brainSTEMSchool.com presents:

Python Coding Quickstart

An Coding eBook for Total NOOBs

Version 1.0
By B. Michael Tomaino

Dedication

For my family
My loving wife and children
You are my heartbeat

License

However I do ask that if you got this book from a friend, please drop me a note and let me know what you think:

Feedback - please answer my One-Question Survey

Table of Contents

Introduction

If you have never programmed before, your only programming experience was years ago and the dust has settled, or you know programming and want to see the nuts and bolts of Python in a quick, concise, and systematic introduction: **this book is for you.**

The purpose of this book is to give a brief but comprehensive overview of the very basics of programming using Python. *It's a quick start not a college course.* In the spirit of Tim Ferriss I'm trying to deconstruct the principles of programming and offer the Minimum Effective Dose to get you on the path to becoming a coder.

As a classroom educator I've taught a lot of young people how to program. Many took my course to fill a requirement and a few were very excited at the outset. However every student who put in the work found his or her ability to think differently. Code is really a problem solving art.

Programming is the art of using a very specific language to solve a problem. More precisely, coding is the way to tell a machine to solve the problem. But the machine can't speak your language - you have to speak its. Also it is stupid. Very stupid. But VERY obedient. It does exactly what you tell it every single time.

Throughout this book I'll use the word "code" and it's variants ("coding") synonymous with "program" and the like. Also the phrase "source code" usually means specifically the document that contains the instructions you program. We will also tend towards using the Python 3 branch rather than Python 2. That doesn't mean anything to you now but if anybody asks at least you'll have an answer.

Finally, be sure to check out my programming blog at: http://BrainSTEMSchool.com and let me know what you think of this book by answering my One-Question Survey.

Now grab a cup of Earl Grey tea, fresh roasted coffee, or a Dublin Dr. Pepper and let's get coding. Nobody said this would be easy.

Enjoy the book.

Mike

Chapter 0: 1, 2, 3 Roadmap

Most code books take much more than 3 steps to confound you!

Let's outline a concise step-by-step plan to go from point blank to calling yourself a programmer:

