

Introduction

Python overview

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- **Python is Interpreted:** Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive:** You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented:** Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language:** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

History of Python

- Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.
- Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.
- Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).
- Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Python Features

- **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.
- **A broad standard library:** Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **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.
- **Scalable:** Python provides a better structure and support for large programs than shell scripting.

Simple Python Program

- `print ("Hello, Python!")`
- `C=10`
- `D=12`
- `print(C+D)`

who uses Python

On-line games



Web services



Applications



Science



Instrument control



Embedded systems



en.wikipedia.org/wiki/List_of_Python_software

who uses Python

So who uses Python and what for?

Python is used for everything! For example:

“massively multiplayer online role-playing games” like Eve Online, science fiction’s answer to World of Warcraft,

web applications written in a framework built on Python called “Django”,

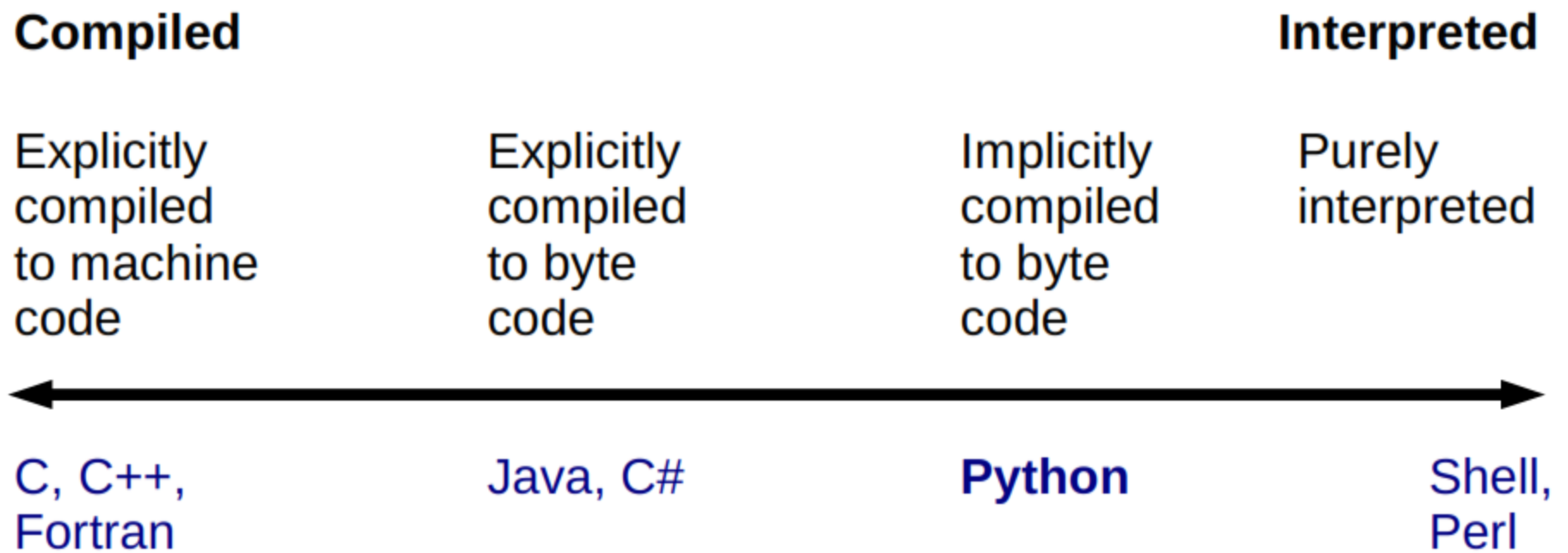
desktop applications like Blender, the 3-d animation suite which makes considerable use of Python scripts,

the Scientific Python libraries (“SciPy”),

instrument control and

embedded systems.

What sort of language is Python?



