The most recent output value is automatically stored by the interpreter in a special variable with the name "\_". So we can print the output from the recent example again by typing an underscore after the prompt:

>>> \_

11

>>>

The underscore can be used in other expressions like any other variable:

>>> \_ \* 3

33

>>>

**Python Internals**

Most probably you will have read somewhere that the Python language is an interpreted programming or script language. The truth is: Python is both an interpreted and a compiled language. But calling Python a compiled language would be misleading. People would assume that the compiler translates the Python code into machine language. Python code is translated into intermediate code, which has to be executed by a virtual machine, known as the PVM, the Python virtual machine. This is a similar approach to Java. There is even a way of translating Python programs into Java byte code for the Java Virtual Machine (JVM). This can be achieved with Jython.   
  
The question is, do I have to compile my Python scripts to make them faster or how can I compile them? The answer is easy: Normally, you don't need to do anything and you shouldn't bother, because "Python" is doing the thinking for you.   
  
For whatever reason you want to compile a python program manually? No problem. It can be done with the module *py\_compile*, either using the interpreter shell,

>>> import py\_compile

>>> py\_compile.compile('easy\_to\_write.py')

>>>

or using the following command at the shell prompt,

python -m py\_compile easy\_to\_write.py

Either way, a file named "easy\_to\_write.pyc" will be created.   
You can also automatically compile all files in a directory using the compileall module. You can do it from the shell prompt by running compileall.py and providing the path of the directory containing the Python files to compile:

monty@python:~/python$ python -m compileall .

Listing . ...

or using the following command at the shell prompt,

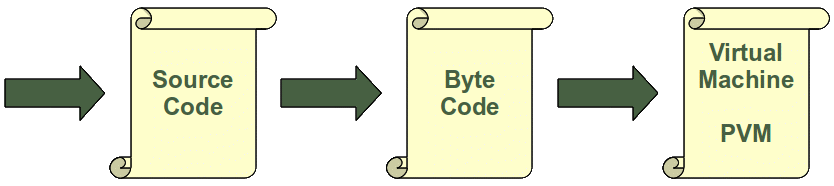
python -m py\_compile easy\_to\_write.py

Either way, a file named "easy\_to\_write.pyc" will be created.   
You can also automatically compile all files in a directory using the compileall module. You can do it from the shell prompt by running compileall.py and providing the path of the directory containing the Python files to compile:

monty@python:~/python$ python -m compileall .

Listing . ...

But as we have said, you don't have to bother about compiling Python code. The compilation is hidden from the user. Some newbies to Python wonder sometimes where these ominous files with the .pyc suffix might come from. If Python has write-access for the directory where the Python program resides, it will store the compiled byte code in a file that ends with a .pyc suffic. If Python has no write access, the program will work anyway. The byte code will be produced but discarded when the program exits.  
Whenever a Python program is called, Python will check, if there exists a compiled version with the .pyc suffix. This file has to be newer than the file with the .py suffix. If such a file exists, Python will load the byte code, which will speed up the start up time of the script. If there exists no byte code version, Python will create the byte code before it starts the execution of the program. Execution of a Python program means execution of the byte code on the Python Virtual Machine (PVM). 



Every time a Python script is executed, byte code is created. If a Python script is imported as a module, the byte code will be stored in the corresponding .pyc file.   
So the following will not create a byte code file:

monty@python:~$ cd python

monty@python:~/python$ python easy\_to\_write.py

It's easy to write a Python script!

monty@python:~/python$

The import in the following session will create a byte code file with the name "easy\_to\_write.pyc":

monty@python:~/python$ ls

easy\_to\_write.py

monty@python:~/python$ python

Python 2.6.5 (r265:79063, Apr 16 2010, 13:57:41)

[GCC 4.4.3] on linux2

Type "help", "copyright", "credits" or "license" for more information.

>>> import easy\_to\_write

It's easy to write a Python script!

>>>

monty@python:~/python$ ls

easy\_to\_write.py easy\_to\_write.pyc

monty@python:~/python$

### Compiler

Definition: A compiler is a computer program that transforms (translates) source code of a programming language (the source language) into another computer language (the target language). In most cases compilers are used to transform source code into executable program, i.e. they translate code from high-level programming languages into low (or lower) level languages, mostly assembly ore machine code.

### Interpreter

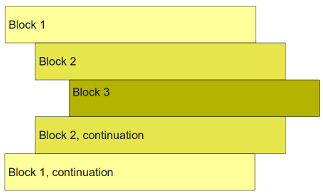
Definition: An interpreter is a computer program that executes instructions written in a programming language. It can either

* execute the source code directly or
* translates the source code in a first step into a more efficient representation and executes this code

#### Help

help("execfile")

## Structuring with Indentation



### Data Types and Variables

Not only the value of a variable may change during program execution but the type as well. You can assign an integer value to a variable, use it as an integer for a while and then assign a string to the variable.

Variables and identifiers are very often mistaken as synonyms. In simple terms: The name of a variable is an identifier, but a variable is "more than a name". A variable has a name, in most cases a type, a scope, and above all a value. Besides this, an identifier is not only used for variables. An identifier can denote various entities like variables, types, labels, subroutines or functions, packages and so on.

A valid identifier is a non-empty sequence of characters of any length with:

* The start character can be the underscore "\_" or a capital or lower case letter.
* The letters following the start character can be anything which is permitted as a start character plus the digits.
* Just a warning for Windows-spoilt users: Identifiers are case-sensitive!
* Python keywords are not allowed as identifier names!

No identifier can have the same name as one of the Python keywords:   
*and, as, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, with, yield*