1. Get a programming environment
2. Learn the basic constructs
3. Use Python to solve problems

Chapter 1: Get a Programming Environment

I'll tell you where you can put your code

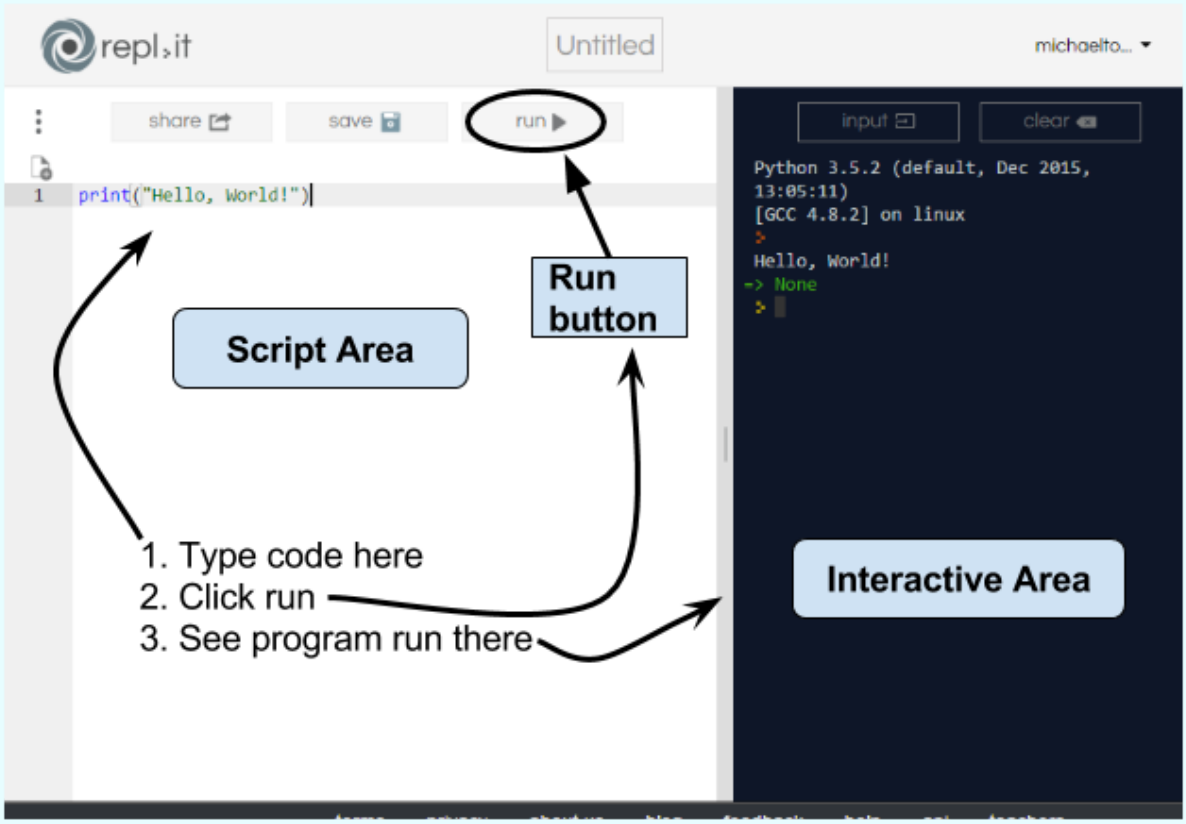
Online Python Interpreter

The easiest and quickest way to leap headfirst into programming with Python is with an online interpreter.

Repl.it

https://repl.it/languages/python3

Great for writing source code scripts then executing them. Has integration with Google sign-in.



Using the Repl.it for learning programming is easy:

1. Type the code in the script window

2. Click Run
3. Watch it go in the interactive window

Alternatively if you just want to see one command execute at a time:

1. Type the code into the interactive window
2. Hit enter

Your Very First Program

There's some law of nature that says that the first thing you code in any language has to be the *Hello, World!* program.

Let's try it.

1. **Type** the following line of code into the script window:

```
print("Hello, World!")
```

2. Click Run

Be sure to type it - don't copy and paste - it's better for learning.

You will see the following output:

```
Hello, World!
```

👁 SHARP EYES 👁

- Spot the color changes in the script editor - that's called **syntax highlighting** - it helps you to see what is what in the grammar or *syntax* of a program
- Notice exactly what prints in the interactive window: *only the text and punctuation*. No quotes, no parentheses, no print statement.
- Ignore that `=> None` It just tells you that the program is done

Comparing Scripting to Interactive Programming

Now type the same line of code into the interactive window and hit enter:

Source Code:	Output:
`print("Hello, World!")`	`Hello, World!`

So what's the advantage? Planning in advance. A script is a procedure that you plan and then execute.

For example, try the following:

```
print("Hello,")
print(" World!")
```

```
Hello,
 World!
```

Click run and observe the output:

It runs the whole script in **SEQUENCE** line by line in order.

Chapter 2: Learn the Basic Concepts

Just enough to get you started.

a. Output (already covered above!) ✔
b. Type
c. Variables
d. Math
e. User Input
f. Conditional Selection
g. Lists
h. Loops
i. Package and Built-in Functions
j. Your Own Functions

Type

Different kinds of data have a different "type" or classification in the system. It's important to keep them straight. Here is a brief summary of a few basic types in python:

Classification	Python Calls It	Description / Purpose
String	str	Text
Integer (Whole Numbers)	int	Whole numbers for counting and rounding
Floating Point	float	Decimal numbers
Boolean	bool	The value `True` or `False` for conditional statements

More on type later but for now you should know that they are not very compatible with one-another. Without doing something special you can't add a decimal to a whole number and get a whole number answer for instance.

Variables

Variables are containers like for sandwiches. They are not "placeholders" like in math.

You can name them almost anything you want but you should avoid using single letter variables.

3 VARIABLE NAMING RULES

- You may use letters, numbers, or the underscore _
- You may not start with a number
- You may not use one of Python's reserved words (see appendix)

… also keep in mind that they're cAsE SeNsItIvE ...

You create a variable by giving or **assigning** a value to it using a SINGLE EQUAL SIGN = such as the following code you can type in the interactive window:

```
name="Mike"
weight=180

print(name)
print("is")
print(weight)
print("pounds")
```

```
Mike
is
180
pounds
```

SHARP EYES

- Each print statement creates one line of output
- The **value** stored in the variable gets printed, not the variable itself.