Byte code compilation

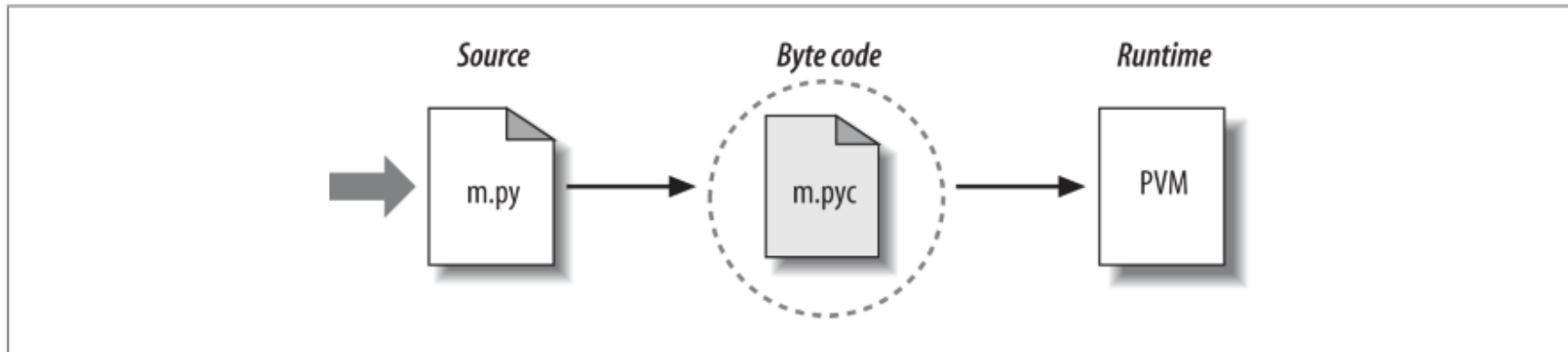
Internally, and almost completely hidden from you, when you execute a program Python first compiles your *source code (the statements in your file)* into a format known as *byte code*. *Compilation is simply a translation step, and byte code is a lower-level, platform-independent representation of your source code*. Roughly, Python translates each of your source statements into a group of byte code instructions by decomposing them into individual steps. This byte code translation is performed to speed execution. Byte code can be run much more quickly than the original source code statements in your text file.

You'll notice that the prior paragraph said that this is *almost completely hidden from you*. If the Python process has write access on your machine, it will store the byte code of your programs in files that end with a *.pyc extension* (*“.pyc” means compiled “.py” source*). Prior to Python 3.2, you will see these files show up on your computer after you've run a few programs alongside the corresponding source code files, that is, in the *same directories*. *For instance, you'll notice a script.pyc after importing a script.py.*

The Python Virtual Machine (PVM)

Once your program has been compiled to byte code (or the byte code has been loaded from existing *.pyc files*), *it is shipped off for execution to something generally known as the Python Virtual Machine (PVM, for the more acronym-inclined among you)*. The PVM sounds more impressive than it is; really, it's not a separate program, and it need not be installed by itself. In fact, the PVM is just a big code loop that iterates through your byte code instructions, one by one, to carry out their operations. The PVM is the runtime engine of Python; it's always present as part of the Python system, and it's the component that truly runs your scripts. Technically, it's just the last step of what is called the "Python interpreter."

The Python Virtual Machine (PVM)



