Sample Chapter

Hello World!

Computer Programming for Kids and Other Beginners

Warren and Carter Sande

MANNING

Hello World!

Computer Programming for Kids and Other Beginners

by Warren Sande
and Carter Sande

Chapter 1

brief contents

CHAPTER 1

Getting Started

We will be using the Python computer language to learn programming. To get started, you first need to have Python installed on your computer. After that, you can start learning how to use it. We will begin by giving Python some instructions, and then we will put a few instructions together to make a program.

Installing Python

The first thing you need to do is install Python on the computer you are going to use. It's possible that Python is already installed on your computer, but for most people, that's not the case. So let's look at how to install it.

IN THE GOOD OLD DAYS

In the early days of personal computers (PCs), people had it easy. With a lot of the first PCs, a programming language called BASIC was built in to the computer. They didn't have to install anything. All they did was turn on the computer, and the screen would say "READY", and they could start typing BASIC programs. Sounds great, huh?

Of course, that "READY" was all you got. No programs, no windows, no menus. If you wanted the computer to do *anything*, you had to write a program! There were no word processors, media players, web browsers, or any of the things we are used to now. There wasn't even a Web to browse. There were no fancy graphics and no sound, except the occasional "beep" if you made a mistake!

Installing Python is pretty easy. In the Resources section of this book's web site (www.helloworldbook.com), find the version of the installer that matches your computer's operating system. There are versions for Windows, Mac OS X, and Linux. All the examples in this book use Windows, but using Python in Mac OS X or Linux is very similar. Just follow the instructions on the web site to run the right installer for your system.

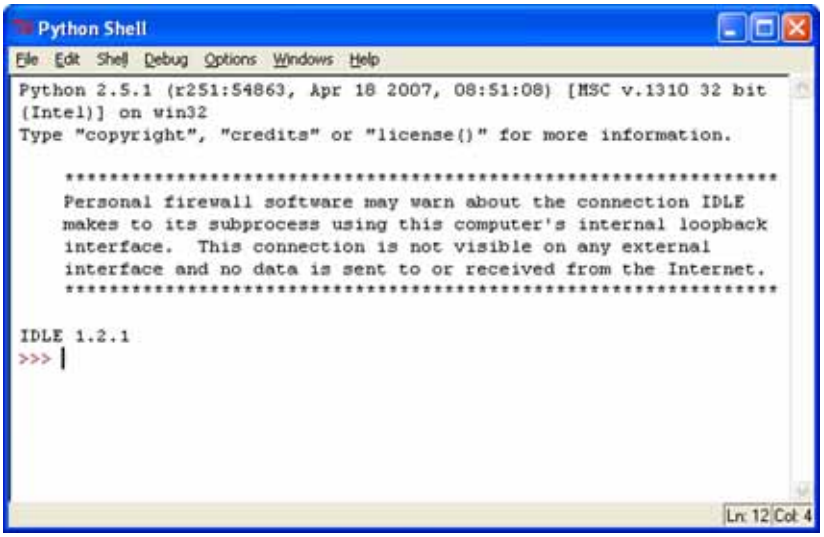
If Python is already installed on your computer, and you are not going to use the installer, you will need to make sure that some "extras" that you'll need for this book are also installed. Have a look at the Installation section of the web site (www.helloworldbook.com) to find out how to do this.

The version of Python that we use in this book is version 2.5. If you use the installer on the book's web site, that's the version you will get. By the time you read this, there might be newer versions of Python out there. All the examples in this book have been tested using Python 2.5. They are likely to work with later versions as well, but I can't see into the future, so there are no guarantees.

Starting Python with IDLE

There are a couple of ways to start using Python. One is called IDLE, and that's the one we will use for now.

In the **Start** menu, under **Python 2.5**, you will see **IDLE (Python GUI)**. Click this option, and you will see the IDLE window open up. It should look something like the window below.



IDLE is a Python *shell*. A shell is basically a way of interacting with a program by typing text, and this shell lets you interact with Python. (That's why you see "Python Shell" in the title bar of the window.) IDLE also happens to be a GUI, which is why it says **Python GUI** in the **Start** menu. IDLE has some other things besides the shell, but we'll get to all that in a minute.

WORD BOX

GUI stands for *graphical user interface*. This means something with windows, menus, buttons, scrollbars, etc. Programs that don't have a GUI are called *text-mode* programs, *console* programs, or *command-line* programs.

