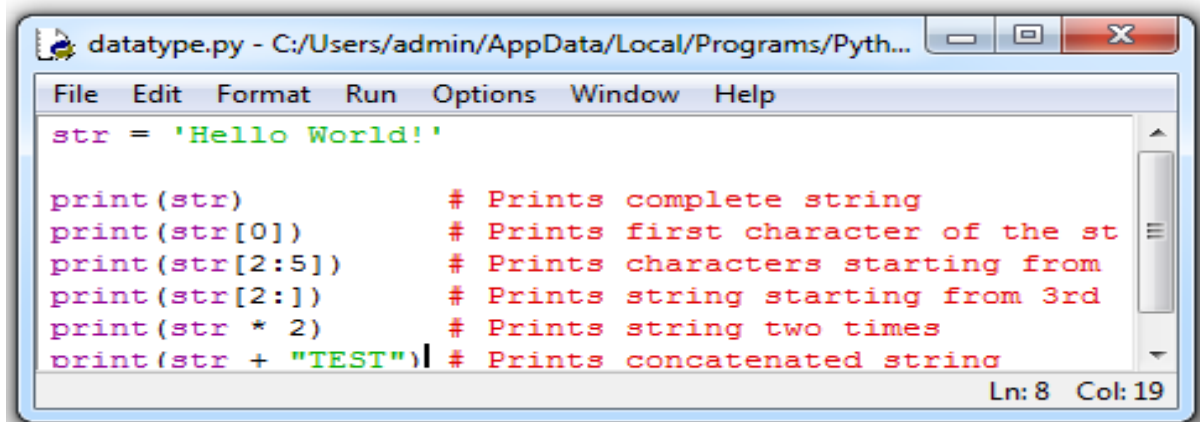
```
>>>
= RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python36/datat
ype.py =
<class 'complex'>
<class 'complex'>
<class 'complex'>
>>> |
```

2. Sequence

A sequence is an ordered collection of items, indexed by positive integers. It is Combination of mutable and non mutable data types. Three types of sequence data type available in Python are Strings, Lists & Tuples.

String: is an ordered sequence of letters/characters. They are enclosed in single quotes (') or double quotes ("). The quotes are not part of string. They only tell the computer where the string constant begins and ends. They can have

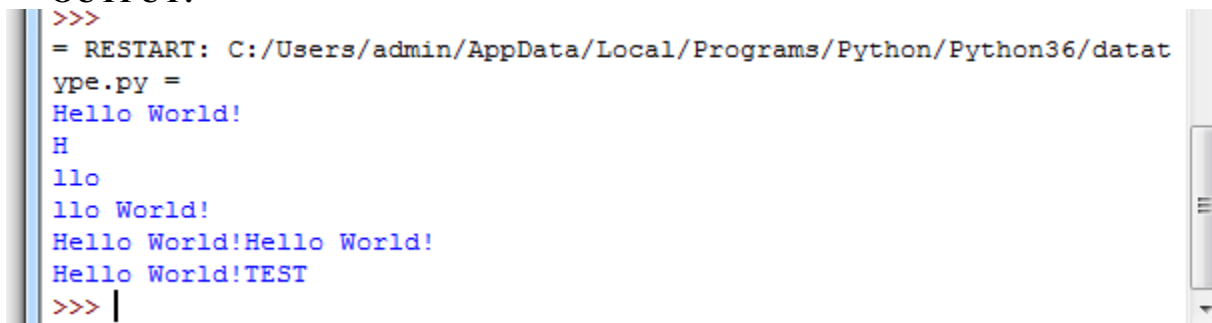
any character or sign, including space in them.



```
datatype.py - C:/Users/admin/AppData/Local/Programs/Pyth...
File Edit Format Run Options Window Help
str = 'Hello World!'

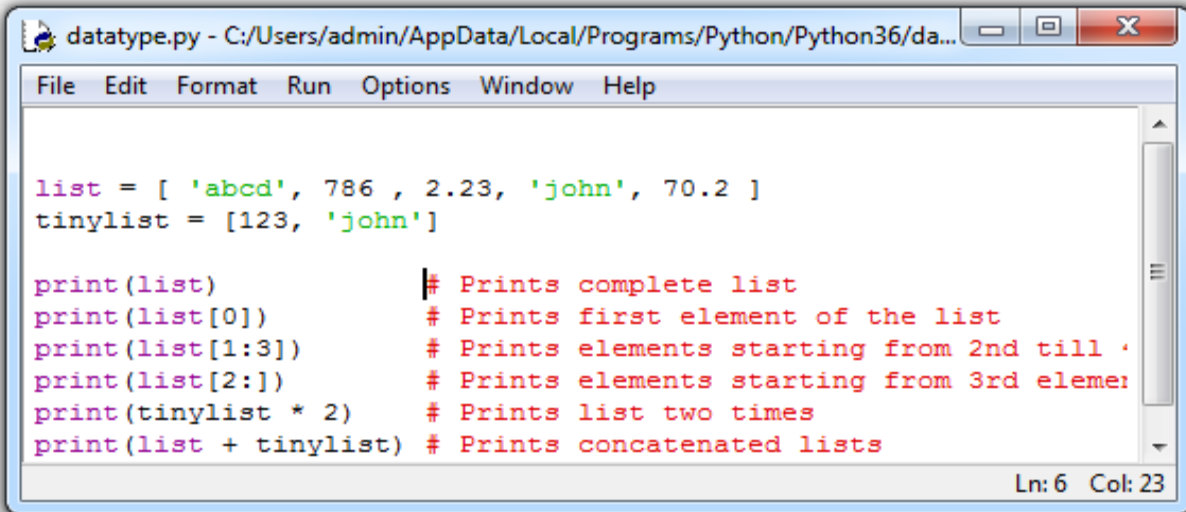
print(str)           # Prints complete string
print(str[0])        # Prints first character of the st
print(str[2:5])      # Prints characters starting from
print(str[2:])       # Prints string starting from 3rd
print(str * 2)       # Prints string two times
print(str + "TEST")  # Prints concatenated string
Ln: 8 Col: 19
```

OUTPUT:



```
>>>
= RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python36/datat
ype.py =
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
>>> |
```

List: List is also a sequence of values of any type. Values in the list are called elements / items. These are mutable and indexed/ordered. List is enclosed in square brackets.



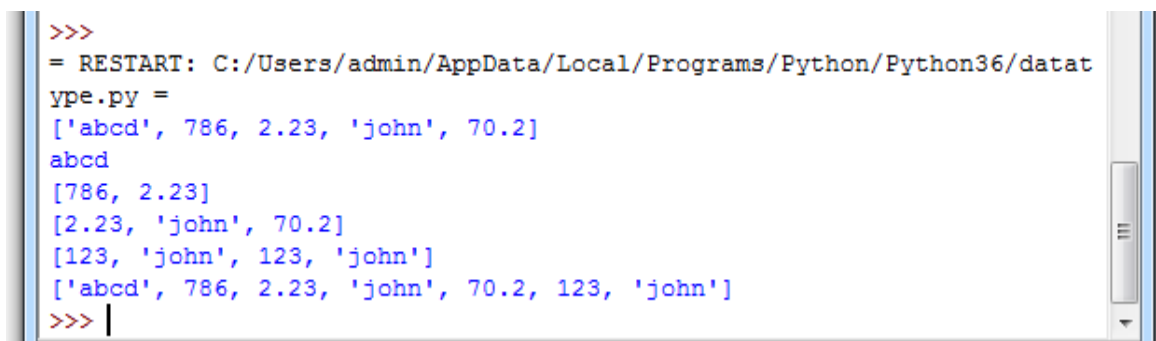
```
datatype.py - C:/Users/admin/AppData/Local/Programs/Python/Python36/da...
File Edit Format Run Options Window Help

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print(list)           # Prints complete list
print(list[0])         # Prints first element of the list
print(list[1:3])       # Prints elements starting from 2nd till '
print(list[2:])        # Prints elements starting from 3rd element
print(tinylist * 2)     # Prints list two times
print(list + tinylist) # Prints concatenated lists

Ln: 6 Col: 23
```

OUTPUT:



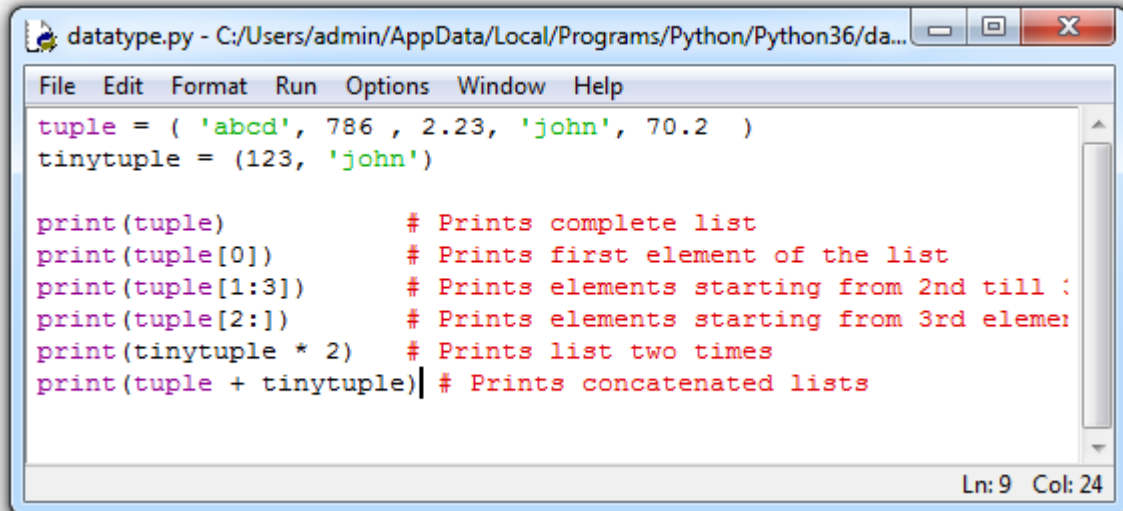
```
>>>
= RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python36/datat
ype.py =
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
>>> |
```

➤ Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed

within parentheses.

The main differences between lists and tuples are: Lists are enclosed in brackets ([]) and their elements and size can be changed, while tuples are enclosed in parentheses(()) and cannot be updated. Tuples can be thought of as read-only lists.

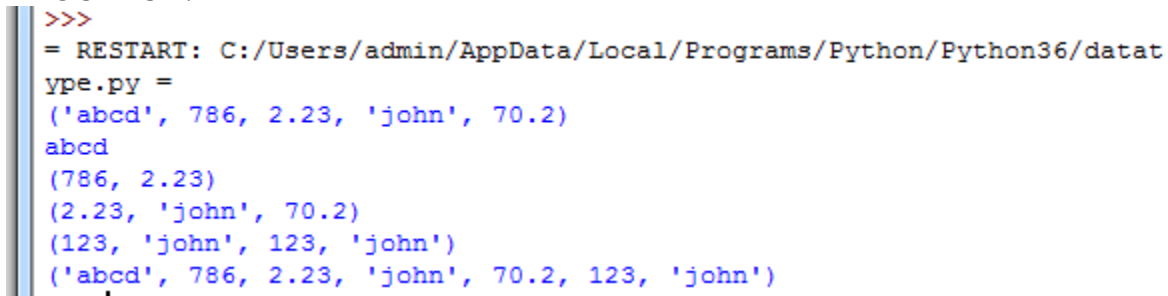


```
datatype.py - C:/Users/admin/AppData/Local/Programs/Python/Python36/da...
File Edit Format Run Options Window Help
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
tinytuple = (123, 'john')

print(tuple)           # Prints complete list
print(tuple[0])        # Prints first element of the list
print(tuple[1:3])      # Prints elements starting from 2nd till :
print(tuple[2:])       # Prints elements starting from 3rd element
print(tinytuple * 2)    # Prints list two times
print(tuple + tinytuple) # Prints concatenated lists

Ln: 9 Col: 24
```

OUTPUT:



```
>>>
= RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python36/datat
ype.py =
('abcd', 786, 2.23, 'john', 70.2)
abcd
(786, 2.23)
(2.23, 'john', 70.2)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.2, 123, 'john')
```

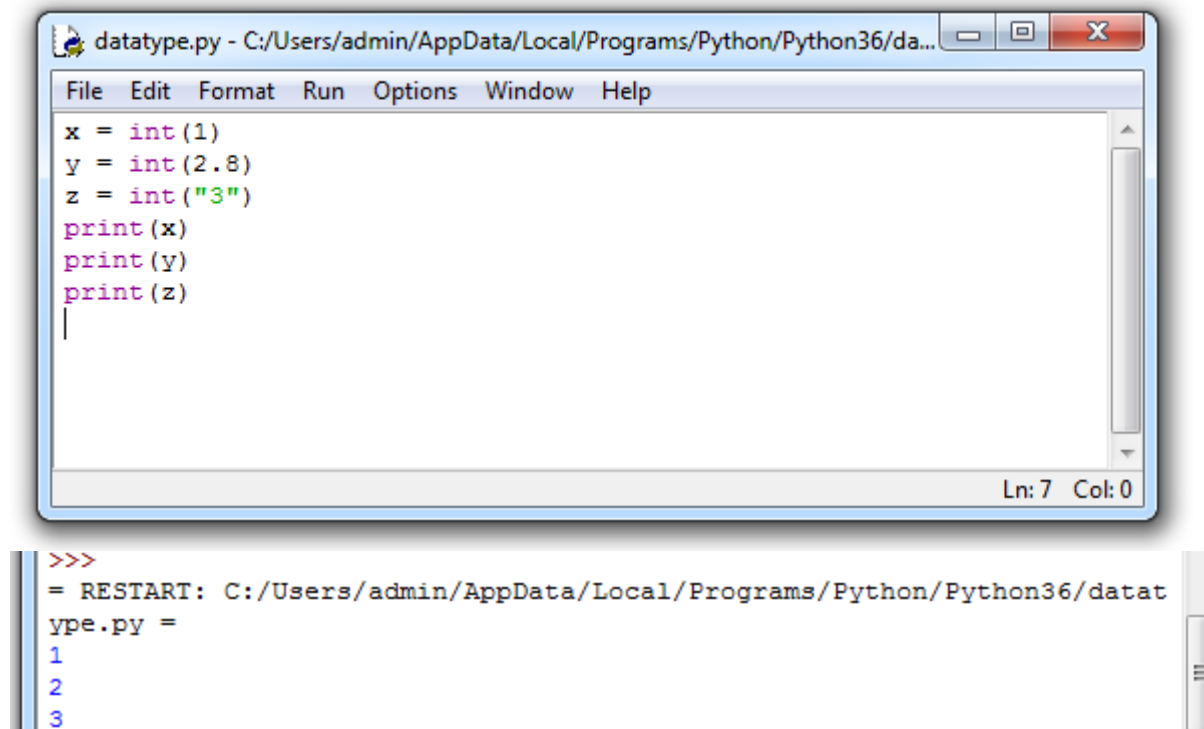
Data Type Conversion

Sometimes, you may need to perform conversions between the built-in types. To convert between types, you simply use the type name as a function.

There are several built-in functions to perform conversion from one data type to another. These functions return a new object representing the converted value.

Sr No	Function & Description
1	int(x [,base]) Converts x to an integer.
2	float(x) Converts x to a floating-point number.
3	str(x) Converts object x to a string representation.
4	eval(str) Evaluates a string and returns an object.
5	tuple(s) Converts s to a tuple.
6	list(s) Converts s to a list.
7	set(s) Converts s to a set.