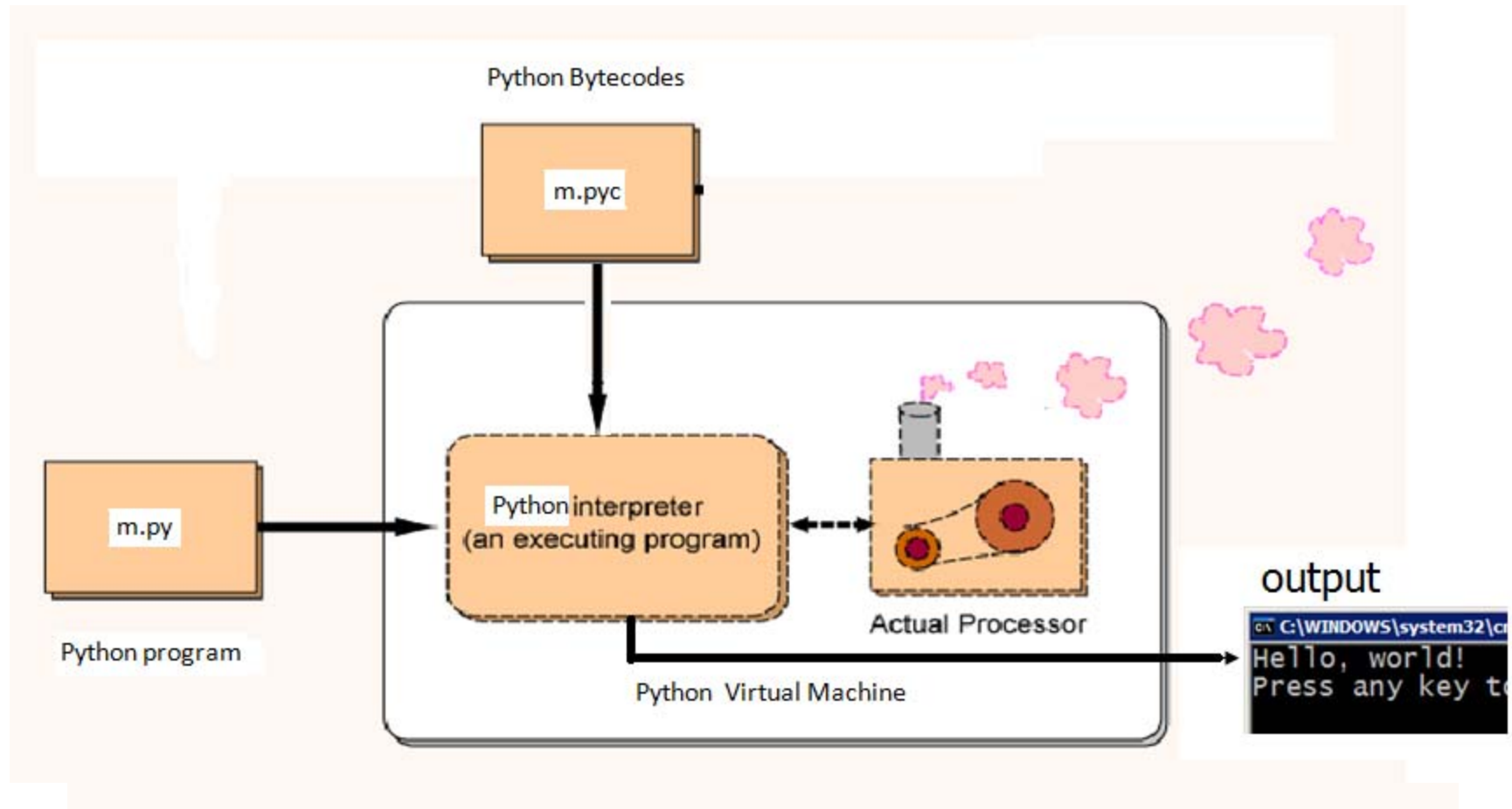
```
*simple.py - C:\Documents and Settings\admin\Desktop\intro-python\lexa
File Edit Format Run Options Window Help
```

```
def proc():
    x=1
    y=x+1
    print("Hello, world!")
```

5	0 LOAD_CONST	1 (1)
	3 STORE_FAST	0 (x)
6	6 LOAD_FAST	0 (x)
	9 LOAD_CONST	1 (1)
	12 BINARY_ADD	
	13 STORE_FAST	1 (y)
7	16 LOAD_GLOBAL	0 (print)
	19 LOAD_CONST	2 ('Hello, world!')
	22 CALL_FUNCTION	1 (1 positional, 0 keyword pair)
	25 POP_TOP	
	26 LOAD_CONST	0 (None)
	29 RETURN_VALUE	

None

The Python Virtual Machine (PVM)



```
*simple.py - C:\Documents and Settings\admin\Desktop\intro-python\lexa
File Edit Format Run Options Window Help
```

```
def proc():
    x=1
    y=x+1
    print("Hello, world!")
```