The ">>>" in the previous figure is the Python *prompt*. A prompt is what a program displays when it is waiting for you to type something. The ">>>" prompt tells you that Python is ready for you to start typing Python instructions.

Instructions, please

Let's give Python our first instruction.
With the cursor at the end of the ">>>" prompt, type

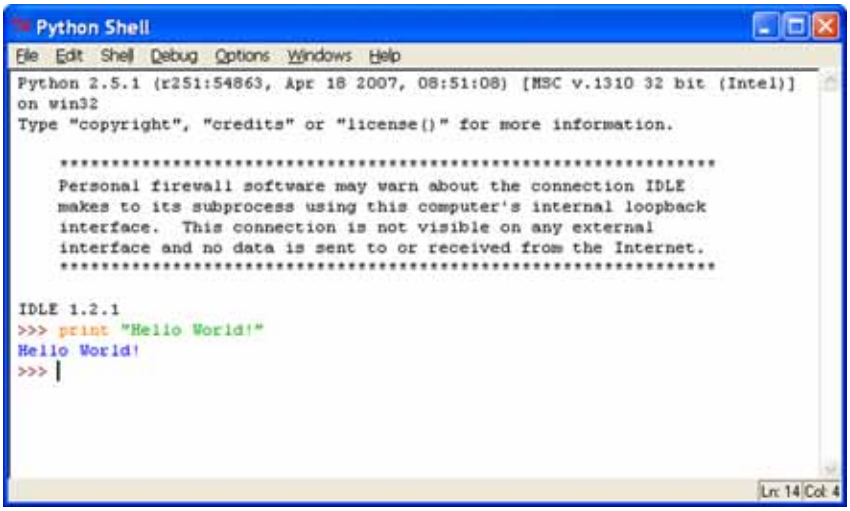
```
print "Hello World!"
```

and press the Enter key. (On some keyboards this is called the Return key.) You need to press the Enter key after every line you type.

After you press the Enter key, you should get this response:

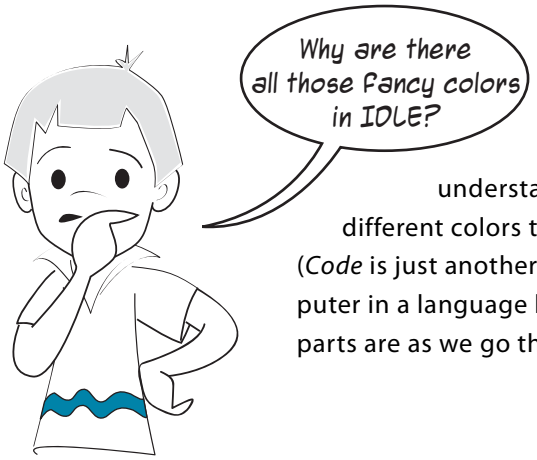
```
Hello World!
>>>
```

The figure below shows how that looks in the IDLE window.



Python did what you told it: it printed your message. (In programming, `print` often means to display text on the screen, instead of printing it on a piece of paper using your printer.) That one line is a Python *instruction*. You're on your way to programming! The computer is under your command!

By the way, in learning to program, there is a tradition that the first thing you make the computer do is display "Hello World!" That's where the title of this book comes from. You are following that tradition. Welcome to the world of programming!



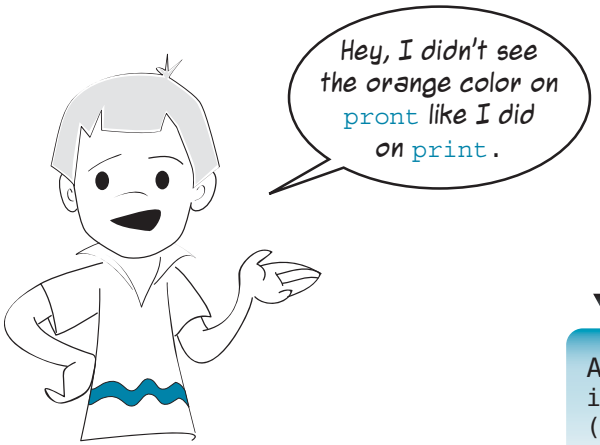
Good question! IDLE is trying to help us understand things a bit better. It's showing things in different colors to help us tell different parts of the code apart. (*Code* is just another term for the instructions you give to the computer in a language like Python.) I will explain what the different parts are as we go through the rest of this book.

