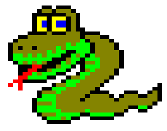
Start Programming with Python



by

Bryce Embry

www.bembry.org

Table of Contents:

Starting Out

So, you want to program? Well, first you have to choose a language. Lots of folks want to start out by learning Java or C++, thinking they'll earn millions of dollars if they can master one of these quickly. But both of these languages are rather complicated; it takes half a dozen lines of code just to say "Hello world." C++ and Java have earned their reputation as powerful languages, but wouldn't it be nice to learn a powerful language that wasn't so complicated? If that dream sounds good to you, let me introduce you to Python.

Python is an open-source object-oriented programming language that combines the power of Java with a simplicity all its own. The Python language is used to build powerful programs, has a friendly and supportive user community, and can help you create the tools you need faster than you could in Java or C++. Some folks claim it is the best programming language in the world. There's only one way to find out – let's try it.

Before we can start writing programs in Python we need to install the Python interpreter. Python is an interpreted language, which means that the Python interpreter must be installed on a computer before it can run the code we write in Python. The Python interpreter is available free at www.python.org for Windows, Mac OS, and Linux, so many Python programs can be run on any operating system without changes.

We are going to start out writing code in IDLE, the interactive interpreter environment that comes with Python. So, once you have Python installed, fire up the IDLE program (Python GUI) and let's start playing.

Code:

Type these lines in the Python interpreter IDLE. Press enter after each line.

```
print "hello world"
print 3*4
print 3**2
print 3/4
print 3.0/4
print 11%5
import calendar
calendar.prcal(2002)
```

The division results may not make sense at first because Python handles integers and decimals differently. When two whole numbers (integers) are divided, Python returns the resulting integer portion of the quotient only. So 3/2 yields 1, and 1/2 yields 0.

Points to Remember:

- The Python interpreter must be installed on a computer in order to run Python programs.
- Python commands can be run in the interactive interpreter.
- Python is case sensitive. "Print" is not the same as "print".
- To have Python print text, the text must be enclosed in quotation marks.
- The mathematic operators in Python are:

+	add
-	subtract
*	multiply
/	divide
**	exponential power
%	remainder

- When two whole numbers (integers) are divided, Python returns only the integer (whole number) portion of the quotient.

Program 1: Variables

Now that we've played around in the interpreter, it's time to start our first program. We've gotten Python to say “Hello”, now can we get it to call the user by his name?

For all of the programs we write we are going to use the text editor instead of the interactive interpreter. In the interactive interpreter, our code is executed immediately and then lost. When we us a text editor, we can write out the whole program, save it, and then have Python run it all at once. Technically, we could write Python code in Notepad or any other text editor, Python comes with a special editor that has some useful features. To get to this editor, in the IDLE interpreter click “File” --> “New Window”.

When we save a program, we must save it with the extension “.py” so that Python will recognize it as code. This code file can be executed from a DOS prompt by typing either “myscript.py” or “python myscript.py”, depending on your installation of Python. Usually, though, we'll run our programs by clicking “Edit --> Run Script” in the IDLE text editor. Or, if we are really lazy, we'll press [Ctrl] [F5] to run the program.

Now that we're writing code in the text editor, let's figure out how to get our program to personalize its greeting. In order to get our program to know our name, we are going to create a variable. Variables are code names we make up that let us feed information we create in one part of a program to a process in a different part of the program. Variable names in Python can be any text we choose as long as it does not start with a number and is not a reserved Python keyword.

Chances are that you don't have all the reserved Python keywords memorized, so you may not know which words you cannot use as a variable name. This is where writing our code in IDLE comes in handy. Any time you type a reserved Python keyword in the IDLE text editor, IDLE colors the word orange. So, if you make up a variable name and it turns orange, you'll know its a reserved keyword and you can't use that word.

Code:

```
user_name = raw_input("What is your name? ")
print "Hello", user_name
```

In line one we set our variable name “user_name” equal to something called “raw_input()”. This raw_input() creature is a built-in function that asks the user for a response and stores that response in the variable. IN quotation marks inside the parentheses, we type the prompt we want the users to see.

On line two we tell Python to print the word “Hello” and then print whatever the user typed in. The comma between “Hello” and user_name is important. Python does not use a lot of punctuation marks to direct the code, but the few it uses are important to remember. Here, the comma tells Python that there is more to print on this line after it's done printing “Hello”.

Points to Remember:

- Python code is saved with the extension .py.
- Code can be executed in IDLE by pressing [Ctrl] [F5]
- Variable names cannot start with a number.
- Variable names cannot be keywords reserved by Python.
- In Python, each command must be on a separate line.
- Python does not use a lot of punctuation, but when it does, it is critical.
- Values being sent to functions are enclosed in parentheses.

Program 2: String Operators

A favorite punishment of some teachers is to make students copy sentences a hundred times. If we could just get the computer to write their sentences for them, we could make life a lot easier on these poor students.