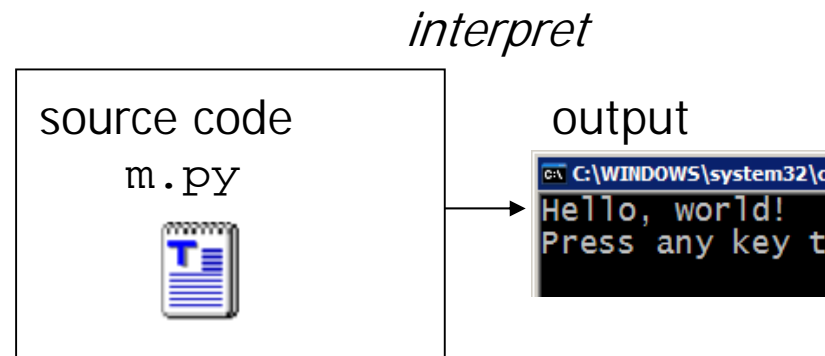
5	0 LOAD_CONST	1 (1)
	3 STORE_FAST	0 (x)
6	6 LOAD_FAST	0 (x)
	9 LOAD_CONST	1 (1)
	12 BINARY_ADD	
	13 STORE_FAST	1 (y)
7	16 LOAD_GLOBAL	0 (print)
	19 LOAD_CONST	2 ('Hello, world!')
	22 CALL_FUNCTION	1 (1 positional, 0 keyword pair)
	25 POP_TOP	
	26 LOAD_CONST	0 (None)
	29 RETURN_VALUE	

None

'6401007d00007c0000640100177d01007400006402008301000164000053'

Compiling and interpreting

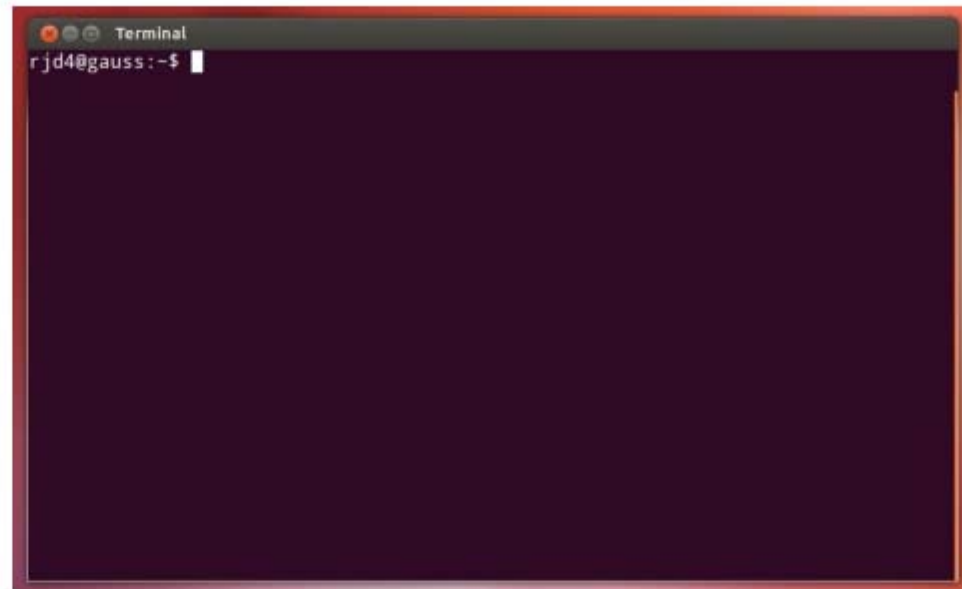
- Python is instead directly *interpreted* into machine instructions.