If it doesn't work

If you made a mistake, you might see something like this:

```
>>> pront "Hello World!"
SyntaxError: invalid syntax
>>>
```

That error message means you typed something that Python didn't understand. In the example above, `print` is misspelled `pront`, and Python doesn't know what to do with that. If that happens to you, try it again and make sure you type it exactly like in the example.



That's right. That's because **print** is a Python keyword, and **pront** is not.

WORD BOX

A *keyword* is a special word that is part of the Python language (also known as a *reserved* word).

Interacting with Python

What you just did was use Python in interactive mode. You typed a command (an instruction) and Python *executed* it immediately.

WORD BOX

Executing a command, instruction, or program is just a fancy way of saying "running" it, or "making it happen."

Let's try something else in interactive mode. Type this at the prompt:

```
>>> print 5 + 3
```

You should get this:

```
8
>>>
```

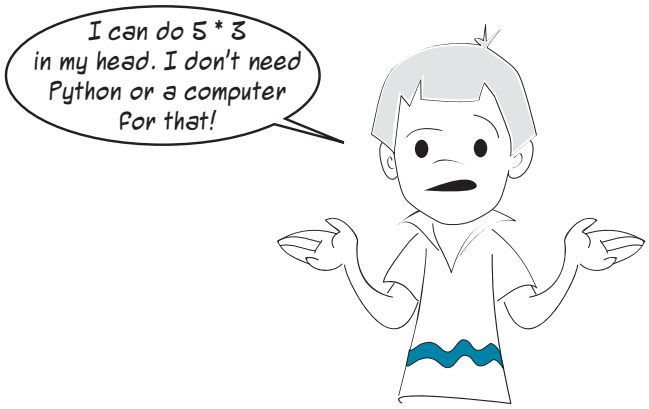
So Python can do addition! That shouldn't be surprising, because computers are good at arithmetic.

Let's try one more:

```
>>> print 5 * 3
15
>>>
```

In pretty much all computer programs and languages, the * symbol is used for multiplication. That character is called an *asterisk* or *star*.

If you are used to writing "5 times 3" as "5 x 3" in math class, you'll have to get used to using * for multiplication in Python instead. (It's the symbol above the number 8 on most keyboards.)

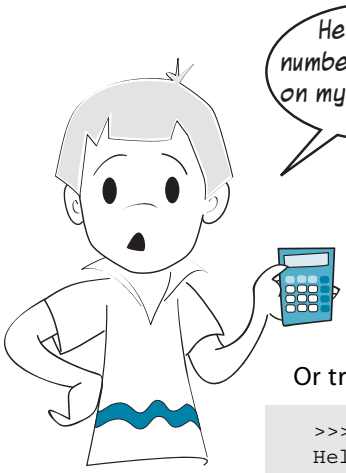


Okay, how about this one:

```
>>> print 2345 * 6789
15920205
>>>
```

Okay, how about this one:

```
>>> print 12345678987654321234567890 * 98765432123456789876543212345678987654321
121932632007315960006096522024081660722451126352690
>>>
```



That's right. With the computer, you can do math on really, really big numbers. Here's something else you can do:

```
>>> print "cat" + "dog"
catdog
>>>
```

Or try this:

```
>>> print "Hello " * 20
Hello Hello Hello Hello Hello Hello Hello Hello Hello Hello
Hello Hello Hello Hello Hello Hello Hello Hello Hello Hello
```

Besides math, another thing computers are good at is doing things over and over again. Here we told Python to print "Hello" twenty times.

We'll do more in interactive mode later, but right now it's ...

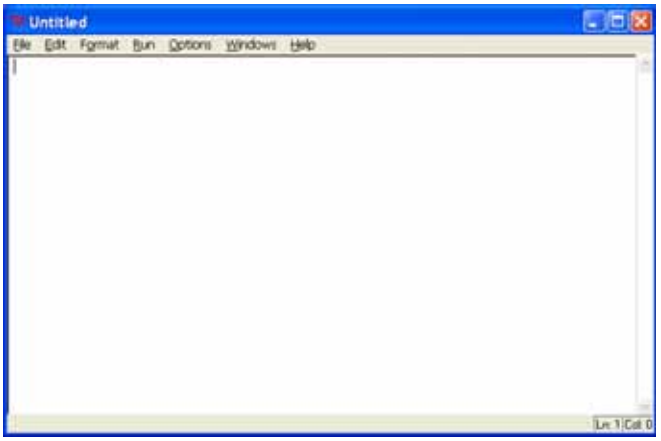
Time to program

The examples we've looked at so far are single Python instructions (in interactive mode). While that's great for checking out some of the things Python can do, those examples are not really programs. As I mentioned before, a program is a number of instructions collected together. So let's make our first Python program.