We've already learned how to let a user type in a sentence and have the program print it out, now we need to figure out how to do that a hundred times. We could write the print command a hundred times in our program, but that's not any better than having to write “I will not talk in school” a hundred times, so there has to be a better way. We could set up a loop that will execute our print command a hundred times, but we aren't ready for that yet. The easiest way to get our computer to print out a sentence one hundred times is to tell Python to multiply our sentence by one hundred.

Code:

```
sentence= raw_input("Enter sentence: ")
print sentence*100
```

The trick to this code is the second line, which tells Python to multiply our sentence by one hundred and print the result. When multiplication operator is used on text strings, Python repeats the string as many times as we tell it.

When you run this program, you'll notice that Python printed the sentence one hundred times, but the sentences are all crammed together. We need some way to tell Python to put each sentence on a new line. Technically, we can do this by adding a “newline” character to the end of the sentence before printing it.

```
Sentence = raw_input("Enter sentence: ")
text_line = sentence + “\n”
print text_line*100
```

In line two, the + operator takes our sentence and adds something to the end of it. Who would ever think of using + to add text strings together? Go figure.

That strange “\n” that we added to our sentence is called an escape character and is the code for a line break. There are a number of escape characters available in Python. They are all preceded by a back-slash and must be in a string (quotation marks) to work. These special codes include \t for a tab, \” for a quotation mark (when we want one printed, instead of indicating the end of a string), and \\ when we want a backslash.

Points to Remember

- Text strings can be repeated by using the * operator.
- Text strings can be added together using the + operator.
- Special codes can be included in a text string by using escape characters such as:
  - \n Newline (line break)
  - \t Tab
  - \” Quotation mark
  - \\ A real, honest-to-goodness, backslash
- Escape characters must be inside quotation marks

Program 3: Data Types

Come to think of it, teachers don't always make students write a sentence a hundred times. Sometimes they want you to write a sentence five hundred times, or even more. I remember working at a school whose policy was to assign students writing assignments as punishment. At the end of each day students would compare how many pages they had to write that night. So, let's expand our program to let the student tell us how many sentences to print out.

If you're thinking ahead, you might have figured that we could just make another raw_input() statement that asks the user how many times they want the sentence written, then tell the computer to multiply by that number. Well, that's almost right. The problem we'd have with this approach is that when Python reads information from raw_input(), it reads it in as a text string. Telling the program to multiply a sentence by a text string just makes no sense. How can you multiply "I will not talk in class" by "mud"?

In Python, there are different kinds of data. There are strings (text, like words and sentences), integers (whole numbers), floating point numbers (numbers with decimal values), lists (lists of stuff), and even complex numbers, long numbers, tuples,and dictionaries (don't ask). Data can be changed from one type to another using the correct commands. So, in order to let our troubled user select how many times to print his sentence we must first ask him for the sentence (a string), ask him how many times he needs the sentence printed (a text string), then convert his numeral (text string that looks like a number) into an integer. Once we have the integer, we multiply his sentence by the integer and print it out.

Code

```
sentence = raw_input("Enter sentence: ")
text_line = sentence + "\n"
repeats_entry = raw_input("How many times? ")
num_repeats = int(repeats_entry)
print text_line * num_repeats
```

This program is the same as the one before with the addition of lines three and four. Line three asks the user how many times he wants his sentence printed. We expect the user to enter a numeral on this line. Line four takes the numeral the user typed in and changes it into an integer using the int() function. If the user types in "5", it will be changed into the number five and our program will work. If the user types in the word "five", the int() function will have no idea what number he wants and will just crash the program. The int() function can only change a numeral into an integer, it can't read your mind.

Points to Remember:

- Python has different types of data:
  - Strings (text)
  - Integers (whole numbers, positive or negative)
  - Floating point numbers (numbers with decimal values)
  - And more ... (long numbers, complex numbers, tuples, lists, dictionaries)
- The raw_input() function imports data as a string (text).
- Data can be converted from one type to another, if it is in an acceptable form.
- The int() function changes a numeral into an integer value.

Program 4: Random Text

Having aided and abetted our miscreants, it's time to turn our attentions to loftier matters – garbage text. There comes a time in a geek's life when he wants a chunk of random text. It doesn't matter what it says, as long as it fills up space and doesn't have a predictable pattern.

To get Python to print out random text, first we will want a list of words for Python to choose from. Python has a special data type called a list that will just fit our needs. We create a list in Python by placing all the choices in brackets [ ] and putting a comma between each choice. Our choices can be strings, integers, other lists, or anything other type of Python data. For our random text, we'll make a list of strings, with each word (or phrase) enclosed in quotation marks to identify it as a string.

Once we have a list of choices, we will use a special function in Python to randomly choose items from our list. Python has a myriad of special functions available. Most of these functions are stored in sets called modules and have to be imported into Python before they can be used. Module sets include functions for email, HTML parsing, sound, network sockets, random selections, calendars, files, operating system functions, and a whole lot more, and they can make a seemingly impossible job very simple. To see a list of modules, click "Help --> Documentation" in IDLE, then select the Module Index. You'll be amazed what all you can do with just a few commands.