EXAMPLE:



The image shows a screenshot of a Python IDE window titled 'datatype.py - C:/Users/admin/AppData/Local/Programs/Python/Python36/da...'. The window contains a menu bar (File, Edit, Format, Run, Options, Window, Help) and a code editor with the following Python code:

```
x = int(1)
y = int(2.8)
z = int("3")
print(x)
print(y)
print(z)
```

Below the code editor, the output of the script is displayed in a separate window. It shows the prompt 'RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python36/datatype.py =' followed by the output of the print statements:

```
>>>
= RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python36/datatype.py =
1
2
3
```

Operators

Mathematical/Arithmetic Operators

Symbol	Description	Example 1	Example 2
+	Addition	>>>55+45 100	>>> 'Good' + 'Morning' GoodMorning
-	Subtraction	>>>55-45 10	>>>30-80 -50
*	Multiplication	>>>55*45 2475	>>> 'Good'* 3 GoodGoodGood
/	Division	>>>17/5 3 >>>17/5.0 3.4 >>> 17.0/5 3.4	>>>28/3 9

%	Remainder/ Modulo	>>>17%5 2	>>> 23%2 1
**	Exponentiation	>>>2**3 8 >>>16**.5 4.0	>>>2**8 256
//	Integer Division	>>>7.0//2 3.0	>>>3/ / 2 1

Relational Operators

Symbol	Description	Example 1	Example 2
<	Less than	>>>7<10 True >>> 7<5 False >>> 7<10<15 True >>>7<10 and 10<15 True	>>>'Hello'< 'Goodbye' False >>>'Goodbye'< 'Hello' True
>	Greater than	>>>7>5 True >>>10<10 False	>>>'Hello'> 'Goodbye' True >>>'Goodbye'> 'Hello' False
<=	less than equal to	>>> 2<=5	>>>'Hello'<= 'Goodbye'

		True >>> 7<=4 False	False >>>'Goodbye' <= 'Hello' True
>=	greater than equal to	>>>10>=10 True >>>10>=12 False	>>>'Hello'>= 'Goodbye' True >>>'Goodbye' >= 'Hello' False
!=, <>	not equal to	>>>10!=11 True >>>10!=10 False	>>>'Hello'!= 'HELLO' True >>> 'Hello' != 'Hello' False
==	equal to	>>>10==10 True >>>10==11 False	>>>'Hello' == 'Hello' True >>>'Hello' == 'Good Bye' False

Logical Operators

Symbol	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.

Assignment Operators

Assignment Operator combines the effect of arithmetic and assignment operator

Symbol	Description	Example	Explanation
=	Assigned values from right side operands to left variable	>>>x=12* >>>y='greetings'	

(**we will use it as initial value of x for following examples*) x=10

+=	added and assign back the result to left operand	>>>x+=2	The operand/ expression/ constant written on RHS of operator is will change the value of x to 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

Input and Output

A Program needs to interact with end user to accomplish the desired task, this is done using Input-Output facility. Input means the data entered by the user (end user) of the program. While writing algorithm(s), getting input from user was represented by Take/Input. In python, we have raw-input() and input () function available for Input.

➤ raw_input()

Syntax of raw_input() is:

raw_input ([prompt])

Optional

If prompt is present, it is displayed on the monitor after which user can provide the data from keyboard. The function takes exactly what is typed from keyboard, convert it to string and then return it to the variable on LHS of „=“.

Example (in interactive mode)

```
>>>x=raw_input („Enter your name: “)
```

Enter your name: ABC

x is a variable which will get the string (ABC), typed by user during the execution of program. Typing of data for the raw_input function is terminated by „enter“ key.

We can use raw_input() to enter numeric data also. In that case we typecast, i.e., change

thedata type using function, the string data accepted from user to appropriate Numeric type.

Example

```
y=int(raw_input(“enter your roll no”))
```

enter your roll no. 5

will convert the accepted string i.e. 5 to integer before assigning it to „y“.

➤ input()

Syntax for input() is:

Input ([prompt])

Optional

If prompt is present, it is displayed on monitor, after which the user can provide data from keyboard. Input takes whatever is typed from the keyboard and evaluates it. As the input provided is evaluated, it expects valid python expression. If the input provided is not correct then either syntax error or exception is raised by python.

Example

```
x= input (“enter data:”)
```

```
>>>Enter data:2
```

Will supply 2 to x

Output is what program produces. In algorithm, it was represented by print. For output in Python we use print. We have already seen its usage in previous examples. Let’s learn more about it.

Print Statement

Syntax:

print("expression/constant/variable")

Print evaluates the expression before printing it on the monitor. Print statement outputs an entire (complete) line and then goes to next line for subsequent output (s). To print more than one item on a single line, comma (,) may be used.

Example

```
>>> print ("Hello")
```

Hello

```
>>> print (5.5)
```

5.5

```
>>> print (4+6)
```

10

Try this on the computer and evaluate the output generated

```
>>>print (3.14159* 7**2)
```

```
>>>print ("I'm",)
```

```
>>>print ("class XI student")
```

```
>>>print ("I'm ", 16, "years old")
```

Comments

As the program gets bigger, it becomes difficult to read it, and to make out what it is doing by just looking at it. So it is good to add notes to the code, while writing it. These notes are known as comments. In Python, comment start with "#" symbol. Anything written after # in a line is ignored by interpreter, i.e. it will not have any effect on the program.

A comment can appear on a line by itself or they can also be at the end of line.

Example

```
# Calculating area of a square
```

```
>>>area = side **2
```

or

```
>>>area= side**2 #calculating area of a square
```

For adding multi-line comment in a program, we can:

i) Place "#" in front of each line, or

ii) **Use triple quoted string**. They will only work as comment, when they are not being used as docstring. (A docstring is the first thing in a class/function /module, and will be taken up in details when we study functions).

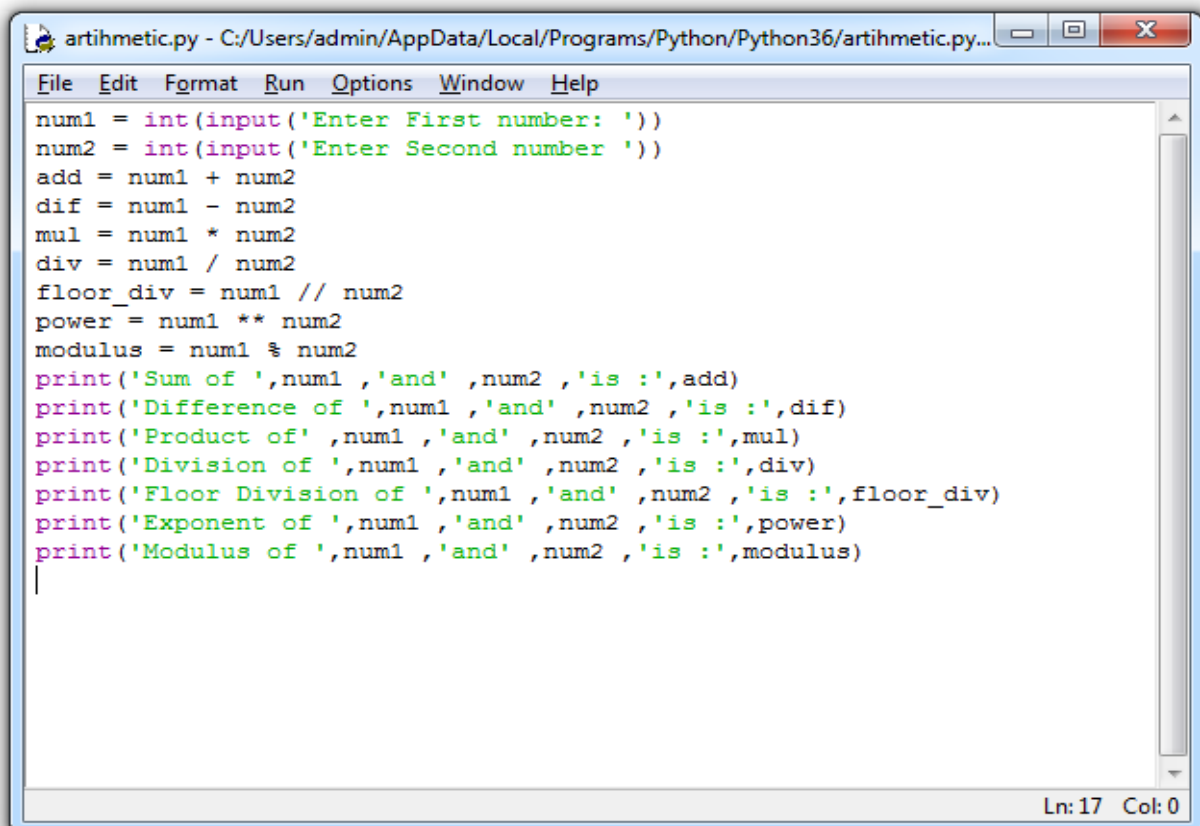
The comment line "#calculating area of a rectangle" can also be written as following using triple quote:

1. `""" Calculating area of a rectangle """`

2. `""" Calculating area
of a rectangle """`

EXAMPLE:

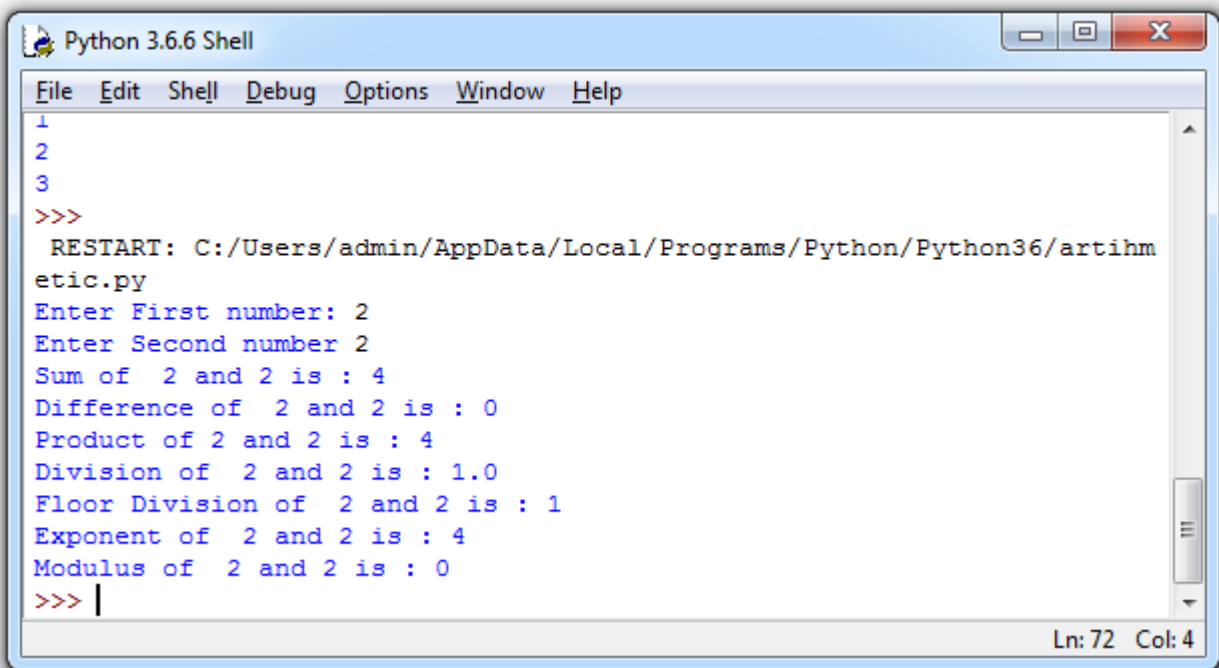
Operator example



artihmetic.py - C:/Users/admin/AppData/Local/Programs/Python/Python36/artihmetic.py...

```
File Edit Format Run Options Window Help
num1 = int(input('Enter First number: '))
num2 = int(input('Enter Second number '))
add = num1 + num2
dif = num1 - num2
mul = num1 * num2
div = num1 / num2
floor_div = num1 // num2
power = num1 ** num2
modulus = num1 % num2
print('Sum of ',num1 , 'and' ,num2 , 'is :',add)
print('Difference of ',num1 , 'and' ,num2 , 'is :',dif)
print('Product of' ,num1 , 'and' ,num2 , 'is :',mul)
print('Division of ',num1 , 'and' ,num2 , 'is :',div)
print('Floor Division of ',num1 , 'and' ,num2 , 'is :',floor_div)
print('Exponent of ',num1 , 'and' ,num2 , 'is :',power)
print('Modulus of ',num1 , 'and' ,num2 , 'is :',modulus)
|
```

Ln: 17 Col: 0



Python 3.6.6 Shell

```
File Edit Shell Debug Options Window Help
1
2
3
>>>
RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python36/artihm
etic.py
Enter First number: 2
Enter Second number 2
Sum of 2 and 2 is : 4
Difference of 2 and 2 is : 0
Product of 2 and 2 is : 4
Division of 2 and 2 is : 1.0
Floor Division of 2 and 2 is : 1
Exponent of 2 and 2 is : 4
Modulus of 2 and 2 is : 0
>>> |
```

Ln: 72 Col: 4

DECISION MAKING / CONDITIONAL STATEMENT

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.

Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.

➤ if statement

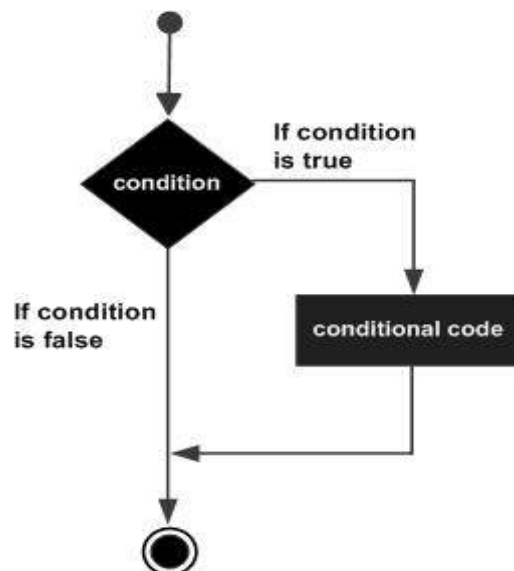
It is similar to that of other languages. The if statement contains a logical expression using which data is compared and a decision is made based on the result of the comparison.

Syntax

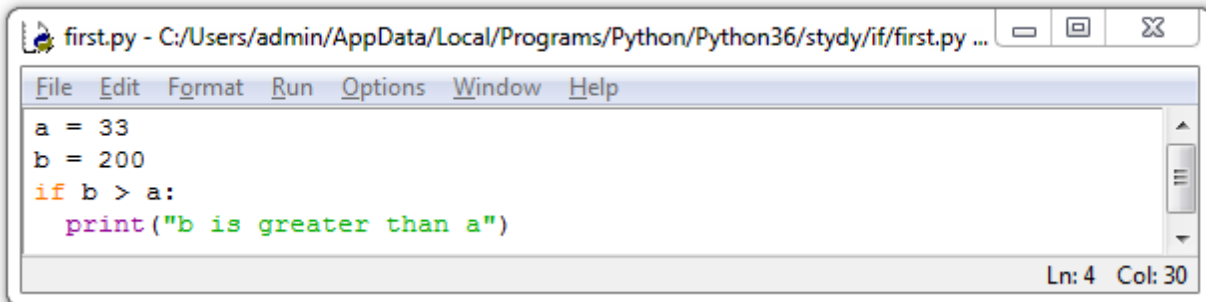
```
if expression:  
    statement(s)
```

If the boolean expression evaluates to TRUE, then the block of statement(s) inside the if statement is executed. If boolean expression evaluates to FALSE, then the first set of code after the end of the if statement(s) is executed.