When I am talking about menu selections, like ***File > New,*** *the first part (****File*** *in this case) is the main menu. The* ***>*** *tells you that the next thing (****New*** *in this case) is an item in the File menu. I will use that notation throughout the book.*

First, you need a way to type in our program. If you just type it in the interactive window, Python won't "remember" it. You need to use a text editor (like Notepad for Windows or TextEdit for Mac OS X) that can save the program to the hard drive. IDLE comes with a text editor that is much better for what you need than Notepad. To find it, select **File > New Window** from IDLE's menus.

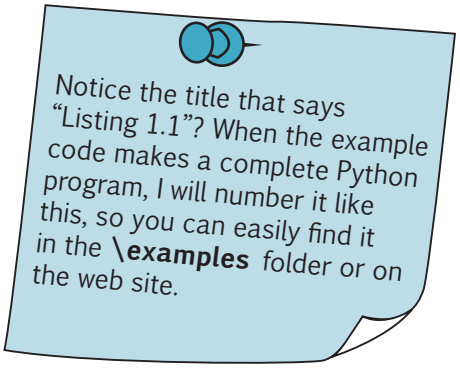
You will see a window like in the figure below. The title bar says "Untitled" because you haven't given it a name yet.



Now, type the program in listing 1.1 below into the editor.

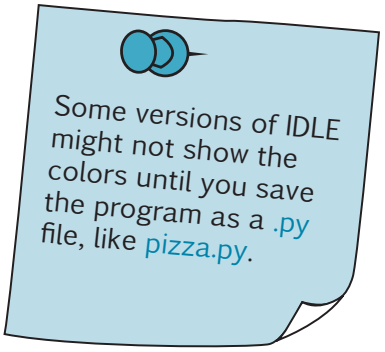
Listing 1.1 Our first real program

```
print "I love pizza!"
print "pizza " * 20
print "yum " * 40
print "I'm full."
```



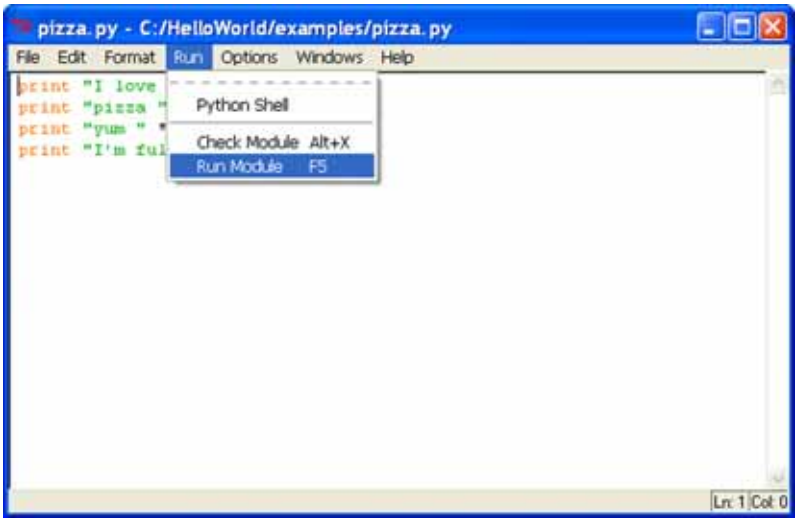
When you are done, save the program using the **File > Save** or **File > Save As** menu option. Call the file **pizza.py**. You can save it wherever you like (as long as you remember where it is, so you can find it later). You might want to create a new folder for saving your Python programs. The ".py" part at the end is important, because it tells your computer that this is a Python program, and not just any old text file.

You might have noticed that the editor used some different colors in the program. Some words are in orange and others are in green. This is because the IDLE editor assumed that you would be typing in a Python program. For Python programs, the IDLE editor shows Python keywords in orange and anything in quotation marks in green. This is meant to help you read your Python code more easily.