If you want to have the message on one line, you have two choices:

Option 1: Tell the system not to "hit enter" at the end of the line

```
name="Mike"
weight=180

print(name, end="")
print("is", end="")
print(weight, end="")
```

```
Mikeis180pounds
```

```
print("pounds", end="")
```

Option 2a: Build a string by **concatenating** with the + sign

```
name="Mike"
weight=180

print(str(name)+"is"+str(weight)
+"pounds")
```

```
Mikeis180pounds
```

SHARP EYES

- `str(name)` **casts** or changes the type of value from **int** to **str** or string. More on that later.
- Ugly ugly ugly - there are no spaces! No problem - just add them in the string **literals**

Option 2b: Build a string with spaces

```
name="Mike"
weight=180

print(str(name)+" is "+
str(weight) + " pounds")
```

```
Mikeis180pounds
```

Option 2b: Build a string with spaces

```
name="Mike"
weight=180

print(str(name)+" is "+
str(weight) + " pounds")
```

```
Mikeis180pounds
```

Option 3: Force spaces by printing with commas ** **EASIEST TO USE** **

```
name="Mike"
weight=180

print(name,"is",weight,"pounds")
```

```
Mike is 180 pounds
```

CAUTION

In option 2b If you try to write a line without the str() casts, you'll get an error message. Try the following two variations and you'll see a demonstration of **mixing types without a cast**:

```
print("hello"+5)
print(5+"hello")
```

Math

If you're working with numbers or variables that contain numbers, you can do basic math operations following normal order of operations using these symbols:

+ adds
- subtracts
* multiplies
/ divides
x**y raises x to the exponent y
( ) work like they should
% modulo - gives remainder of division

Exercises:

Code and run the following examples and observe the results. Feel free to experiment with your own.

```
print(5+5)
print(10-5)
print(3*(4+5))
print(3*4+5)
print(2**4)
print(3*3)
print(2*4.5)
```

👁 SHARP EYES 👁

Look at the of the last two print statements. Shouldn't they be identical? It's not the same because Python treats the decimal numbers different than whole numbers.

1. Whole numbers: **int**
2. Decimals: **float**
3. If you combine the two in a math expression, the .

Using math in an output statement.

Example:

```
name="Mike"
weight=180

print(name,"is",weight-5,"pounds")
```

```
Mike is 175 pounds
```

Containing the result of an expression in a variable

Example:

```
name="Mike"
weight=180
newWeight=weight*2

print(name,"is",newWeight,"pounds"
)
```

```
Mike is 360 pounds
```

CODING IS NOT MATH

A statement like `newWeight=weight*2` is evaluated in 2 parts:

1. First the expression weight*2 is calculated
2. Second the result of that is **assigned** or contained within the other variable, newWeight

Because of that **1, 2 SEQUENCE**, a mathematically confusing statement like the following is perfectly legitimate (and in fact common):

```
x=x+1
```

Example:

```
x=1
print(x)
x=x+1
print(x)
x=x*5
print(x)
```

```
1
2
10
```

Input

Programs aren't very exciting unless the user has a way to interact.
Input in Python is very easy - you use an input() statement.

INPUT COMES IN STRINGS

If you want something other than text you'll need to cast it.

Input Example:

```
name=input("What is your name?")
print("Greetings,",name)
```

```
What is your name? Michael
Greetings, Michael
```

Input Example 2 - no problems here:

```
age=input("What is your age?")
print("Welcome to",age)
```

```
What is your age? 27
Welcome to 27
```

Input Example 3a - big trouble in little Python:

```
age=input("What is your age?")
age=age+5
print("Soon you'll be",age)
```

```
Traceback (most recent call last):
  File "python", line 2, in <module>
TypeError: Can't convert 'int' object
to str implicitly
```

There you have a problem you saw before - you can't mix types in that expression. You need a cast:

Input Example 3b - **FIXED**

```
age=input("What is your age?")
age=int(age)+5
print("Soon you'll be",age)
```

```
What is your age? 27
Soon you'll be 32
```

Selection

In programming, decisions are made based on true/false conditions. This true/false condition is called **boolean**.