Flow Diagram

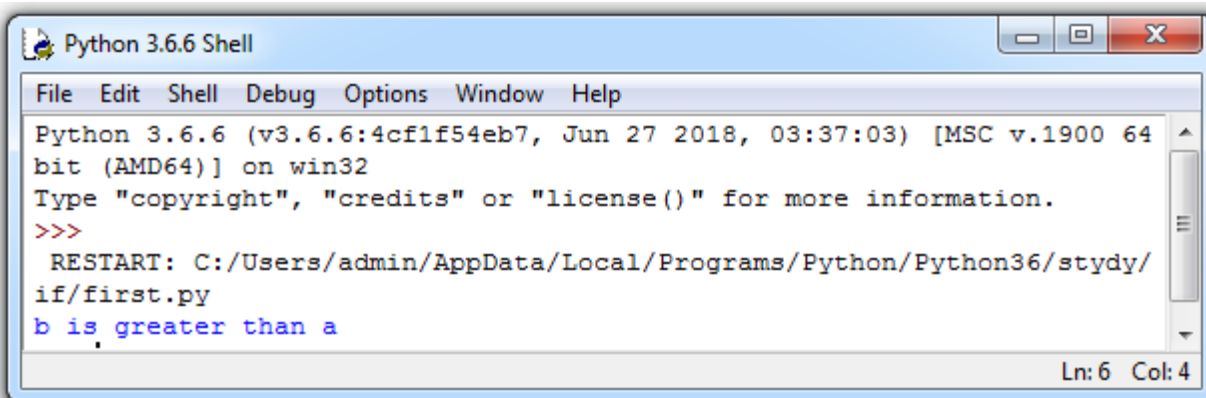


Example



```
first.py - C:/Users/admin/AppData/Local/Programs/Python/Python36/stydy/if/first.py ...  
File Edit Format Run Options Window Help  
a = 33  
b = 200  
if b > a:  
    print("b is greater than a")  
Ln: 4 Col: 30
```

OUTPUT



```
Python 3.6.6 Shell  
File Edit Shell Debug Options Window Help  
Python 3.6.6 (v3.6.6:4cf1f54eb7, Jun 27 2018, 03:37:03) [MSC v.1900 64  
bit (AMD64)] on win32  
Type "copyright", "credits" or "license()" for more information.  
>>>  
RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python36/stydy/  
if/first.py  
b is greater than a  
Ln: 6 Col: 4
```

➤ **if...else statement**

An **else** statement can be combined with an **if** statement. An **else** statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.

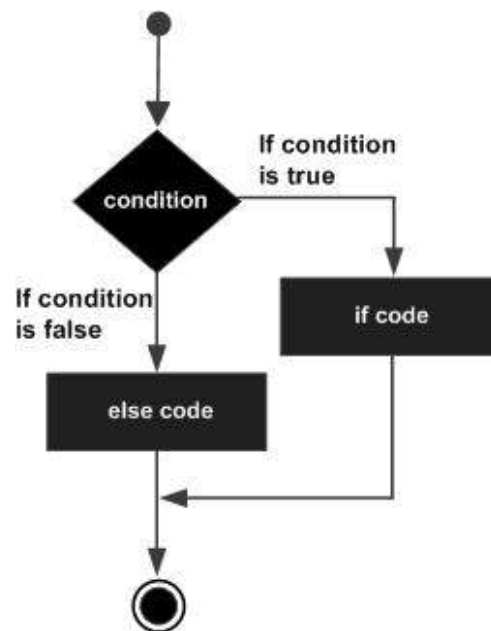
The **else** statement is an optional statement and there could be at most only one **else** statement following **if**.

Syntax

The syntax of the if...else statement is –

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

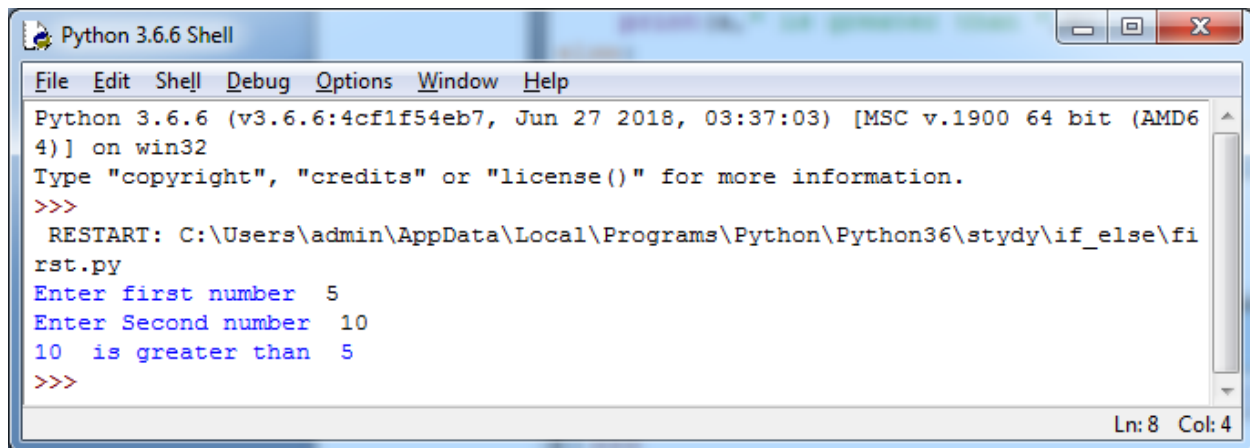
Flow Diagram



Example

```
first.py - C:\Users\admin\AppData\Local\Programs\Python\Python36\stydy\if_else\first.py (3.6.6)  
File Edit Format Run Options Window Help  
a = int(input("Enter first number "))  
b = int(input("Enter Second number "))  
if a > b :  
    print(a, " is greater than ", b)  
else:  
    print(b, " is greater than ", a)  
Ln: 6 Col: 32
```

OUTPUT:

A screenshot of a Python 3.6.6 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following output:

```
Python 3.6.6 (v3.6.6:4cf1f54eb7, Jun 27 2018, 03:37:03) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
  RESTART: C:\Users\admin\AppData\Local\Programs\Python\Python36\stydy\if_else\first.py
Enter first number 5
Enter Second number 10
10 is greater than 5
>>>
```

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

➤ **if...elif...else**

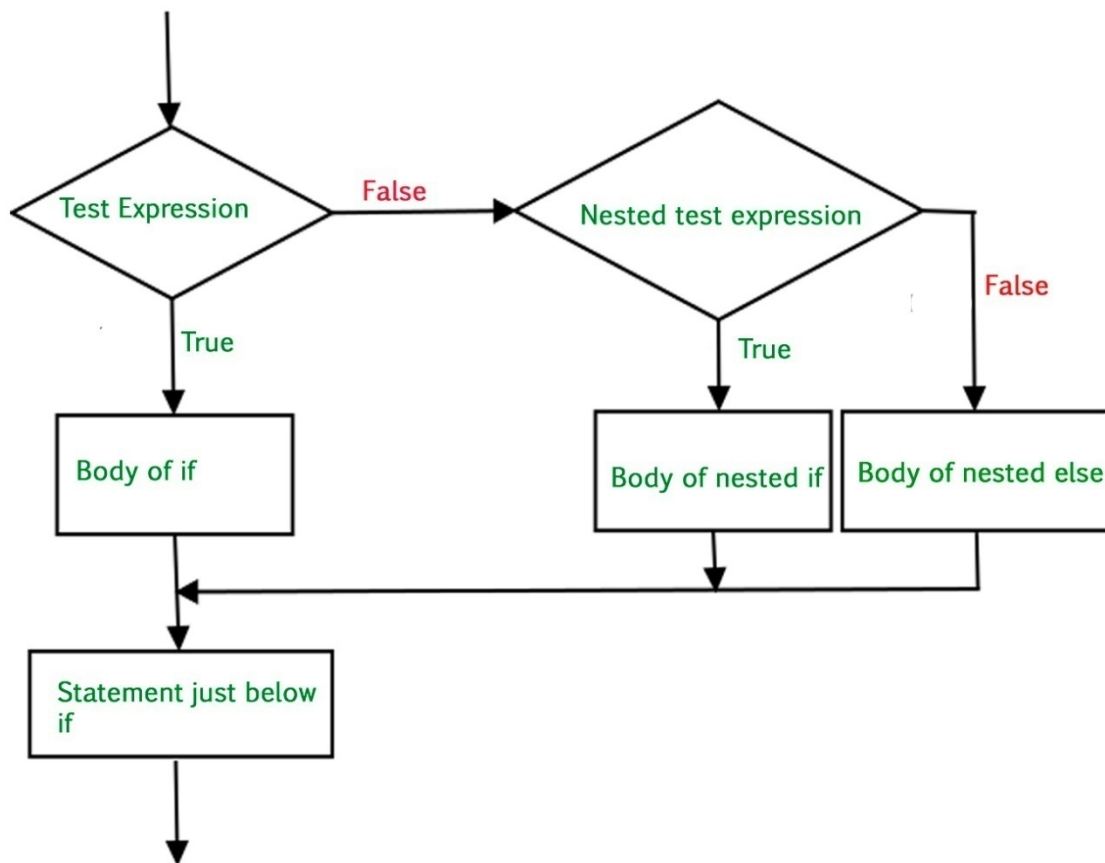
There may be a situation when you want to check for another condition after a condition resolves to true. In such a situation, you can use the nested **if** construct.