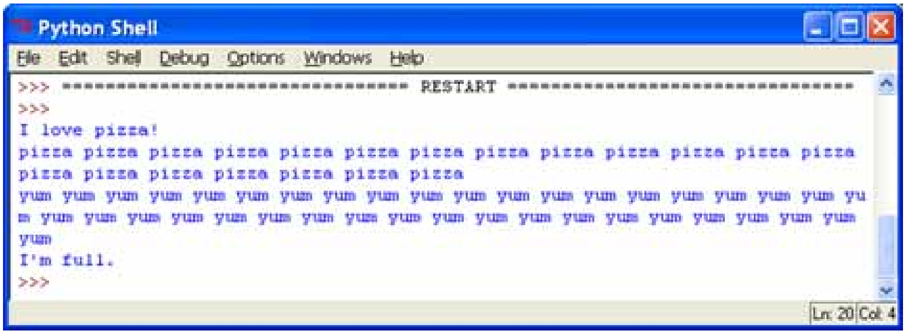


Running your first program

Once you have saved your program, go to the **Run** menu (still in the IDLE editor), and pick **Run Module** (as shown in the next figure). This will run your program.



You will see that the Python shell window (the one that first came up when you started IDLE) becomes active again, and you will see something like the following:



The `RESTART` part tells you that you started running a program. (This will be helpful when you are running your programs over and over again to test them.)

Then the program runs. Okay, so it doesn't do very much. But you got the computer to do what you told it to do. Our programs will get more interesting as we go along.

If something goes wrong

What happens if you have an error in your program, and it doesn't run? There are two different kinds of errors that can happen. Let's look at both kinds, so you will know what to do if either one happens to you.

Syntax errors

IDLE does some checking of your program before it even tries to run it. If IDLE finds an error, it is usually a *syntax error*. Syntax is the spelling and grammar rules for a programming language, so a *syntax error* means that you have typed something that is not proper Python code.

Here is an example:

```
print "Hello, and welcome to Python!"
print "I hope you will enjoy learning to program."
print Bye for now!"
```

Missing quote mark

We missed a quote mark between `print` and `Bye for now!"`

If you tried to run this program, IDLE would pop up a message saying "There's an error in your program: invalid syntax." Then you would have to look at your code to see what's wrong. IDLE will highlight (in red) the place where it found the error. It might not be exactly where the problem is, but it should be close.

Runtime errors

The second kind of error that can happen is one that Python (or IDLE) can't detect before it runs the program. This kind of error only happens when the program runs, so it is called a *runtime error*. Here's an example of a runtime error in a program:

```
print "Hello, and welcome to Python!"
print "I hope you will enjoy learning to program."
print "Bye for now!" + 5
```

If we save this and try to run it, the program actually starts to run. The first two lines are printed, but then we get an error message:

```
>>> ============================ RESTART ============================
>>>
Hello, and welcome to Python!
I hope you will enjoy learning to program.
Traceback (most recent call last):
  File "C:/HelloWorld/examples/error1.py", line 3, in <module>
    print "Bye for now!" + 5
TypeError: cannot concatenate 'str' and 'int' objects
>>>
```