For our program, we need to import the module called "random". That module contains a number of different functions, so we'll have to declare which one we want. The function designed to randomly pick an item out of a list is called "choice". To use this "choice" function in our code we must identify that "choice" is a function from the random module, so we call it using the code "random.choice()". The random.choice() function needs to know what list to randomly choose its text from, so we put the name of our list inside the parentheses.

Code

```
text_choices = ["dog", "cat", "bird", "fish"]
repeats_entry = raw_input("How many words? ")
num_repeats = int(repeats_entry)
import random
word = random.choice(text_choices)
print word * num_repeats
```

When we run this program it only chooses one key word and prints it over and over. We were really wanting the program to pick a new word each time. Can you see why this program doesn't do that? To get this program the way we want, we need to learn about looping code, which we'll do next.

Points to Remember:

- Lists are contained in brackets [ ] with each item separated by a comma.
- Modules contain pre-made functions for Python.
- To use the functions in a module, you must import the module first.
- To call the function, give the module name, a period, and then the function name.
- The "Module Index" in the documentation has lots of cool stuff.

Program 5: Loops

One of the joys of writing a computer program is making your computer automatically do those repetitive tasks that can drive you crazy, like changing the extension of every file in a directory, creating dozens (or hundreds) of new user accounts, searching through files for a certain word, and so forth. The tasks are routine -- do step A, then B, then C, then D, get a new file, repeat. In programming, we call it a loop.

The mundane task we want our program to do is pick a new random word, print it, then repeat. Anything that we want the program to repeat we put inside the loop, everything else is outside the loop. Inside the loop in our program we need to tell Python to pick a new word and print it, since that is the code we want repeated. We don't need Python to repeat the part about asking us how many words we want, so we'll keep that outside the loop.

To set up our loop, first we create a counter variable. This is a variable that we'll use to keep track of how many times the program has gone through the loop. We will start this counter at zero and add one to it each time the program goes through the loop. It is very important to remember to add one to it during the loop, otherwise our poor little computer will spend the rest of its days printing random text and never increment the counter (it will always think it's still on the first pass through the loop). This is a common error to make. If your program seems to be not responding, it may be stuck in a loop. Press [Ctrl] [C] to rescue it.

We start the loop in our program with the keyword “while”. Since we know how many words we want, we tell the program “while our counter is less than the number of words we want, do this.” Actually, we don't say it exactly like that. We have to phrase the test in the form of a mathematical expression, so it looks more like “while count < num_repeats”. At the end of this line we put a colon to let the program know we are starting the section of code it is supposed to repeat.

After we create our while test, we write the instructions we want repeated. Each instruction that is part of our loop is indented to identify it as part of the special loop program. This indentation doesn't just make the code look prettier, it is required in Python. The indentation is what Python uses to separate the portion of the program that is part of the loop from the portion that is outside the loop.

Code:

```
repeats_entry = raw_input("How many words? ")
num_repeats = int(repeats_entry)
text_choices = ["dog", "cat", "bird", "fish"]
counter = 1
import random
while counter < num_repeats:
        word = random.choice(text_choices)
        print word
        counter += 1
```

The last line has a nifty little operator, +=. The code means add one to the counter variable and is a shortcut to writing “counter = counter + 1”. This += operator will work for adding text to a string just as well as it does in adding a number to an integer. There are also some related operators, such as - = (subtract from variable) and *= (multiply variable by), which can be useful.

And now, at last, our program does what we want; it creates random text.

Points to Remember:

- Sections of code can be repeated by placing them in a while loop.
- A "while" loop will repeat until the test statement is false.
- Comparison tests use the following syntax:

A = = B	A equals B
A != B	A does not equal B
A < B	A is less than B
A > B	A is greater than B
A <= B	A is less than or equal to B
A >= B	A is greater than or equal to B

- The "while" statement must end in a colon.
- If your program locks up, you may have it caught in an infinite loop.
- Commands that are part of a loop must be indented.
- The following operators will manipulate integer variables:

A += B	Add B to A. (Also works for strings.)
A - = B	Subtract B from A
A *= B	Multiply A by B. (Also works for strings.)
A /= B	Divide A by B
A **= B	Raise A to the power B

Program 6: Random Password Generator

If we can write a program that will generate random text, then what's to stop us from writing a program that will generate a random password?

The main difference between a random text generator and a random password generator is that we are working with individual letters instead of whole words, and we want to collect all the letters together before printing them. To collect our letters together we create a variable and designate it as an empty string of text. Then, inside a loop, we have the program randomly pick one letter and add it to our variable each time it goes through the loop. Once our word is as long as we want it to be, we exit the loop and print the variable. For this program, we will name that variable “password”.