In a nested **if** construct, you can have an **if...elif...else** construct inside another **if...elif...else** construct.

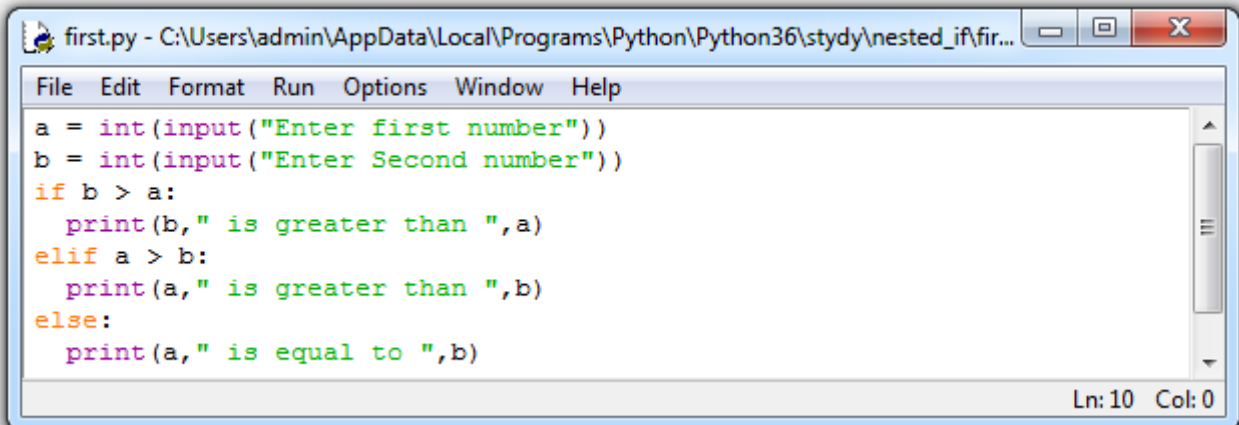
Syntax

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

```
else:  
statement(s)  
elif expression4:  
statement(s)  
else:  
statement(s)
```



EXAMPLE :

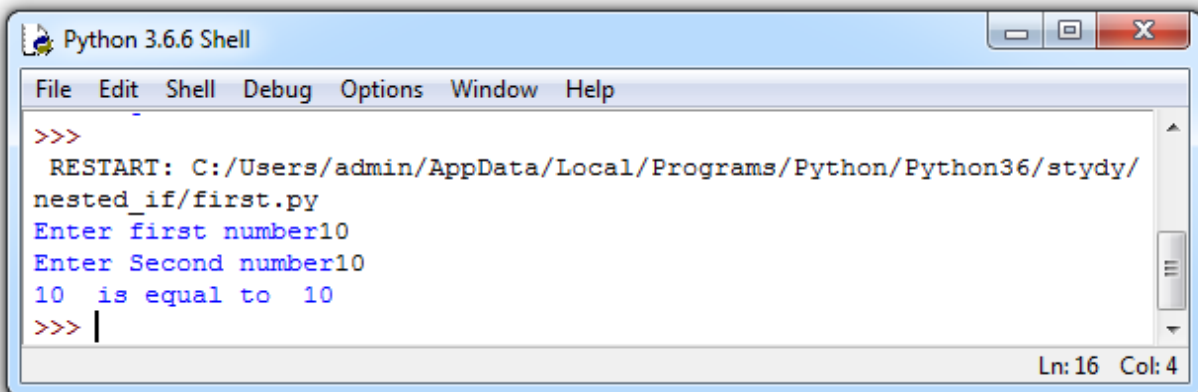


A screenshot of a Python IDE window titled "first.py - C:\Users\admin\AppData\Local\Programs\Python\Python36\study\nested_if\fir...". The window contains a Python script with a nested if-else statement. The code is as follows:

```
a = int(input("Enter first number"))
b = int(input("Enter Second number"))
if b > a:
    print(b, " is greater than ",a)
elif a > b:
    print(a, " is greater than ",b)
else:
    print(a, " is equal to ",b)
```

The status bar at the bottom right indicates "Ln: 10 Col: 0".

OUTPUT:



A screenshot of a Python 3.6.6 Shell window titled "Python 3.6.6 Shell". The window shows the execution of the script from the previous image. The output is as follows:

```
>>>
RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python36/study/
nested_if/first.py
Enter first number10
Enter Second number10
10 is equal to 10
>>> |
```

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

LOOPS

if you have code that you want to do many times concurrently, you would use a loop instead of writing the same code block many times. A loop is defined by its header, which defines when to stop executing the code block. Frequently, loops employ a loop counter to keep track of how many times the code block has executed.

➤ while loop

A **while** loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

Syntax

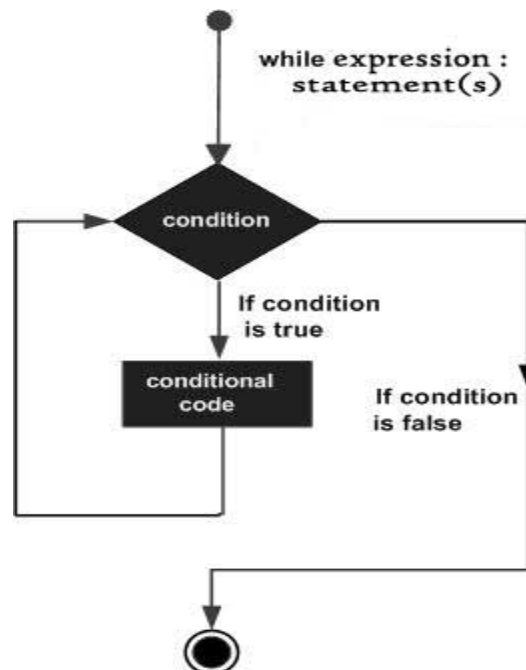
```
while expression:  
statement(s)
```

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true.

When the condition becomes false, program control passes to the line immediately following the loop.

In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

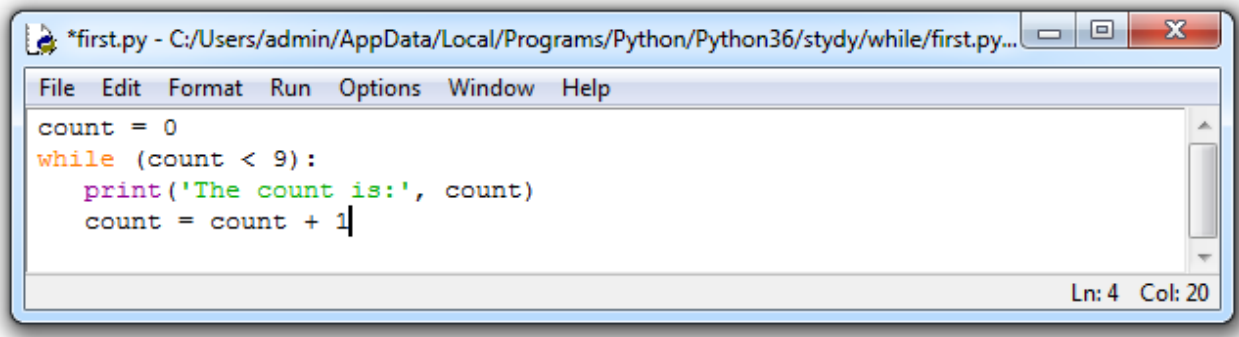
Flow Diagram



Here, key point of the while loop is that the loop might not ever run. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