Type 1: if this then do that. It's easy in code - in Python you put the condition after the word "if" then use a colon (:) . The "do that" action needs to be indented. When the indentation stops, the conditional action is over.

Simple Conditional Example with user input - **RUN TWICE WITH DIFFERENT USER INPUT**

```
age=input("What is your age?")

if int(age) >= 21:
    print("You may drink
alcohol")
print("Be safe")
```

```
What is your age? 27
You may drink alcohol
Be safe

What is your age? 18
Be safe
```

SHARP EYES

- The first print statement is indented and the second is not
  - Thus the first statement prints if the condition is met
- The variable **age** is a string and must be cast in order to be compared to a number

Type 2: if this then do that; otherwise do something else

Conditionals in if-else form

```
numb=input("Enter a number 1 to
10:")

if int(numb) == 7:
    print("Lucky Seven!")
else:
    print("Try again.")
```

```
Enter a number 1 to 10:8
Try again.

Enter a number 1 to 10:3
Try again.

Entera  number 1 to 10:7
Lucky Seven!
```

Type 3: multiple possible execution branches (use as many elif's as you like)

Conditionals in if-else form

```
numb=input("Enter a number 1 to
10:")
numb=int(numb)
if numb==7:
    print("Lucky Seven!")
elif numb==6 or numb==8:
    print("Close")
else:
    print("Try again.")
```

```
Enter a number 1 to 10:8
Close

Enter a number 1 to 10:2
Try again.

Entera  number 1 to 10:7
Lucky Seven!
```

In the third example see how the cast was done on its own line then reassigned to the same variable name? That makes the conditionals in the if structure much neater.

Another option would be to use the following input line:

```
numb=int(input("Enter a number 1 to 10:"))
```

Which makes the cast in the same statement as the input.

Conditional Expressions

You can use the following operators in conditional expressions:

Operator	Meaning
>	Greater Than
<	Less Than
>=	Greater Than or Equal To
<=	Less Than or Equal To
!=	Not Equal To
==	Equal To
and	Logical And
or	Logical Or
not	Logical Not

Lists

Lists are collections of variables all with the same name and can be told apart by their unique **index**

They have tremendous capability in Python but we are only going to cover the bare minimums here:

1. Use the brackets [ ] to designate a list
2. The list item indexes start with 0
3. Each element on a list is an individual variable
4. You can add to a list, change the elements of a list, and remove elements from a list

```
myList=[ 12, 43, 66, 38 ]

print( "Length: ", len(myList)
)

print(myList[0])
print(myList[1])
print(myList[2])
print(myList[3])

print(myList[4])
```

```
Length: 4
12
43
66
38
Traceback (most recent call last):
  File "python", line 11, in
<module>
IndexError: list index out of range
```

CAUTION

That last print statement is no-go! Remember, even though there are 4 elements in the list, **there is NO element number 4**

👁 SHARP EYES 👁

Did you spot the `len()` function? It counts the number of elements in a list - just put the name of the list between the parentheses.

```
print( "Length: ", len(myList) )
```

In this next code example we'll see how to **add to the end** of a list and **remove elements from the end**. For more information on one of Python's most powerful constructs, check this Python.org Lists Page out!

```
myList=[]
print( "Length: ", len(myList)
)

myList.append("Hello")
myList.append("World")
myList.append("!")
print( "Length: ", len(myList)
)

myList.pop()
print( "Length: ", len(myList)
)
print(myList[0])
print(myList[1])
```

```
Length:  0
Length:  3
Length:  2
Hello
World
```

FYI - Strings Can Act Like Lists

You can access individual elements (characters) of a string as if the string were a list. For example:

```
myString="Hello, World!"
print(myString[4])
```

Would output the letter 'o'

Loops

Loops make things happen over and over again. They're great for counting or for waiting for something to happen.

There are two kinds of loop: for and while

"for" Loops

These are great for lists. Use them to perform an action once for each item on a list.

Example: for loop iterating through each item on a list

```
myList=[4, 8, 15, 16, 23, 42]
for number in myList:
    print(number)
```

```
4
8
15
16
23
42
```

In the previous example you see how the print statement prints a single variable: `number`. However each time the loop iterates that variable has a different value-one for each element of the list.

The very useful `range()` function

The `range()` function creates a "range" of numbers which is like an automatic list and can be cast into a list.