Once the password is complete, the program should print it out. It is important that our “print password” statement comes after the loop; otherwise, the program will print each step it takes to build the password (“p”, “pa”, “pas”, “pass”, and so on). To designate the print statement as being separate from the loop, we “dedent” it. “Dedent” is the opposite of “indent” (bet you never thought about “indent” having an opposite). In other words, we signify that this line of code is outside the loop by aligning it with the “while” statement itself. This makes our code pretty, the logic of the code easier to follow, and is actually required to make the program work correctly.

It is also important that we create that password variable before we start our loop, outside the loop itself. If we put the password = “ ” code inside the loop, every time the loop started over the program would set our password variable to be an empty string, effectively erasing any letters that were previously added to our password. With this construction, no matter how many times we went through the loop, our program would always have only one letter in it. That's why we create the password variable outside the loop.

Code:

```
char_list = ["a", "b", "c", "d", "e", "f", "g"]
length_entry = raw_input("Password length: ")
pwd_length = int(length_entry)

counter = 1
password = ""
import random
while counter <= pwd_length:
        pwd_char = random.choice(char_list)
        password += pwd_char
        counter += 1
print password
```

Points to Remember:

- Code that follows a loop sequence, but is not part of the loop, must be “dedented”.
- If a loop is going to change a variable, create the variable outside the loop.

Program 7: if ... else tests

Now that we have a password, let's give our user a chance to test it. Let's add a feature to our program that lets the user type in the password, then checks if he typed it correctly.

Our password test needs to have three parts. First, we'll ask the user to enter the password. Next we'll compare their input with the password created earlier. And finally, depending on whether their input matches the password or not, we'll run either the “correct” code or the “incorrect” code. Getting the password from the user is just a matter of using another raw_input(), which we have done before. The new trick we need to learn is how to compare the two words and select with section of code to run. We do this using an “if ... else” test.

An “if” test is something like the “while” test we did in program five when we worked with loops. For the test we ask Python to compare two items (in this case two words) and execute the following indented code only if the test result is true. In our password testing program, we ask if the word the user entered (user_password_entry) matches the password created earlier in the program. The actual code says “if password = = user_password_entry:”. We use the “= =” operator to see if the two text strings are identical, including capitalization. If they are, the indented code beneath will be executed. If they are not identical, then the subsequent indented code will be skipped.

When the “if” statement is false (the two words don't match), and the associated “if” commands are skipped, we need to provide an alternative set of code to run. To do this, we create an “else” portion of code. Any code in the “else” section will be executed only when the “if” portion is skipped. This “else” code must be the first dedented line after the “if” section, and the commands associated with this “else” feature must be indented beneath it.

Code

```
# Create random password
char_list = ["a", "b", "c", "d", "e", "f", "g"]
pwd_length_entry = raw_input("Password length: ")
pwd_length = int(pwd_length_entry)

counter = 1
password = ""
import random
while counter <= pwd_length:
    pwd_char = random.choice(char_list)
    password += pwd_char
    counter += 1
print password

#Test password
print "----------Test Password ----------------"
user_password_entry = raw_input("Enter password: ")

if user_password_entry == password:
    print "Correct."

else:
    print "Incorrect password."
```

You'll notice in this code a couple lines that begin with #. The # symbol is used to mark a comment in the code. Comments are not executed by Python, they are just notes to help humans reading the code.

Points to Remember:

- A program can choose one of two sets of code to run based on an "if" test.
- An "if" test compares two or more values and runs the "if" code when the test is true.
- An "else" section of code is run only when the "if" test is false.
- The "else" command must be the first dedented line after the "if" section of code.
- Lines beginning with # are comments and will not be processed by Python.

Program 8: Manipulating Strings

In the real world, truly random passwords provide poor security. They are hard to remember, so users write the password on a post-it note by the computer, which is bad security. It's usually a better idea to generate passwords that are only partially random. So, to improve our password program, let's create a program that makes a password by taking the first two letters of a user's last name and adding a random number to the end.

To create such a program, first we want to get a random number. We could use random.choice() to randomly pick a number from a list, but then we'd have to create a list with hundreds of choices, which sounds like a lot of work. To save time, we'll use another function in the random module called randrange(). This function returns a random number within a range of numbers we specify. If we set the range from 100 to 999, random.randrange() will return a random three-digit number.

Once we have a three-digit number, we want to get the first two letters from the user's last name. To get this, we'll ask the user to type in their whole last name then make the program pick the first two letters. The raw_input() function will get the last name for us, so the trick is how to make the program pick only the first two letters. The easiest way to do this is to use a feature called string slicing.

In Python, every letter of a string is assigned an index number, which is kind of like an address for accessing that letter. In the word “Python”, the letter “P” is at index 0, “y” is at index 1, “t” is at index 2, and so on. Since each letter in a text string has an index address, we can use a special feature in Python to tell it which characters in the text string we want. To get only the first two letters from a text string assigned to the variable last_name, we say last_name[0:2]. When given this command, Python will get the characters from index zero up to index two, but will not actually get the character at index two. If we wanted the first three characters, we'd use the code last_name[0:3]. If we wanted everything except the first character, we'd say last_name[1:]. By not declaring the last index, Python just gets everything from the first index we declare to the end of the word. Using the index numbers, we can get any portion of a text string we want.