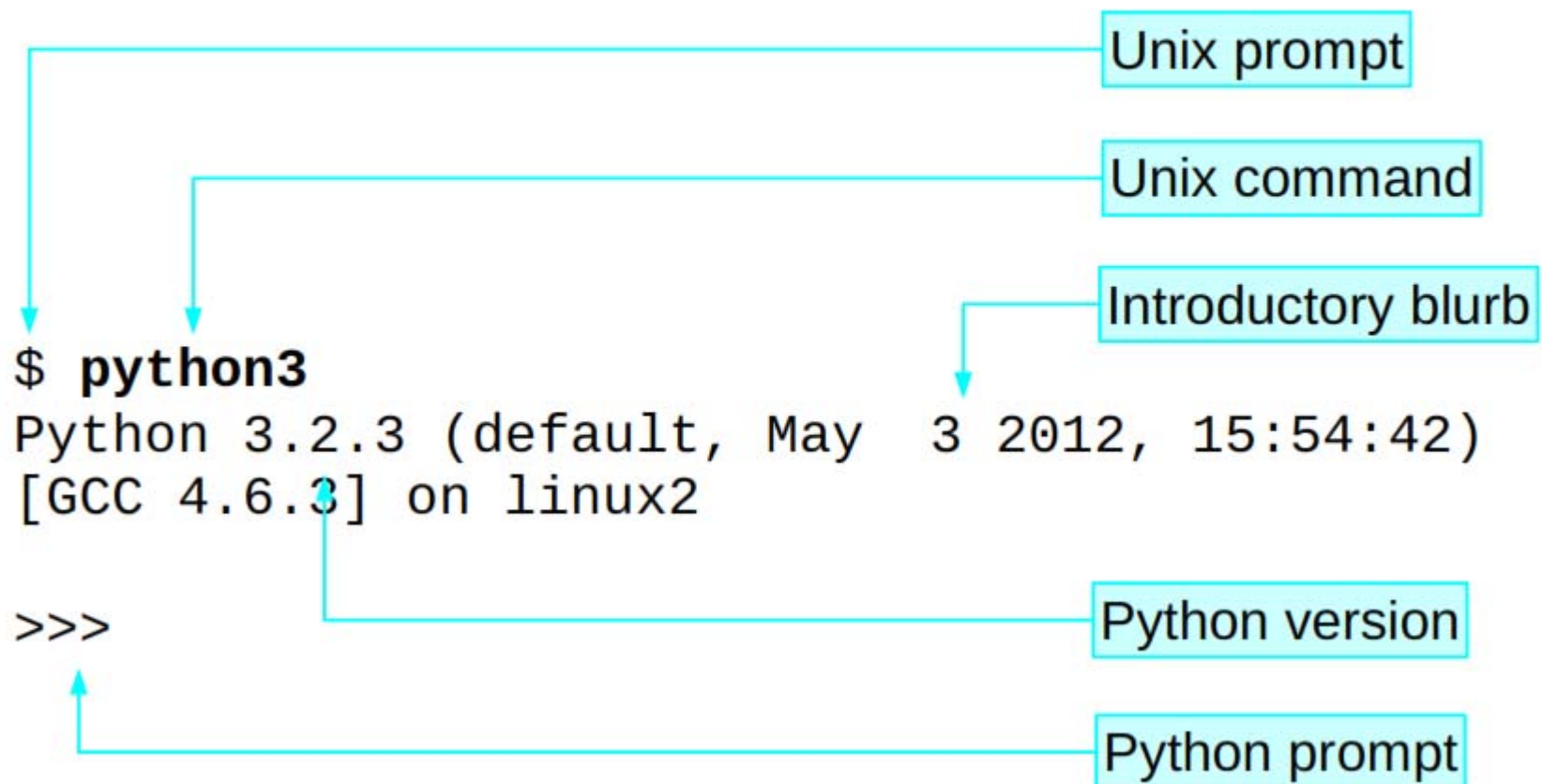
Portability

Most Python programs run unchanged on *all major computer platforms*. *Porting* Python code between Linux and Windows, for example, is usually just a matter of copying a script's code between machines. Moreover, Python offers multiple options for coding portable graphical user interfaces, database access programs, web based systems, and more. Even operating system interfaces, including program launches and directory processing, are as portable in Python as they can possibly be.