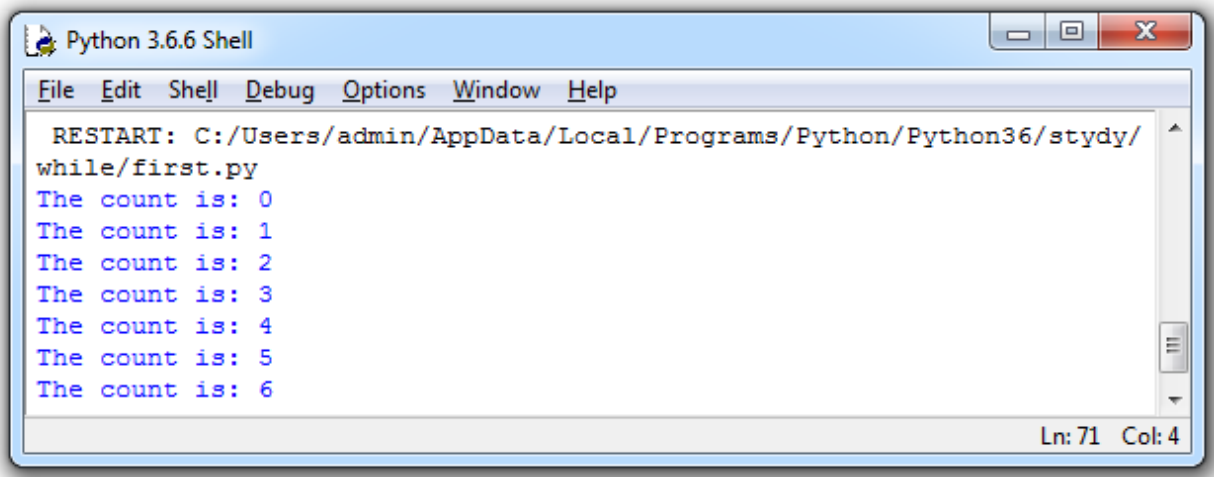
EXAMPLE:

To print first 9 numbers



```
*first.py - C:/Users/admin/AppData/Local/Programs/Python/Python36/stydy/while/first.py...
File Edit Format Run Options Window Help
count = 0
while (count < 9):
    print('The count is:', count)
    count = count + 1
Ln: 4 Col: 20
```

OUTPUT:



```
Python 3.6.6 Shell
File Edit Shell Debug Options Window Help
RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python36/stydy/
while/first.py
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
Ln: 71 Col: 4
```

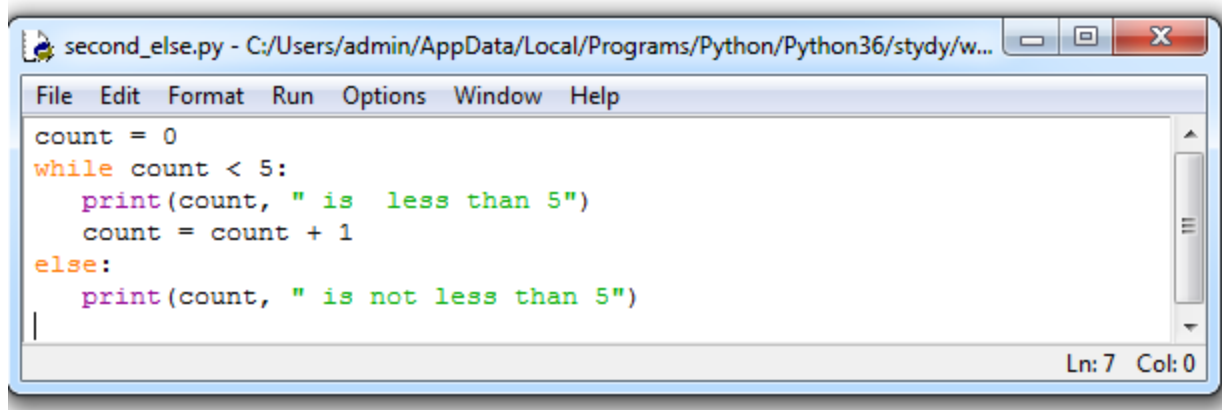
Using else Statement with Loops

Python supports to have an **else** statement associated with a loop statement.

- If the **else** statement is used with a **while** loop, the **else** statement is executed when the condition becomes false.

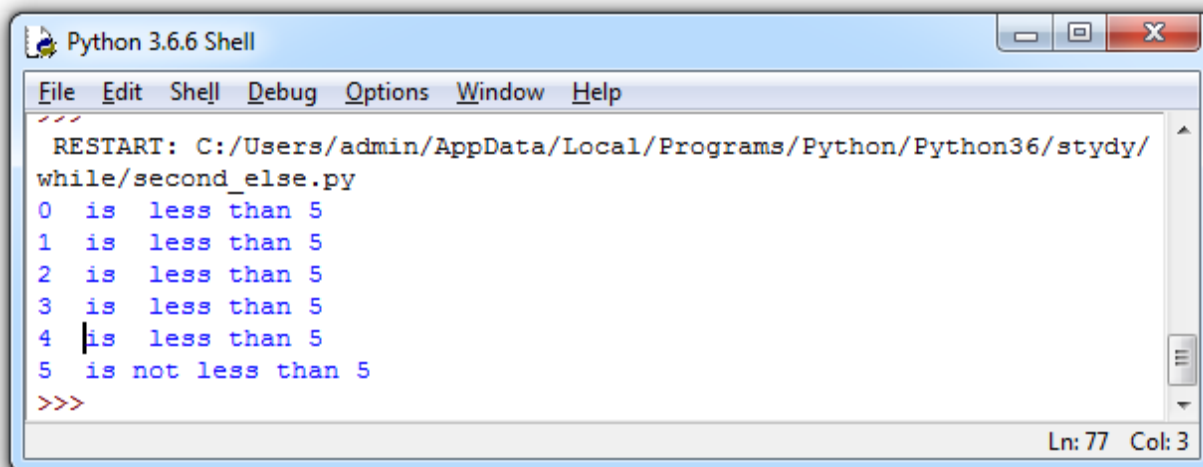
The following example illustrates the combination of an else statement with a while statement that prints a number as long as it is less than 5, otherwise else statement gets executed.

EXAMPLE:



```
second_else.py - C:/Users/admin/AppData/Local/Programs/Python/Python36/stydy/w...
File Edit Format Run Options Window Help
count = 0
while count < 5:
    print(count, " is less than 5")
    count = count + 1
else:
    print(count, " is not less than 5")
|
Ln: 7 Col: 0
```

OUTPUT



```
Python 3.6.6 Shell
File Edit Shell Debug Options Window Help
RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python36/stydy/
while/second_else.py
0 is less than 5
1 is less than 5
2 is less than 5
3 is less than 5
4 is less than 5
5 is not less than 5
>>>
Ln: 77 Col: 3
```

➤ for loop

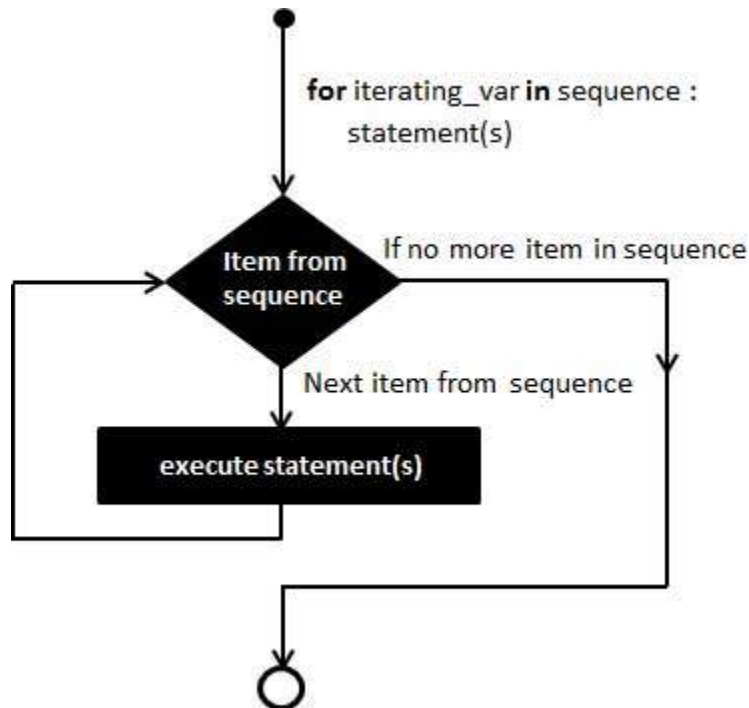
It has the ability to iterate over the items of any sequence, such as a list or a string.