Now we have the first letters and a random number. It would seem we are done with our program, but there are two important details we need to consider. First, are the letters in our password going to be capitalized or not? If we want both letters capitalized we need to use a special function in the string module to convert the letters to uppercase. So, before adding our two letters to the password, we need to import the string module and use the string.upper() function. The string.upper() function capitalizes all the letters in a string. If we wanted both letters in lowercase, we would use the function string.lower(), which puts all letters in lowercase.

The second concern in creating our hybrid password is that Python cannot add an integer to a string, so it will crash if we just say “add this number to these letters”. The random.randrange() function gave us a random integer. If we want to add this integer to the letters of the last name, we need to convert the number into a string first. We've done something similar before when we converted a string to an integer using the int() function. Now, to convert the integer to a string we use the str() function. Once Python has a set of numerals in string format it can add them to the letters we yanked out of the user's last name.

And finally our program is ready. It's taken a lot of work, but for all this explanation, the code is really not that long.

Code:

```
# Generate  a three-digit random number
import random
number = random.randrange(100, 999)

# Get first two letters of last name
base_name = raw_input("Enter last name: ")
name_prefix = base_name[0:2]

#Put letters in uppercase
import string
name_prefix_upper = string.upper(name_prefix)

# New password
password = name_prefix_upper + str(number)
print password
```

Notice that we did not convert the number to a string until we were ready to add it to the password. We could have converted it to a string any time after we generated it, and technically speaking the variable "number" is still an integer, but converting it on this line saves us from having to create a new variable while showing that this in-line conversion is possible.

Points to Remember:

- The function random.randrange() returns a random integer in the given range.
- Each character in a string has an index address.
- A portion of a string can be obtained by "slicing" the string.
- To slice a string, place brackets [ ] after the variable name. Inside the brackets include the start index and the stop index, separated by a colon. Example: variable_name[0:2]
- If a stop index is not provided in a slice description, the end of the string is assumed.
- Python will not add an integer to a string.
- Use str() to convert an integer to a string.

Program 9: Reading from Text File

Having a user type in his last name to create a password will work great as long as we have only one or two users. When it comes time to create passwords for a few hundred students, however, we really don't want to sit in front of the computer all day typing in last names. It would be a lot easier if the program could just read the names from a text file. So, let's start working in that direction. In this program we will read text from a file that has only one word in it. You'll want to make such a file before running the program.

The main structure of the password generating program is going to stay the same. We still need to generate a random number, get a last name, find the first two letters of the last name and then add the random number to the end. The process that is going to change is how we get the name. In this program, we want to read from a file.

Reading from a file in Python is a three step process. Step one is opening the file and assigning the opened file to a variable name, or file handle. Step two is actually reading the information from the file located at the file handle. When we read a file we can read it in as either a single text string or as a list where each line of the text file is a separate item in the list. Since our file has only one word, we'll read it in as a text string. The last step for reading a text file is closing the file. We close the file as soon as we are done reading it. The text that has been read into the program will remain available throughout the program, but there is no need to keep the file open once we have finished reading it.

Our code, then, starts with opening the text file. Opening a file requires that we know the file name (including it's extension) and know what mode we want to open it in. The mode is a declaration of what we want to do with the file -- read from it(“r”), write to it(“w”), or add stuff to the end of the file(“a”). In this program we want to read the text in the file, so we will open it in read mode, “r”. To open the file we use the built-in function open(), and assign this function to a variable name (aka file handle) so we can manipulate the open file later. The open() function expects two pieces of information, the filename and the mode, and it requires that the information is presented to it in that order.

Once the file is open in read mode, we need to actually read the contents of the file using the read() function. The read() function takes the text from the file, imports it as a string, and stores it in a variable. Before it can read a file, the read() function has to know which file it is reading. To tell it the file name, we first type the variable name we used for opening the file, then put a period, and finally the read() function call. So, if we named our file handle “open_file”, then we would read the open file using the code open_file.read(). Python will then read the contents of the file and store the resulting string into our assigned variable.

Finally, once the contents of the file have been read into the program and we are done working with the actual file, we need to close the file. Amazingly enough, we use the close() function to close the file. The close function works just like the open() function, requiring the file handle followed by a period and then the close() function call.

Since our text file has only one name in it, the part of our that takes the name and creates a random password will not change.

