```
Text enclosed inside \texttt{verbatim} environment
is printed directly
and all \LaTeX{} commands are ignored.
```

Just as in the example at the introduction, all text is printed keeping line breaks and white spaces. There's a starred version of this command whose output is slightly different.

```
Text␣enclosed␣inside␣\texttt{verbatim}␣environment
is␣printed␣directly
and␣all␣\LaTeX{}␣commands␣are␣ignored.
```

===

To use the lstlisting environment you have to add the following line to the preamble of your document:

\usepackage{listings}

```
name = input(’What is your name?\n’)
print (’Hi, %s.’ % name)

```

In this example, the output ignores all LATEX commands and the text is printed keeping all the line breaks and white spaces typed. Let's see a second example:

```
name = input(’What is your name?\n’)
print (’Hi, %s.’ % name)

```

The additional parameter inside brackets [language=Python] enables code highlighting for this particular programming language (Python), special words are in boldface font and comments are italicized.

Below is an example of adding a C++ snippet

```
#include <iostream>
using namespace std;

// main() is where program execution begins.
int main() {
  cout << "Hello World"; // prints Hello World
  return 0;
}

```

==

Code is usually stored in a source file, therefore a command that automatically pulls code from a file becomes very handy.

The next code will be directly imported from a file

```
#program to find area of circle in Python using math file
import math
r = float(input("Enter the radius of the circle: "))
area = math.pi* r * r
print("%.2f" %area)
```

As you see, the code colouring and styling greatly improves readability.
There are essentially two commands that generate the style for this example:

\lstdefinestyle{mystyle}{...}
Defines a new code listing style called ”mystyle”. Inside the second pair of braces the options that define this style are passed; see the reference guide for a full description of these and some other parameters.
\lstset{style=mystyle}
Enables the style ”mystyle”. This command can be used within your document to switch to a different style if needed.

===================================

```
    name = input('What is your name?\n')
    print ('Hi, %s.' % name)

```

Listing 1: Python Program Example

Adding the comma-separated parameter caption=Python example inside the brackets, enables the caption. This caption can be later used in the list of Listings.

\lstlistoflistings

Listings

===================================

Options to customize code listing styles
backgroundcolor - colour for the background. External color or xcolor package needed.
commentstyle - style of comments in source language.
basicstyle - font size/family/etc. for source (e.g. basicstyle=)
keywordstyle - style of keywords in source language (e.g. keywordstyle=\color{red})

```
numberstyle - style used for line-numbers
numbersep - distance of line-numbers from the code
stringstyle - style of strings in source language
showspaces - emphasize spaces in code (true/false)
showstringspaces - emphasize spaces in strings (true/false)
showtabs - emphasize tabulators in code (true/false)
numbers - position of line numbers (left/right/none, i.e.  no line numbers)
prebreak - displaying mark on the end of breaking line (e.g.  prebreak=\raisebox{0ex}[0ex][0ex]{\ensur
captionpos - position of caption (t/b)
frame - showing frame outside code (none/leftline/topline/bottomline/lines/single/shadowbox)
breakwhitespace - sets if automatic breaks should only happen at whitespaces
breaklines - automatic line-breaking keepspaces - keep spaces in the code,
useful for indetation tabsize - default tabsize rulecolor - Specify the colour
of the frame-box
   ==========================================
```

Example Python Code

```python
import qiskit as q
from qiskit import Aer,execute
from qiskit import IBMQ
from qiskit.tools.visualization import plot_histogram,
plot_bloch_multivector
from qiskit.tools.monitor import job_monitor
import matplotlib
statevector=q.Aer.get_backend("statevector_simulator")#
statevector simulator
qasm_sim=q.Aer.get_backend("qasm_simulator")#Quantum simulator
def do_jobs(circuit):
result=execute(circuit,backend=statevector).result()#
statevectors of statevec
statevec = result.get_statevector()
n_qubits=circuit.num_qubits#get total no of qubits in circuit
circuit.measure([i for i in range(n_qubits)],[i for i in range(
n_qubits)])# measure qubits and store in classical bits

result2=execute(circuit,backend=qasm_sim).result()#qasm
simulator
counts = result2.get_counts()
return statevec,counts

circuit=q.QuantumCircuit(2,2)#qubit=2 and classical=2
circuit.h(0)#H gate on 1st qubit
circuit.x(1)#X gate on 2nd qubit
circuit.h(1)#X gate on 2nd qubit
statevec,counts=do_jobs(circuit)
circuit.draw(output="mpl",filename='superpositionckt.png')
plot_bloch_multivector(statevec).show()#plot blochsphere
plot_histogram(counts).show()#plot histogram

```

==