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.

Flow Diagram



Example

Print each fruit in a fruit list:

```
first.py - C:/Users/admin/AppData/Local/Programs/Python/Python36/stydy/for/first.py...
File Edit Format Run Options Window Help
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
```

The screenshot shows a Python IDE window titled "first.py - C:/Users/admin/AppData/Local/Programs/Python/Python36/stydy/for/first.py...". The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code editor contains the following Python code:
`fruits = ["apple", "banana", "cherry"]`
`for x in fruits:`
 `print(x)`
The status bar at the bottom right indicates "Ln: 4 Col: 0".

OUTPUT

```
>>>
RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python36/stydy/
for/first.py
apple
banana
cherry
>>> |
```

Ln: 89 Col: 4

range() Function

To loop through a set of code a specified number of times, we can use the range() function. The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number. The statement introduces a function range (), its syntax is range(start, stop, [step]) # step is optional

Example

Using the range() function:

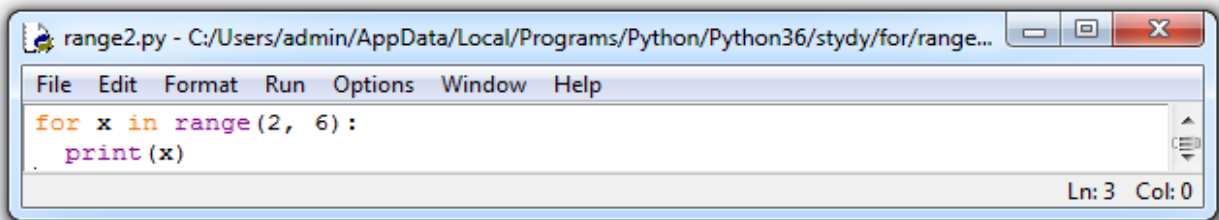
```
range.py - C:/Users/admin/AppData/Local/Programs/Python/Python36/stydy/for/ra...
File Edit Format Run Options Window Help
for x in range(6):
    print(x)
Ln: 3 Col: 0
```

OUTPUT:

```
Python 3.6.6 Shell
File Edit Shell Debug Options Window Help
RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python36/stydy/
for/range.py
0
1
2
3
4
5
>>> |
Ln: 97 Col: 4
```

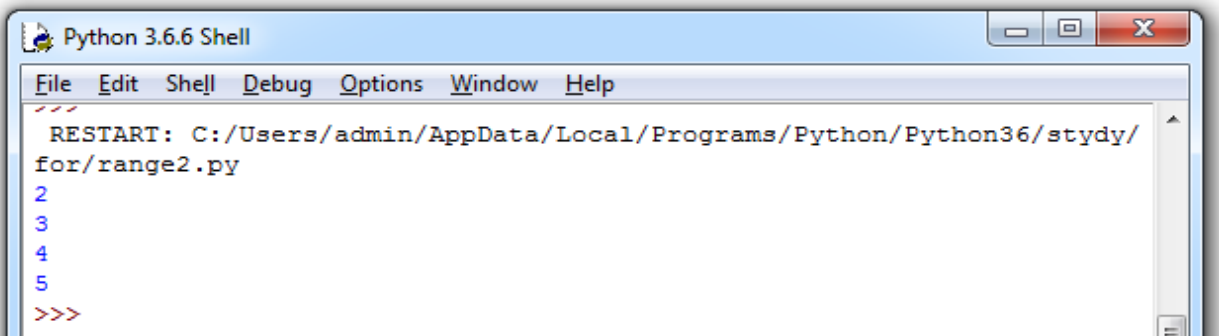
Example

Using the start parameter:



```
range2.py - C:/Users/admin/AppData/Local/Programs/Python/Python36/stydy/for/range...
File Edit Format Run Options Window Help
for x in range(2, 6):
    print(x)
Ln: 3 Col: 0
```

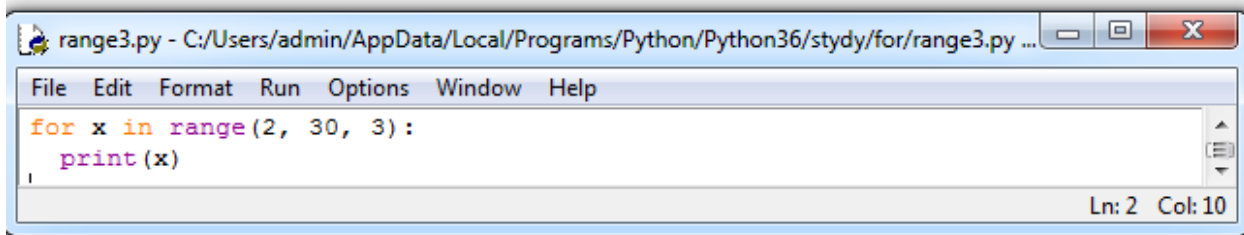
OUTPUT:



```
Python 3.6.6 Shell
File Edit Shell Debug Options Window Help
RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python36/stydy/
for/range2.py
2
3
4
5
>>>
```

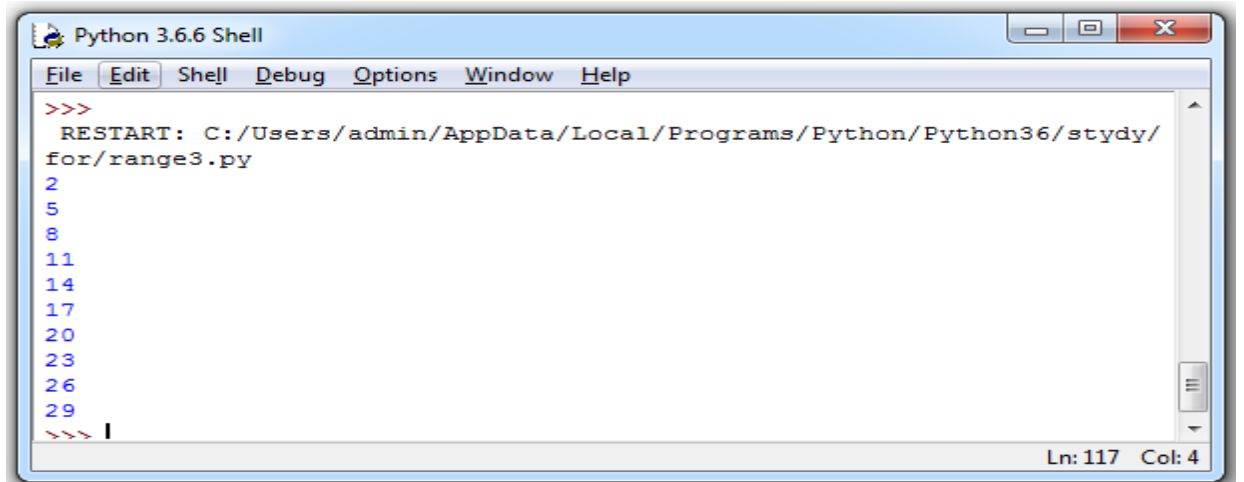
Example

Increment the sequence with 3 (default is 1):



```
range3.py - C:/Users/admin/AppData/Local/Programs/Python/Python36/stydy/for/range3.py ...
File Edit Format Run Options Window Help
for x in range(2, 30, 3):
    print(x)
Ln: 2 Col: 10
```

OUTPUT:



```
Python 3.6.6 Shell
File Edit Shell Debug Options Window Help
>>>
RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python36/stydy/
for/range3.py
2
5
8
11
14
17
20
23
26
29
>>> |
Ln: 117 Col: 4
```