Code:

```
        # Generate random number for password
        import random
        number = random.randrange(100, 999)

        # Open file and read last name
        filename = raw_input("Name of input file: ")
        file = open(filename, "r")
        file_text = file.read()
        file.close()

        #Get first two letters of last name
        name_prefix = file_text[0:2]

        #Put letters in uppercase
        import string
        name_prefix_upper = string.upper(name_prefix)

        # New password
        password = name_prefix_upper + str(number)
        print password
```

Points to Remember:

- There are three steps to reading a file:
  - Open file using open()
  - Read from file using read() or readlines()
  - Close file using close()
- When a file is opened it must be assigned to a file handle (variable name).
- Files may be opened in one of three different modes:
  - r   Read only
  - w   Write only
  - a   Append (add stuff to the end)
- The read() function reads text from a file into the program as a simple text string.
- To read from a from a file, the file handle must be identified Ex: file_handle.read().
- To close a file, the file handle must be identified Ex. file_handle.close().

Program 10: Readlines and Iterations

Now that we've gotten a program to read in one word from a file, let's get it to read lots of names from a file. For this program, we will assume that each name is on a separate line, as they would be if the contents of a database had been saved as a comma delimited text file. Before starting this program, create a text file with a different name on each line.

We need to make two fundamental changes to our previous code to get it to handle multiple names on multiple lines. First, instead of reading the text file into a single string, we will want to read it in as a list, with each name being a separate item in the list. The second change is to put our password creation code into a loop, so that for every name in our list the program develops a new password.

To read the text file in as a list we simply change the file.read() instruction to file.readlines(). The readlines function is a built-in function designed to read a text file with multiple lines and save each line as a new item in a list. So, if the file had the names “Adam, Barnabas, Caleb, Daniel” each on a separate line, readlines() would create a list like this: [“Adam\n”, “Barnabas\n”, “Caleb\n”, “Daniel\n”]. That goofy “\n” at the end of each name is the newline character, and for this program we can just ignore it.

With our list of names imported, the next thing to do is set up a special loop to create a password based on each name in the list. We could use a while loop, but that can get a little confusing when working with lists. A better solution is to use a “for” loop. A “for” loop automatically runs the loop code for each item in the list. It starts with the first item in the list, stores that name in the variable we designate, runs the code using that name, then goes back and gets the next name in the list. To set up the “for” loop we use the code “for name in name_list:”. The variable “name” is actually a variable we make up at this point. The “name” variable will hold the current name we are working on in the list and let us access it in the loop sequence.

So, with our “for” loop in place, the only thing left to do is move the password creation code into our for loop. Make sure each of these commands is indented correctly, and we're ready to go.

Code:

```
# Open file and read last names
filename = raw_input("Name of input file: ")
file = open(filename, "r")
names_list = file.readlines()
file.close()

#Create a password for each name in list
import random, string
for name in names_list:
    name_prefix = name[0:2]
    number = random.randrange(100, 999)
    name_prefix_upper = string.upper(name_prefix)
    password = name_prefix_upper + str(number)
    print password
```

Note that the code has the “import” commands grouped together before the loop. Since we only have to import the modules once, there is no need to have the import statement as part of the loop code.

Points to Remember:

- readlines() imports a text file into a list, with each line from the file as a different item in the list.
- The readlines() function also reads in the line break character.
- A "for" loop runs a set of code for each item in a list.
- When creating a "for" loop, create a variable name for the value to be accessed inside the loop sequence.

Program 11: Saving to a file

Our last program printed all these great passwords, but as soon as we exit Python, we've lost all of them. So, let's put some finishing touches on this program and get it to save our names and passwords together in a new comma-delimited text file. A comma-delimited text file is a special format that allows our data to be ported into a database, which will make our output even more useful.

To create this program we start with the same code from program ten. The two programs will create passwords in the same way, so that portion of the code will not change. The main change will be saving our passwords in a text file. Saving to a text file is similar to reading from a text file and involves three steps: opening the file in write or append mode, writing text to the file, and closing the file. If we open the file in append mode we can write one line at a time to the file, adding each new line as we create the new passwords. If we used write mode, we would have to write all the passwords at once, since write mode overwrites existing text by default. So, for our program we will want to open the file in append mode.

Once we have the file opened we are ready to write to it. It would be nice if we could make the Python program create a comma-delimited text file so that we could import this information back into a database. So, we'll set up each line of our new text file with the user's name, a comma, and then their new password, and we'll make sure each user is placed on a different line. Using that format, we'll be able to share with our database.

So, the first item on each line in the text file will be the user's name. Before saving the user's name to our file we need to edit it. When we read our name in from the text file it had a "\n" character at the end of every name. If we leave this character in the name and try to put the password right after it, the password will actually end up on a different line (that's what the new line character does, after all). So we need to cut that "\n" off the name using the string slicing technique. The code "name[:-1]" will give us all of the name except the last character, which in this case is our unwanted newline character.

With that pesky newline cut off the user name, all we need to do is add the whole line together. To do that we will create a new variable named "whole_line" and fill it with the name, a comma, the password, and a newline character. Once the whole line is assembled, we write it to the file. If we include this write command inside the loop that creates the new passwords, each password will be written to the line as it is created.

At the very end, once all the user names and passwords have been written to the file, we close the file.

And now, at last, we have created some semi-random passwords for all of our users and have a text file that contains both the user name and their password. Not bad for one day's work, especially since we were writing "Hello World" when we started.