A great use of for loops is with the `range()` function. We will see more package functions later in the book. For now you can see a simple demonstration:

1. Create a range and assign it to a variable
2. Cast the range to a list and print its contents
3. Iterate through the range printing each number and a message

Example: iterating through a range

```
myRange=range(5)
print( list(myRange) )
for number in myRange:
    print(number, "potatoes.")
```

```
[0, 1, 2, 3, 4]
0 potatoes.
1 potatoes.
2 potatoes.
3 potatoes.
4 potatoes.
```

SHARP EYES

"1 potatoes"? We have to fix this program
let's add a conditional statement that says:

if the number is 1
    then print "potato"
otherwise
    print "potatoes"

Example: iterating through a range with a conditional

```
myRange=range(5)
print( list(myRange) )
for number in myRange:
  if number==1:
      print(number, "potato.")
  else:
      print(number,
"potatoes.")
```

```
[0, 1, 2, 3, 4]
0 potatoes.
1 potato.
2 potatoes.
3 potatoes.
4 potatoes.
```

"while" loops

These are great for waiting for something to happen. One good use is getting user input:

Example: while loop taking user input and adding items to a list - BLANK LINE EXITS LOOP

```
shoppingList=[]
item=input("Enter list items:")
while(item):
  shoppingList.append(item)
  item=input("Enter list items:")

print("Items on list:")
print(shoppingList)
```

```
Enter list items: eggs
Enter list items: cheese
Enter list items: tp
Enter list items:
Items on list:
['eggs', 'cheese', 'tp']
```

What causes this while loop to exit?

When the user just hits enter, an empty string is assigned to the variable `item`. The while loop is asking if the item has any content - it doesn't - so the loop ends.

Here's another example that asks a user to guess a number:

Example: while loop until a certain value is found

```
magicNumber=37
guess=input("Guess the magic number:")
While int(guess)!=magicNumber:
  guess=input("No way! Guess again:")

print("You got it! Good guess.")
```

```
Guess the magic number: 123
No way! Guess again: 99
No way! Guess again: -15
No way! Guess again: 37
You got it! Good guess.
```

👁 SHARP EYES 👁

In both these examples, there is an input() statement before the loop AND inside of the loop.

This is called "priming the pump" because we need the while condition to be true in order to enter the loop for the first time. Once we're in the loop we have to give Python an opportunity to exit the loop.

Package and Built-in Functions

A **function** is a named procedure - a predefined script or mini-program within your code. They can be tremendously useful and Python has an incredible amount of helpful functions and packages.

Built-In Functions

A complete list of built-in functions is listed here: https://docs.python.org/3/library/functions.html

The built-in functions are always available whereas package functions have to be imported to be available.

Here are some useful built-in functions to get you started (we've also covered print(), range(), input(), int() and str() above):

Note: for some of the following we will use the list: `myList=[-3,2,0,5,10]`

Function	Description	Example	Result
abs()	Absolute Value	`print(abs(-7))`	7
len()	Length of string or list	`len(myList)`	5
max()	Maximum value in list	`max(myList)`	10
min()	Minimum value in list	`min(myList)`	-3
round()	Rounds a decimal number	`round(2.5)`	2
sum()	Calculates sum of numbers in a list	`sum(myList)`	14

To use them, just put them ANYWHERE you want the result to appear. Check out this example that builds on a previous example:

Example: Evaluating user input with built-in functions

```
numList=[]
item=input("Number or blank line:")
while(item):
  numList.append(int(item))
  item=input("Number or blank line:")

myLen=len(numList)
myMax=max(numList)
myMin=min(numList)
mySum=sum(numList)

print("Length:",myLen)
print("Max:",myMax)
print("Min:",myMin)
print("Sum:",mySum)
```

```
Number or blank line: 4
Number or blank line: 8
Number or blank line: 15
Number or blank line: 16
Number or blank line: 23
Number or blank line: 42
Number or blank line:
Length: 6
Max: 42
Min: 4
Sum: 108
```

Package Functions

It is WELL beyond the scope of this introduction to go into any productive depth on the vast amount of package functions and tools Python offers. Furthermore you can make use of a host of 3rd party packages available online.

If you want to explore the package functions that come with Python, check out Python.org's Standard Library Documentation Page.

