Chapter 2

Types, Variables, Simple I/O:

The Useless Trivia Program

Using Quotes with Strings

- An example of a string, "Game Over." But strings can become much longer and more complex.

- Using quotes can help you to create strings to accomplish all of this.

game_over2.py

```
# Game Over - Version 2
# Demonstrates the use of quotes in strings

print("Program 'Game Over' 2.0")

print("Same", "message", "as before")

print("Just",
      "a bit",
      "bigger")

print("Here", end=" ")
print("it is...")

print(
      """
```

```
         _______        ____      ___  ____  _______
        /  ____|       /   |     /   |/   | |  ____|
        | |           / /| |    / /|   /| | | | |___
        | |    _    /  _   |   / / |__/| | |  ___|
        | |___| |  / /   | |  / /       | | | |____
        \_______/ /_/    |_| /_/        |_| |_______|

         _______   _     _    _______   _______
        /   _  \  | |   / /  |  ____|   |  _  \
        |  | |  | | |  / /   | |___     | |_| |
        |  | |  | | | / /    |  ___|    |  _  /
        |  |_|  | | |/ / /     | |____    | | \ \
        \_______/ |____/      |_______| |_|  \_\

        """
    )

input("\n\nPress the enter key to exit.")
```

The program shows that it's pretty simple to present text in different ways by using quotes.

Using Quotes Inside Strings

- You can use either a pair of single (' ') or double quotes (" ") to create string values.

- 'Game Over' represents the same string as "Game Over" .

- the 1st appearance of a string in the program

print "Program 'Game Over' 2.0"

uses both kinds of quotes.

- Only the single quotes show up, because they are part of the string. But the double quotes are not part of the string.

- If you use a pair of double quotes to “bookend” your string, you can use as many single quotes inside the string as you want. And, if you surround your string with a pair of single quotes, you can use as many double quotes inside the string as you like.

- Once you've used one kind of quote as bookends for your string, you can't use that type of quote inside your string.

"With the words, 'Houston, we have a problem.', Jim Lovell became one of our most famous astronauts." √

"With the words, "Houston, we have a problem.", Jim Lovell became one of our most famous astronauts." ✗

Printing Multiple Values

- Print multiple values with a single call to print()—just list multiple argument values, separated by commas.

- Print multiple values with the line

print("Same", "message", "as before")

passing 3 arguments to the function: "Same" , "message" , and "as before.", to display Same message as before.

- Each value is printed with a **space** as a separator. This is the default behavior of the print() function.

- **print ("Just",**
  "a bit",
  "bigger")

form a single statement that prints the one line of text

Just a bit bigger

Specifying a Final String to Print

- By default, the print() function prints a newline character as a final value.

- But you have the ability specify your own final string to be printed.

- To specify that a space be the final character printed instead of the newline

```
print("Here", end=" ")
print("it is...")
```

To prints the text “Here it is...” all on one line.

- accomplish this by passing a space to the end parameter of the print() function with the code **end=" "** .

- You can specify other string to be printed as the final value.

Creating Triple-Quoted Strings

- A *triple-quoted string* is a string enclosed by a pair of 3 quotes in a row. It doesn't matter which kind of quotes you use, as long as you bookend with the same type.

```
"""
.........
"""
```

- Triple-quoted strings can span multiple lines. They print on the screen exactly the way you type them.

Using Escape Sequences with Strings

- *Escape sequences* allow you to put special characters into your strings.

- The escape sequences are made up of 2 characters: a backslash followed by another character.

Introducing the Fancy Credits Program

fancy_credits.py

```
# Fancy Credits
# Demonstrates escape sequences

print("\t\t\tFancy Credits")

print("\t\t\t \\ \\ \\ \\ \\ \\ \\")
print("\t\t\t\tby")
print("\t\t\tMichael Dawson")
print("\t\t\t \\ \\ \\ \\ \\ \\ \\")

print("\nSpecial thanks goes out to:")
print("My hair stylist, Henry \'The Great,\' who never
says \"can\'t.\"")

# sound the system bell
print("\a")

input("\n\nPress the enter key to exit.")
```

Moving Forward a Tab Stop

- To set some text off from the left margin where it normally prints, you could use the Tab key.

- With strings, you can use the escape sequence for a tab, **\t**.

```
print("\t\t\tFancy Credits")
```

- When the program prints the string, it prints 3 tabs and then Fancy Credits .

- Tab sequences are good for setting off text, but they’re also perfect for arranging text into columns.

Printing a Backslash

- To print a backslash, just use 2 backslashes in a row, \\.

- Each of the following lines prints 3 tabs followed by 7 backslashes, separated by spaces:

```
print("\t\t\t \\ \\ \\ \\ \\ \\ \\")
print("\t\t\t \\ \\ \\ \\ \\ \\ \\")
```

Inserting a Newline

- The newline sequence is represented by **\n** .