Code:

```
# Open file and read last names
filename = raw_input("Name of input file: ")
file = open(filename, "r")
names_list = file.readlines()
file.close()

# Open a file for saving passwords
outfile_name = raw_input("Name of output file: ")
outfile = open(outfile_name, "a")

#Create a password for each name in list
import random, string
for name in names_list:
    number = random.randrange(100, 999)
    name_prefix = name[0:2]
    name_prefix_upper = string.upper(name_prefix)
    password = name_prefix_upper + str(number)
    whole_line = name[:-1] + ", " + password + "\n"
    outfile.write(whole_line)

#Close output file
outfile.close()
print "Finished"
```

Points to Remember:

- Open a text file in append mode if you will writing to the file more than once.
- A string index of [:-1] cuts off the last letter of a string.
- A comma-delimited text file has a comma separating the different data types.

Example 1: Batch Editing File Names

As an extra bonus for this training session, here is a script that will change the names of the files in a directory, so you can find all the ".htm" files and rename them ".html", or whatever name changes you need to make.

This program relies heavily on a module called "os". The os module contains functions that allow Python to work with the operating system, performing such tasks as listing files in a directory, erasing files, and renaming files. When using the os commands it is important to set the current working directory so you know where Python is erasing or renaming files. The os.chdir() function sets the working directory.

Another important tool in this script is the "try ... except" sequence. This is a special type of "if ... else" statement that is designed to catch errors. In our script we ask Python to open the directory designated by the user. If that directory does not exist (the user misspelled the directory name, for instance), Python would normally return an error code that will crash the program. By placing the os.chdir() function inside a "try" sequence, Python will try to enter the directory. If the directory does not exist, instead of just crashing the program, Python will jump to the statements in the "except" portion of the code and run those. The "try ... except" setup is a nice way to cleanly catch possible errors, instead of just letting a program die.

Code:

```
print "Program to change file names"

#Get desired directory
directory_entry = raw_input("What directory: ")

#if directory exists, move focus to that directory
import os
dir_found = "false"
while dir_found == "false":
    try:
        os.chdir(directory_entry)
        dir_found = "true"
    except:
        print "Invalid directory."
        directory_entry = raw_input("What directory: ")

#What to replace
old_text = raw_input("Search for: ")
new_text = raw_input("Replace with: ")

#rename each file in directory accordingly
file_list = os.listdir(directory_entry)
import string
for filename in file_list:
    if string.find(filename, old_text) >= 0:
        new_filename = string.replace(filename, old_text, new_text)
        os.rename(filename, new_filename)
        print "Renaming ", filename, " to ", new_filename
    else:
        pass

print "Finished"
```

Example 2: Batch-Edit Text Files in Directory

This is a program that finds all the occurrences of a text string in a set of text files and changes the text as indicated by the user. The program is case sensitive, and is limited to the files in a single directory (it does not drill down into subdirectories). This is another bonus program to play with. Enjoy.

Code:

```
#program to change file names
print "Batch Edit Text in Text Files"

#Get desired directory
directory_entry = raw_input("What directory: ")
import os
dir_found = "false"

#if directory exists, move focus to that directory
while dir_found == "false":
    try:
        os.chdir(directory_entry)
        dir_found = "true"
    except:
        print "Invalid directory."
        directory_entry = raw_input("What directory: ")

#What to replace
old_text = raw_input("Search for: ")
new_text = raw_input("Replace with: ")

#rename each file in directory accordingly
file_list = os.listdir(directory_entry)
import string, os.path
for filename in file_list:
    if os.path.isfile(filename):
        input_file = open(filename, "r")
        orig_file_text = input_file.read()
        input_file.close()
    else:
       continue

    if string.find(orig_file_text, old_text) >= 0:
        print "Editing ", filename
        new_file_text = string.replace(orig_file_text, old_text,new_text)
        output_file = open(filename, "w")
        output_file.write(new_file_text)
        output_file.close()
    else:
        pass

print "Finished"
```

Appendix A: Sample Code from Other Languages

Lest you think Python is too confusing, here is a look at the code required to get the computer to print “hello world” in a few other programming languages.

X86 Assembly “Hello World”

```
title   Hello World Program (hello.asm)
; This program displays "Hello, World!" dosseg
.model small
.stack 100h .data
hello_message db 'Hello, World!',0dh,0ah,'$' .code
main  proc
mov    ax,@data
mov    ds,ax       mov    ah,9
mov    dx,offset hello_message
int    21h       mov    ax,4C00h
int    21h
main  endp
end   main
```

C++ “Hello World”

```
#include <iostream.h>
main()
{
cout << "Hello World! ";
}
return 0
```

Java “Hello World”

```
class HelloWorldApp
  {
    public static void main(String[] args)
       {
        System.out.println("Hello World!");
        }
    }
```

Python

```
print “Hello world”
```

Appendix B:
Resources for Learning More

This is a highly subjective survey of books and resources available for learning Python.