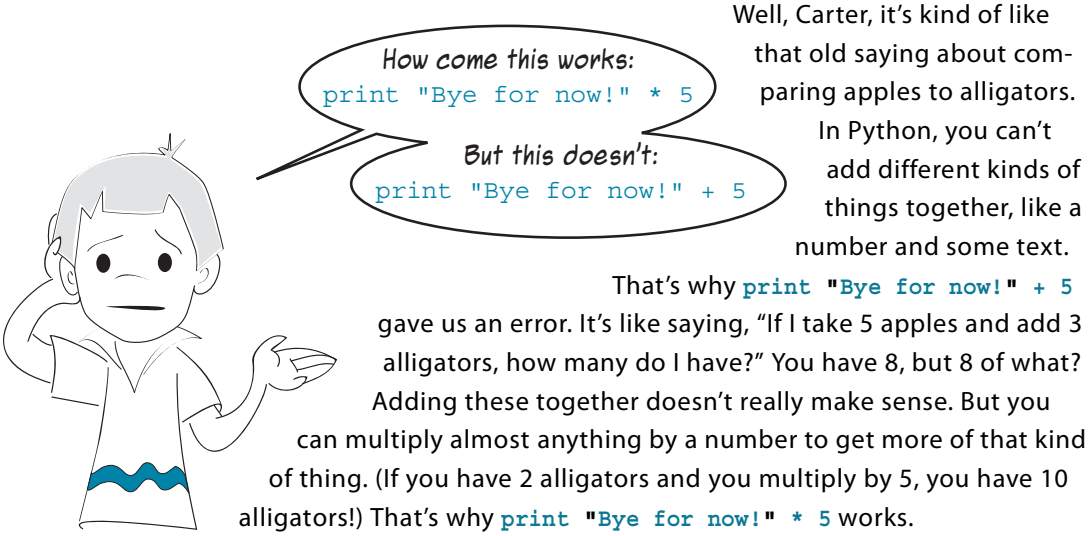
Start of the error message

Where the error was

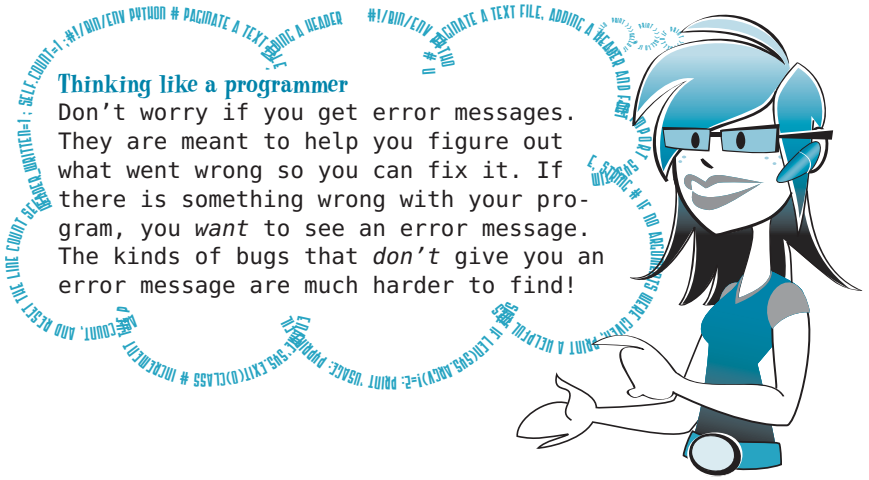
The "bad" line of code

What Python thinks is wrong

The line starting with `Traceback` is the start of the error message. The next line tells you where the error happened—the filename and line number. Then it displays the bad line of code. This helps you find where the problem is in your code. The last part of the error message tells you what Python thinks is wrong. Once you know more about programming and Python, it will be easier to understand what the message means.



Well, Carter, it's kind of like that old saying about comparing apples to alligators. In Python, you can't add different kinds of things together, like a number and some text. That's why `print "Bye for now!" + 5` gave us an error. It's like saying, "If I take 5 apples and add 3 alligators, how many do I have?" You have 8, but 8 of what? Adding these together doesn't really make sense. But you can multiply almost anything by a number to get more of that kind of thing. (If you have 2 alligators and you multiply by 5, you have 10 alligators!) That's why `print "Bye for now!" * 5` works.



Thinking like a programmer

Don't worry if you get error messages. They are meant to help you figure out what went wrong so you can fix it. If there is something wrong with your program, you *want* to see an error message. The kinds of bugs that *don't* give you an error message are much harder to find!

Your second program

The first program didn't do much. It just printed some stuff on the screen. Let's try something a bit more interesting.

The next code in listing1.2 is for a simple number-guessing game. Start a new file in the IDLE editor using **File > New Window**, just like you did the first time. Type in the code from listing 1.2 and then save it. You can call it whatever you want, as long as it ends with ".py". **NumGuess.py** might be a good name.

It's only 18 lines of Python instructions, plus a few blank lines to make it easier to read. It shouldn't take too long to type in. Don't worry that we haven't talked about what this code all means yet. We'll get to that very soon.

Listing 1.2 Number-guessing game

```
import random

secret = random.randint(1, 100)
guess = 0
tries = 0

print "AHOY!  I'm the Dread Pirate Roberts, and I have a secret!"
print "It is a number from 1 to 99.  I'll give you 6 tries. "

while guess != secret and tries < 6:
    guess = input("What's yer guess? ")
    if guess < secret:
        print "Too low, ye scurvy dog!"
    elif guess > secret:
        print "Too high, landlubber!"
    tries = tries + 1
```

Picks a secret number

Gets the player's guess

Allows up to 6 guesses

Uses up one try

```
if guess == secret:
    print "Avast! Ye got it!  Found my secret, ye did!"
else:
    print "No more guesses!  Better luck next time, matey!"
    print "The secret number was", secret
```

Prints message at end of game

When you are typing it in, notice the indenting of the lines after the `while` instruction, and the extra indenting of the lines after `if` and `elif`. Also notice the colons at the ends of some of the lines. If you type the colon in the correct place, the editor will help you by indenting the next line for you.