- By using this sequence, you can insert a newline character into your strings for a blank line.

- You can use a newline right at the beginning of a string to separate it from the text last printed.

print("\nSpecial thanks goes out to:")

The computer prints a blank line, then prints Special thanks goes out to: .

Inserting a Quote

- To insert a quote into a string, use the sequence \' for a single quote and \" for a double quote.

print("My hair stylist, Henry \'The Great,\' who never says \"can\'t.\"")

* \'The Great\' prints as 'The Great'

* Each \' sequence is printed as a single quote

* \"can\'t\" prints as "can't"

* Both \" sequences print as double quotes

* The lone \' sequence prints as a single quote

Sounding the System Bell

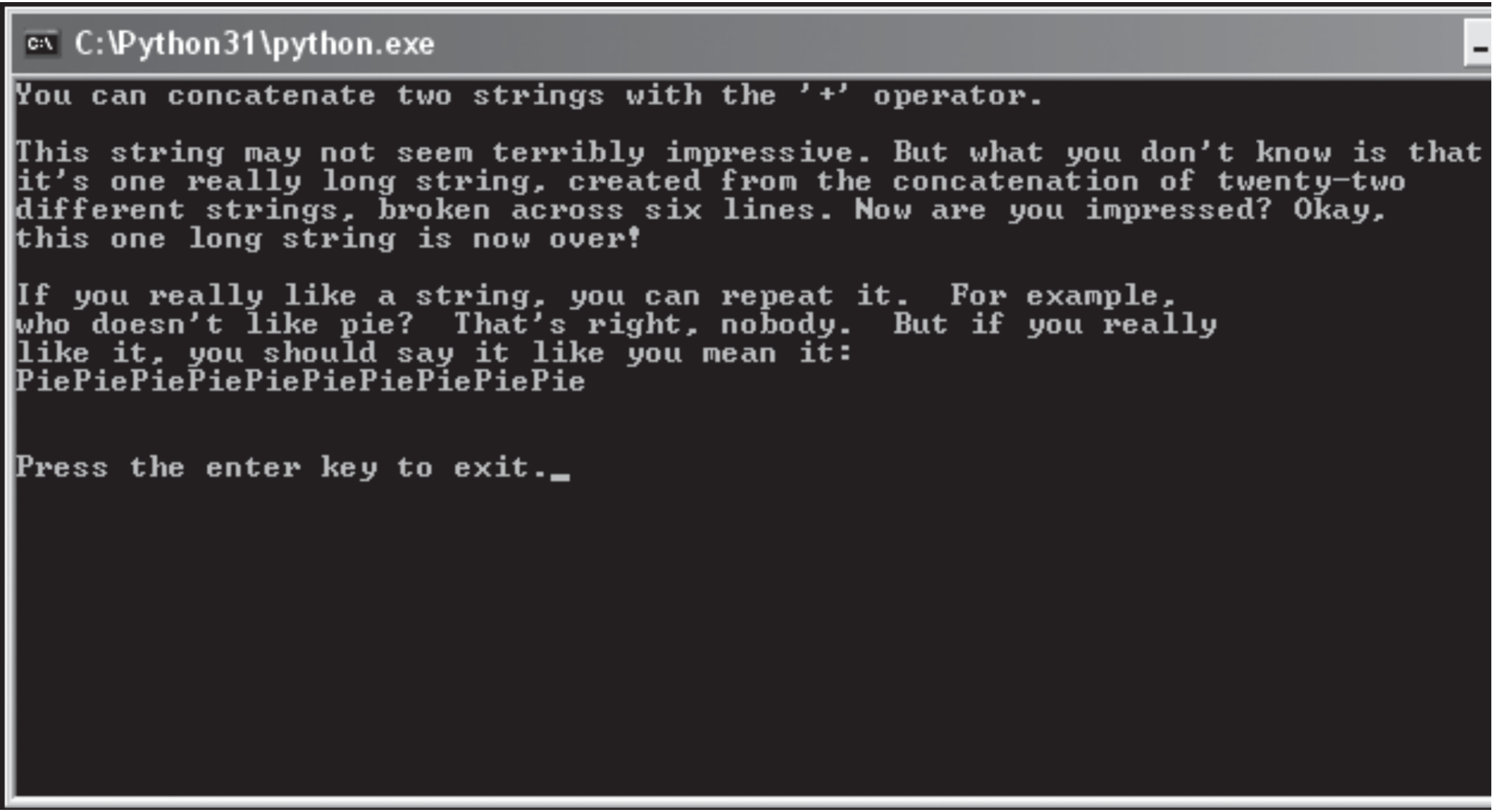
- **print("\a")** sounds the system bell of your computer.

- It does this through the escape sequence, **\a**, which represents the system bell character. Every time you print it, the bell rings.

- Some escape sequences only work as advertised if you run your program directly from the operating system and not through IDLE. **\a** is a good example.