Running Python — 1



Running Python — 2

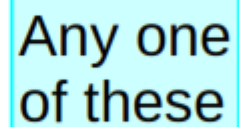


Quitting Python

```
>>> exit()
```

```
>>> quit()
```

```
>>> Ctrl + D
```



Any one
of these

A first Python command

The diagram illustrates the execution of a Python command. It shows a sequence of text elements with arrows pointing to specific parts of the code or output:

- Python prompt**: Points to the first `>>>` prompt.
- Python command**: Points to the `print('Hello, world!')` command.
- Output**: Points to the `Hello, world!` output.
- Python prompt**: Points to the second `>>>` prompt.

```
>>> print('Hello, world!')
```

Hello, world!

```
>>>
```

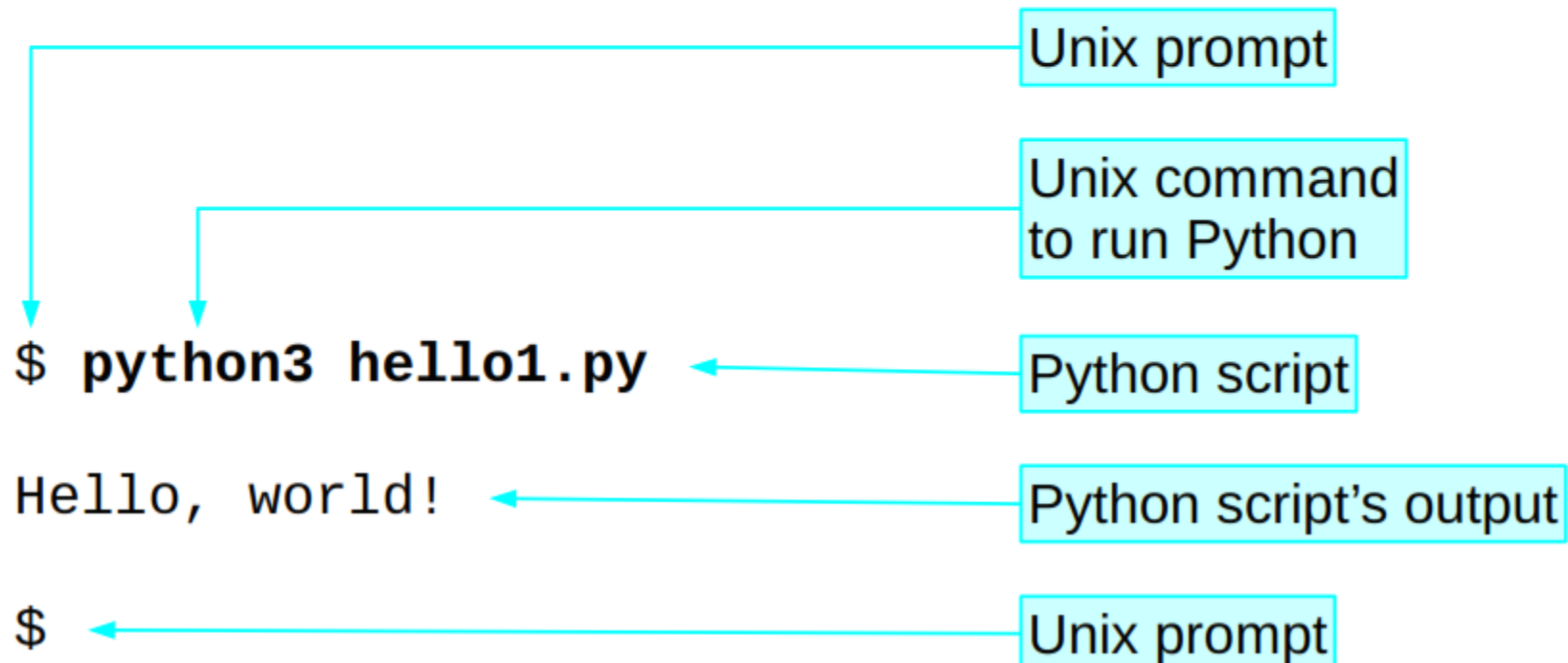
Python scripts

File in home directory

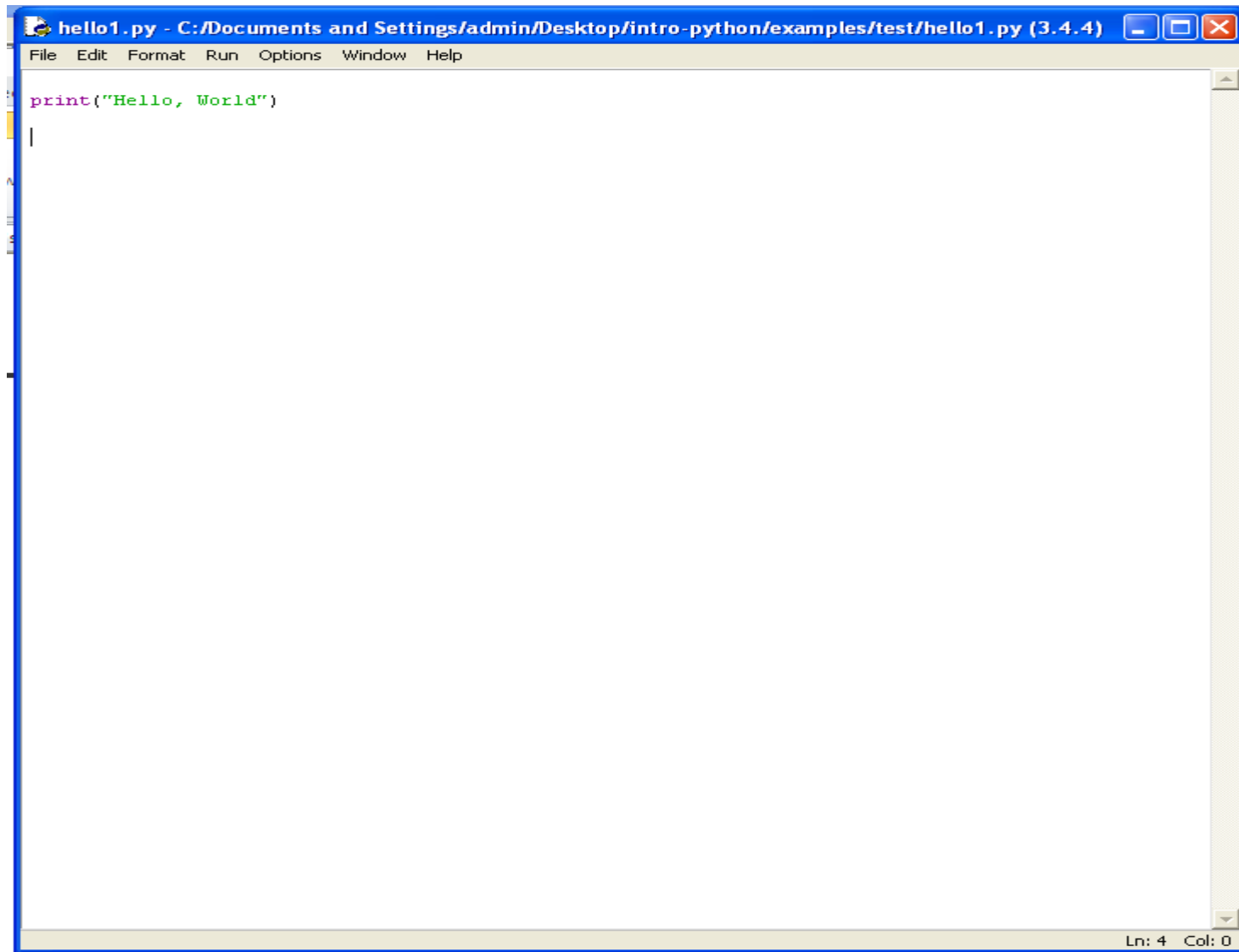
Run from *Unix* prompt

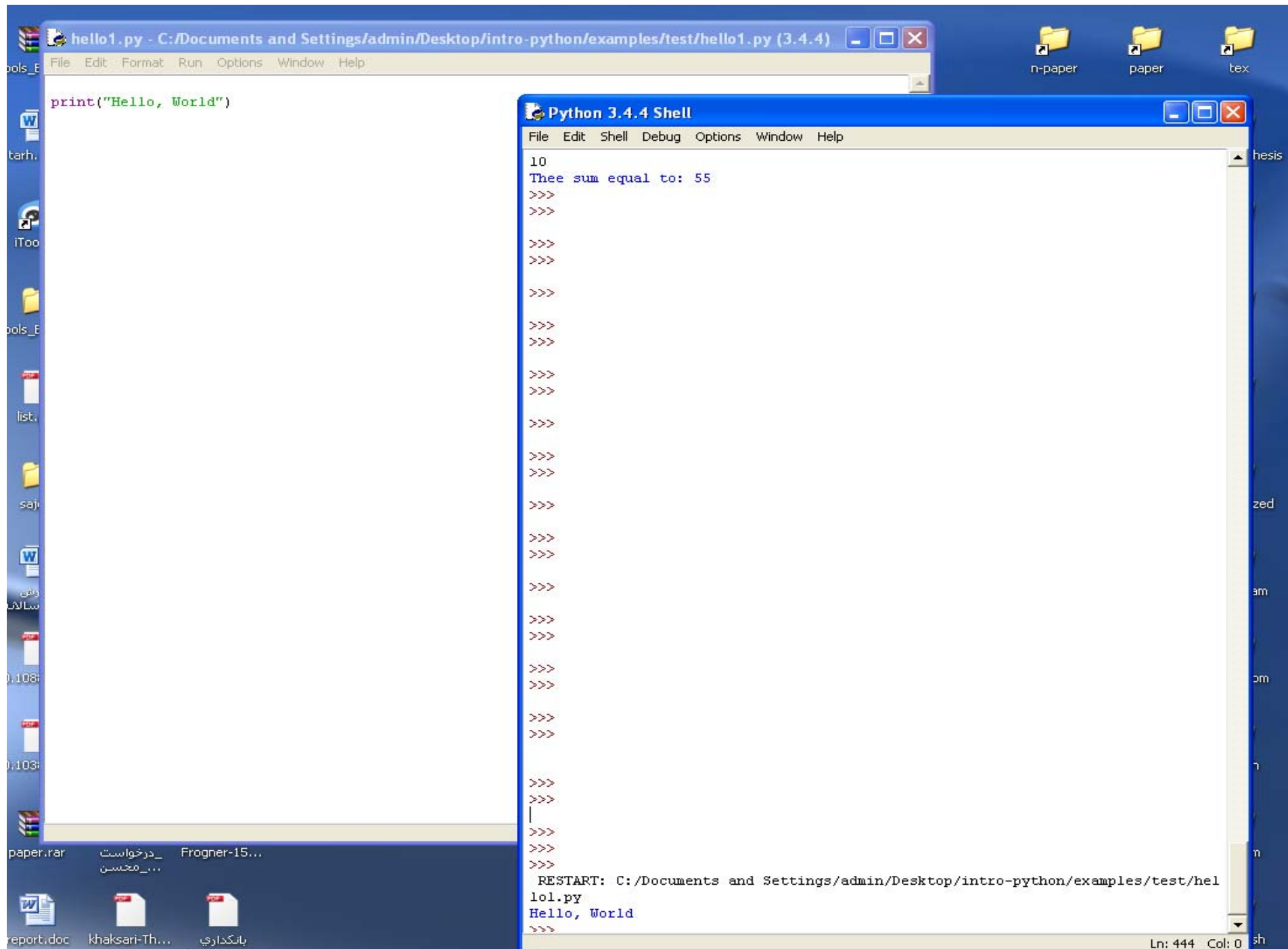
```
print('Hello, world!')
```

hello1.py



Python IDLE





Now here is a more complicated program:

```
print("Jack and Jill went up a hill")
print("to fetch a pail of water;")
print("Jack fell down, and broke his crown,")
print("and Jill came tumbling after.")
```

When you run this program it prints out:

```
Jack and Jill went up a hill
to fetch a pail of water;
Jack fell down, and broke his crown,
and Jill came tumbling after.
```


End