Here is a highlighting of some useful functions in section 9.2 math - Mathematical Functions:

How to use package functions:

1. You must **import** the packages
2. You must specify the package AND function with a dot such as: **`math.floor(3.14)`**

Aren't This Kind Of
"dot" Functions Called "Methods"?

Yes. But I don't care about that distinction.

Here are some useful math package functions to get you started:

Note: for some of the following we will use the list: **`myList=[-3,2,0,5,10]`**

Function	Description	Example	Result
math.ceil(x)	Ceiling - whole number greater than	`math.ceil(3.14)`	4
math.floor(x)	Floor - whole number less than	`math.floor(3.14)`	3
math.factorial(x)	Returns x factorial	`math.factorial(4)`	24
math.gcd(a, b)	Greatest common divisor of the integers a and b	`math.gcd(124,256)`	4

DON'T FORGET

Don't forget to import the package before you use its functions:

```
import math
```

```
import math
num1=int(input("Enter the first number:"))
num2=int(input("Enter the second number:"))
print("The greatest common divisor the two numbers is:",math.gcd(num1,num2))
```

```
Enter the first number: 234
Enter the second number: 456
The greatest common divisor of the two numbers is: 6
```

SHARP EYES

This time you wrapped the int() cast around the input() function. The **return value** of the input function is the **argument** sent to the int() cast. The int() cast returns an int that is returned and assigned to the variables num1 and num2

Your Own Functions

To make your life easier, you can write your own functions. You shouldn't usually write the same nontrivial code twice if you can put it in a subroutine and just tell the system to repeat that routine.

As a useless example for demonstration purposes, it you want the to define a simple procedure such as saying hello, do this:

Example of creating and executing a simple procedure - printing a line.

```
def sayHi():
     print("Hello")

sayHi()
sayHi()
sayHi()
```

```
Hello
Hello
Hello
```

Two Aspects of Using Functions

- **Definitions:** These are the instructions for a function to follow when it's called
- **Call:** This is when you invoke, execute, or perform the function's definition

Function definitions start with the **`def`** followed by the function's name and then use the parentheses and colon. The **body** of the function is indented.

You really need to mind only 3 things when working with functions:

1. **Parameters/Arguments:** What info to give the function, if any
2. **Body**: The mini-program or procedure the function follow
3. **Return Value**: What data, if any, the function produces and leaves behind

Let's write a function to perform the simple **"ROT13"** encoding system for simple text encryption.

More on ROT13 here at Wikipedia but all you need to know is that it replaces each letter in the alphabet with the one 13 places away so the same code can be used to encrypt and decrypt text.

There are certainly *quite a few ways* to approach this problem but we're going to create two strings of letters and use corresponding indexes to make the translations. For simplicity's sake we'll only concern ourselves with lowercase letters. Observe:

Example of Defined Function - also with line numbers for easy reference

```
1:  def rot13(oneCharacter):
2:        fromAlphabet="abcdefghijklmnopqrstuvwxyz"
3:        toAlphabet=  "nopqrstuvwxyzabcdefghijklm"
4:        newIndex=fromAlphabet.find(oneCharacter)
5:        if newIndex != -1:
6:              newLetter=toAlphabet[newIndex]
7:        else:
8:              newLetter=oneCharacter
9:
10:       return newLetter
11:
12:
13: message=input("Enter message:")
14:
15: cryptoMessage=""
16: for character in message:
17:       cryptoMessage=cryptoMessage+rot13(character)
18:
19: print("Encrypted:",cryptoMessage)
20:
21: newMessage=""
22: for character in cryptoMessage:
23:       newMessage=newMessage+rot13(character)
24:
25: print("Decrypted:",newMessage)
```

```
Enter message: hello, world!
Encrypted: uryyb, jbeyq!
Decrypted: hello, world!
```

Let's break it down:

1:	Declare the name of the function, *rot13*, and the parameter, *oneCharacter*
2:	Defines the *fromAlphabet* string to contain original alphabet characters in order
3:	Defines the *toAlphabet* string to contain the cryptocharacters for the translation
4:	Finds the index of the original character in the *fromAlphabet* string
5:	If the character IS in the *fromAlphabet* string
6:	Then assign the corresponding character from the *toAlphabet* string
7:	Otherwise
8:	Retain the original character (such as for spaces and punctuation)
9:	
10:	Return the newly cryptographically converted letter
11:	

```
12:
13:  Get user input and assign it to the variable, message
14:
15:  Initiate a blank string variable named message
16:  For each character in the string message that the user just input
17:  Apply the ROT13 algorithm to the character and append it to the cryptomessage string
18:
19:  Output the encrypted message
20:
21:  Lines 21-25 perform the same as lines 15-19 in order to decrypt the message.
22:
23:
24:
25:
```

Chapter 3: Use Python to Solve Problems

Problems And Solutions

Make A Frame

Problem: Ask the user for a short message then print it with a frame around it

Solution:

```
1:  message=input("Enter a message:")
2:  starLine="**"
3:  for character in message:
4:        starLine=starLine+"*"
5:  starLine=starLine+"**"
6:  print(starLine)
7:  print("* "+message+" *")
8:  print(starLine)
```

Make A Frame - Vertical Edition

Problem: Ask the user for a message then print it one word per line with a frame around it

Solution: (uses str.split() function - more info on string methods here at python.org)

```
1:  message=input("Enter a message:")
2:  messageList=message.split()
3:  maxLen=0
4:
5:  for word in messageList:
6:        if len(word) > maxLen:
7:              maxLen=len(word)
8:  starLine="**"
9:
10: for star in range(maxLen):
11:       starLine=starLine+"*"
12: starLine=starLine+"**"
13: print(starLine)
14:
15: for word in messageList:
16:       outLine="* "+word
17:       for space in range(maxLen-len(word)):
18:             outLine=outLine+" "
19:       outLine=outLine+" *"
20:       print(outLine)
21:
22: print(starLine)
```

Pig Latin

Problem: Write program that translates a text to Pig Latin and back. Pig Latin is made by taking the first letter of every word, moving it to the end of the word and adding 'ay'. "How now brown cow" becomes "Owhay ownay rownbay owcay".
Solution: (uses `str.join()` function - more info on string methods here at python.org **.** Also uses "slicing" to get a substring. String slicing is explained here and list slicing is here on the same page.**)**

```
1:  message=input("Enter a message:")
2:  messageList=message.split()
3:  newMessageList=[]
4:
5:  for word in messageList:
6:        newword=word[1:]+word[0].lower()+"ay"
7:        newMessageList.append(newword)
8:
9:  newMessage=' '.join(newMessageList)
10:
11: print(newMessage)
```

Problems for You To Solve (on your own)

Extend the ROT13 program to accommodate case sensitivity

Extend the Pig Latin program to handle punctuation at the end of a sentence

Extend the Pig Latin program to maintain capitalization

Write a program to convert between English and Moorse Code

Chapter 4: Conspicuously Absent

I left a few things out of this text - mostly for the sake of succinctness. Remember, this is a quickstart guide and not a comprehensive course. Here are a few things worth mentioning, however:

Comments

Comments are programmer's notes within the code. You can think of them as sticky notes within code. They are completely ignored by the interpreter. In a given line of code, anything after the # symbol will be ignored by Python. Non-coders have a hard time understanding the point of comments and seasoned programmers cannot live without them. By the way, coders like to call the # "octothorpe" or "pound sign" as opposed to "hashtag" which is a nomenclature that has gained popularity in recent years due to social networking trends.

Boolean Logic and Combining Control Expressions

In if or while statements, as well as in other contexts, it's sometimes useful to have logical expressions or complex evaluations. The three boolean operators: and, or, not; went largely without explanation in this document. They are extremely valuable and important and are

in

in is a very useful operator for searching and finding out if a value exists in a list. For instance in the following code you can use in with an if statement:

```
names=["Mike","Ben","Michelle"]
if "Mike" in names:
     print("He's here!")
```

```
He's here!
```

Python is full of these shortcuts and useful features. Explore more comprehensive guides and documentations to get a better mastery. Python is the kind of language that you can never stop learning better ways to do things. It's vast but every step of the way what you learn is immediately applicable.

Appendix A: Reserved Words / Keywords

```
and       del       from      not       while
as        elif      global    or        with
assert    else      if        pass      yield
break     except    import    print
class     exec      in        raise
continue  finally   is        return
def       for       lambda    try
```