Books:

- Learning Python by Mark Lutz, et al
  - *This text is a solid introduction to Python and a good reference to come back to. The book assumes some understanding of basic programming concepts, so it is not an introduction to programming as much as an introduction to Python. I have found this very useful and would probably make it one of my top recommendations.*

- Learn to Program Using Python by Alan Gauld
  - *This book is not exactly a Python reference; it is really an introduction to programming in general, using Python as the teaching language. I found this book very helpful for getting started, and a gentle introduction to many programming concepts, including some concepts found in other programming languages. It is a book you will likely "grow out of", but it's nice to have a starter book like this.*

- Python Programming on Win32 by Mark Hammond and Andy Robinson
  - *If you are thinking about writing scripts that will automate tasks you do in Microsoft programs, you'll want this book. It is one of the only references available for learning how to access the COM interfaces in Microsoft through Python, and it is written well.*

- Python How to Program by Deitel and Deitel
  - *This book provides by far the broadest coverage of Python, giving you a little information on a lot of different topics. The book includes chapters on creating graphical interfaces with Tkinter, working with databases, making CGI-BIN applications, using the Pygame module to make multimedia applications, working with XML, and a smorgasboard of other topics. The breadth of the coverage in this book is impressive, though most of the topics are not covered in any depth. I found the writing in this book to be less clear than other books, but it does offer a lot of information.*

- Python and Tkinter Programming by John Grayson
  - *This is one of the most complete references on how to create graphical user interfaces using the standard Tkinter module in Python. Documentation on Tkinter is hard to find, and the appendix to this text is the most comprehensive reference available. The text itself is not always helpful, often giving only cursory explanations and then referring you to the appendix, but it does have some well developed examples.*

- Teach Yourself Python in 24 Hours by Ivan Van Laningham
  - *This is probably the weakest of all the books listed. The examples are not practical (unless you really like working with ancient calendars), and the content just does not measure up to the other books.*

- Other Books I've Heard Good Things About
  - Core Python Programming by Wesley Chun
  - The Quick Python Book by Daryl Harms and Kenneth McDonald

Online:

- bembry.org
  - This is my personal site. It contains my teaching notes from the high school Python course I taught. The notes start with "what is a programming language" and move into how to create graphical user interfaces using Tkinter. I also have a number of useful links on the site, as well as a PDF version of these notes. My email address is there too, so feel free to contact me if you have questions.
  - URL: http://www.bembry.org

- Python Tutor mailing list
  - Tutor@python.org is a mailing list dedicated to folks asking, and answering, questions about how to program in Python. The questions range from the most simple to some rather challenging quandaries. Folks on the list usually post their code, so you get a chance to see how other programmers approach a certain problem. The site below includes a link to the archives, where you can search through past questions and answers.
  - URL: http://mail.python.org/mailman/listinfo/tutor

- Python.org
  - This is the official homepage for the Python programming language. The site contains a number of links for learning more about the language, downloading the current version, looking up language references, and more. Good resource in general.
  - URL: http://www.python.org

- Useless Python
  - Another rich resource, Useless Python contains a vast collection of sample codes (some of them truly worthless, as the name suggests), along with links to some really good resources and materials. Look around this site for a while, it's a lot of fun. (My favorite is "peggame.py", a program designed specifically to give you a solution to those stupid triangular peg games they have at restaurants).
  - URL: http://www.uselesspython.com

Appendix C:
Extending Python's Capabilities

Here are some more modules and extensions available for doing even more with Python.

py2exe

The py2exe code allows you to convert a Python program into a stand-alone executable, so it can be run on any Windows computer without having Python installed. This site includes the download and notes on how to use the program. The script can be a bit confusing to learn at first, but the code works well.
URL: http://starship.python.net/crew/theller/py2exe/

Pygame

Pygame is a set of Python modules for designing games and multimedia applications. This site also includes a number of neat games made entirely in Python.
URL: http://www.pygame.org/

Python Imaging Library

If you are interested in manipulating images with Python, you'll want to get this set of modules. These modules allow you to create programs that can edit images, including scaling, enhancing, and changing image formats.
URL: http://www.pythonware.com/products/pil/index.htm

Tkinter

Tkinter is a set of modules for designing graphical user interfaces for your Python programs. Tkinter is a standard part of Python, so there is nothing extra to download. There is not much documentation on Tkinter available, so figuring out how to use it can be a kind of treasure hunt. There are a few sites on the Internet that are helpful, and I also have some notes available on bembry.org that might be useful.

Win32 Extensions

The win32 extensions are modules that allow Python to access Microsoft COM interfaces. In other words, it's a way to mess with Windows and Microsoft applications through Python, doing such things as automating Windows NT/ 2000 administration tasks, extracting and manipulating information from an Excel worksheet, and more. If you use Windows and Python, this is a very useful set of modules to have in your tool chest.
URL: http://starship.python.net/crew/mhammond/

GNU Free Documentation License