Once you've saved it, run it using **Run > Run Module**, just like you did for the first program. Try playing it and see what happens. Here is a sample of when I ran it:

```
>>> ====================== RESTART ======================
>>>
AHOY!  I'm the Dread Pirate Roberts, and I have a secret!
It is a number from 1 to 99.  I'll give you 6 tries.
What's yer guess? 40
Too high, landlubber!
What's yer guess? 20
Too high, landlubber!
What's yer guess? 10
Too low, ye scurvy dog!
What's yer guess? 11
Too low, ye scurvy dog!
What's yer guess? 12
Avast! Ye got it!  Found my secret, ye did!
>>>
```

It took me five guesses to get the secret number, which turned out to be 12.

We will be learning all about the `while`, `if`, `else`, `elif`, and `input` instructions in the next few chapters. But you can probably already get the basic idea of how this program works:

- The secret number is randomly picked by the program.
- The user inputs his guesses.
- The program keeps checking the guess against the secret number: is it higher or lower?
- The user keeps trying until he guesses the number or runs out of turns.
- When the guess matches the secret number, the player wins.

What did you learn?

Whew! We covered quite a lot. In this chapter, you

- installed Python.
- learned how to start IDLE.
- learned about interactive mode.
- gave Python some instructions, and it executed them.
- saw that Python knows how to do arithmetic (including really big numbers!)
- started the IDLE text editor to type in your first program.
- ran your first Python program!
- learned about error messages.
- ran your second Python program: the number-guessing game.

Test your knowledge

1. How do you start IDLE?
2. What does `print` do?
3. What is the symbol for multiplication in Python?
4. What does IDLE display when you start to run a program?
5. What is another word for running a program?

Try it out

1. In interactive mode, use Python to calculate the number of minutes in a week.
2. Write a short program to print three lines: your name, your birth date, and your favorite color. The output should look something like this:

```
My name is Warren Sande.
I was born January 1, 1970.
My favorite color is blue.
```

   Save the program and run it. If the program doesn't do what you expect, or you get any error messages, try to fix it and make it work.

PROGRAMMING

Hello World!

Computer Programming for Kids and Other Beginners

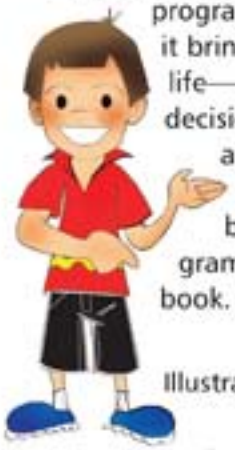
Warren Sande and Carter Sande

"Computer programming is a powerful tool for children to 'learn learning.' ... Children who engage in programming transfer that kind of learning to other things."

—Nicholas Negroponte
One Laptop Per Child Project, January 2008

Your computer won't respond when you yell at it, so why not talk to it in its own language? If you learn to program, you can do just that. You'll be able to do really cool things quickly, and even make your own games! Programming is fun!

Hello World! Computer Programming for Kids and Other Beginners is a wonderfully written introduction to programming. Using fun examples, it brings computing concepts to life—concepts like memory, loops, decisions, input and output, data, and graphics. It's written in a language a kid can follow, but anyone who wants to program a computer can use this book. Even adults!



Illustrated by Martin Murtonen

For online access to the authors, go to www.manning.com/HelloWorld

Hello World! uses Python, the programming language also chosen for the One Laptop Per Child Project. Readers with no previous knowledge of computing will be programming in no time at all.

WHAT'S INSIDE

- Explains everything in clear language—no "geek speak"
- Loaded with pictures, cartoons, and fun examples
- Complete set of practice questions and exercises
- Reviewed by professional educators, kid-tested, and parent-approved

Warren Sande is an Electronic Systems Engineer who uses Python and other computer languages in his daily work. His son, ***Carter Sande***, is an elementary school student who loves computers, playing the piano, jumping on the trampoline, bike riding, and his little sister. His nickname at school is "Tech Support."

ISBN-13: 978-1933988498
ISBN-10: 1933988495
53499

MANNING US $34.99 / Can $34.99