Sequence	Description
\\	Backslash. Prints one backslash.
\'	Single quote. Prints a single quote.
\"	Double quote. Prints a double quote.
\a	Bell. Sounds the system bell.
\n	Newline. Moves cursor to beginning of next line.
\t	Horizontal tab. Moves cursor forward one tab stop.

Concatenating and Repeating Strings



silly_strings.py.

```
# Silly Strings
# Demonstrates string concatenation and repetition

print("You can concatenate two " + "strings with the '+' operator.")

print("\nThis string " + "may not " + "seem terr" \
+ "ibly impressive. " + "But what " + "you don't know" \
+ " is that\n" + "it's one real" + "l" + "y" \
+ " long string, created from the concatenation " + "of " \
+ "twenty-two\n" + "different strings, broken across " \
+ "six lines." + " Now are you" + " impressed? " + "Okay,\n" \
+ "this " + "one " + "long" + " string is now over!")

print("\nIf you really like a string, you can repeat it. For example,")
print("who doesn't like pie? That's right, nobody. But if you really")
print("like it, you should say it like you mean it:")
print("Pie" * 10)

input("\n\nPress the enter key to exit.")
```

Creating the Initial Comments

- Although comments don’t have any effect while the program runs, they are an important part of every project.

- Experienced programmers also use the initial comments area to describe any modifications they make to code over time.

- This provides a history of the program right up front. This practice is especially helpful when several programmers have their hands on the same code.

Concatenating Strings

- *Concatenating* strings means joining them together to create a whole new string.

print("You can concatenate two " + "strings with the '+' operator.")

- The + operator joins the 2 strings, "You can concatenate two " and "strings with the '+' operator." , together to form a new, larger string.

- When you join 2 strings, their exact values are fused together, with *no* space or separator character inserted between them.

- So, if you were to join the 2 strings "cup" and "cake", you'd end up with "cupcake" and not "cup cake".

- You can concatenate strings as long as you want.

Using the Line Continuation Character

- You can stretch a single statement across multiple lines.
- All you have to do is use the line-continuation character, \.
- Put it anywhere (but not inside a string) to continue your statement on the next line.
- The computer will act as if it sees one long line of code.

Repeating Strings

print("Pie" * 10)

creates a new string, "PiePiePiePiePiePiePiePiePiePie" , and prints it out. That's the string "Pie" repeated 10 times.

- To repeat a string, just put the string and number of repetitions together with the repetition operator, *.

Introducing the Word Problems Program

```
C:\Python31\python.exe
If a 2000 pound pregnant hippo gives birth to a 100 pound calf,
but then eats 50 pounds of food, how much does she weigh?
Press the enter key to find out.
2000 - 100 + 50 = 1950

If an adventurer returns from a successful quest and buys each of
6 companions 3 bottles of ale, how many bottles are purchased?
Press the enter key to find out.
6 * 3 = 18

If a restaurant check comes to 19 dollars with tip, and you and
your friends split it evenly 4 ways, how much do you each throw in?
Press the enter key to find out.
19 / 4 = 4.75

If a group of 4 pirates finds a chest full of 107 gold coins, and
they divide the booty evenly, how many whole coins does each get?
Press the enter key to find out.
107 // 4 = 26

If that same group of 4 pirates evenly divides the chest full
of 107 gold coins, how many coins are left over?
Press the enter key to find out.
107 % 4 = 3


Press the enter key to exit._
```

word_problems.py

```
# Word Problems
# Demonstrates numbers and math

print("If a 2000 pound pregnant hippo gives birth to a 100 pound calf,")
print("but then eats 50 pounds of food, how much does she weigh?")
input("Press the enter key to find out.")
print("2000 - 100 + 50 =", 2000 - 100 + 50)

print("\nIf an adventurer returns from a successful quest and buys each of")
print("6 companions 3 bottles of ale, how many bottles are purchased?")
input("Press the enter key to find out.")
print("6 * 3 =", 6 * 3)
```

```
print("\nIf a restaurant check comes to 19 dollars with tip,
and you and")
print("your friends split it evenly 4 ways, how much do you
each throw in?")
input("Press the enter key to find out.")
print("19 / 4 =", 19 / 4)

print("\nIf a group of 4 pirates finds a chest full of 107 gold
coins, and")
print("they divide the booty evenly, how many whole coins
does each get?")
input("Press the enter key to find out.")
print("107 // 4 =", 107 // 4)

print("\nIf that same group of 4 pirates evenly divides the
chest full")
print("of 107 gold coins, how many coins are left over?")
input("Press the enter key to find out.")
print("107 % 4 =", 107 % 4)

input("\n\nPress the enter key to exit.")
```

Understanding Numeric Types

- Python allows programmers to use several different types of numbers.

- The 2 types used in this program, and probably the most common, are *integers* and *floating-point numbers* (or *floats*).

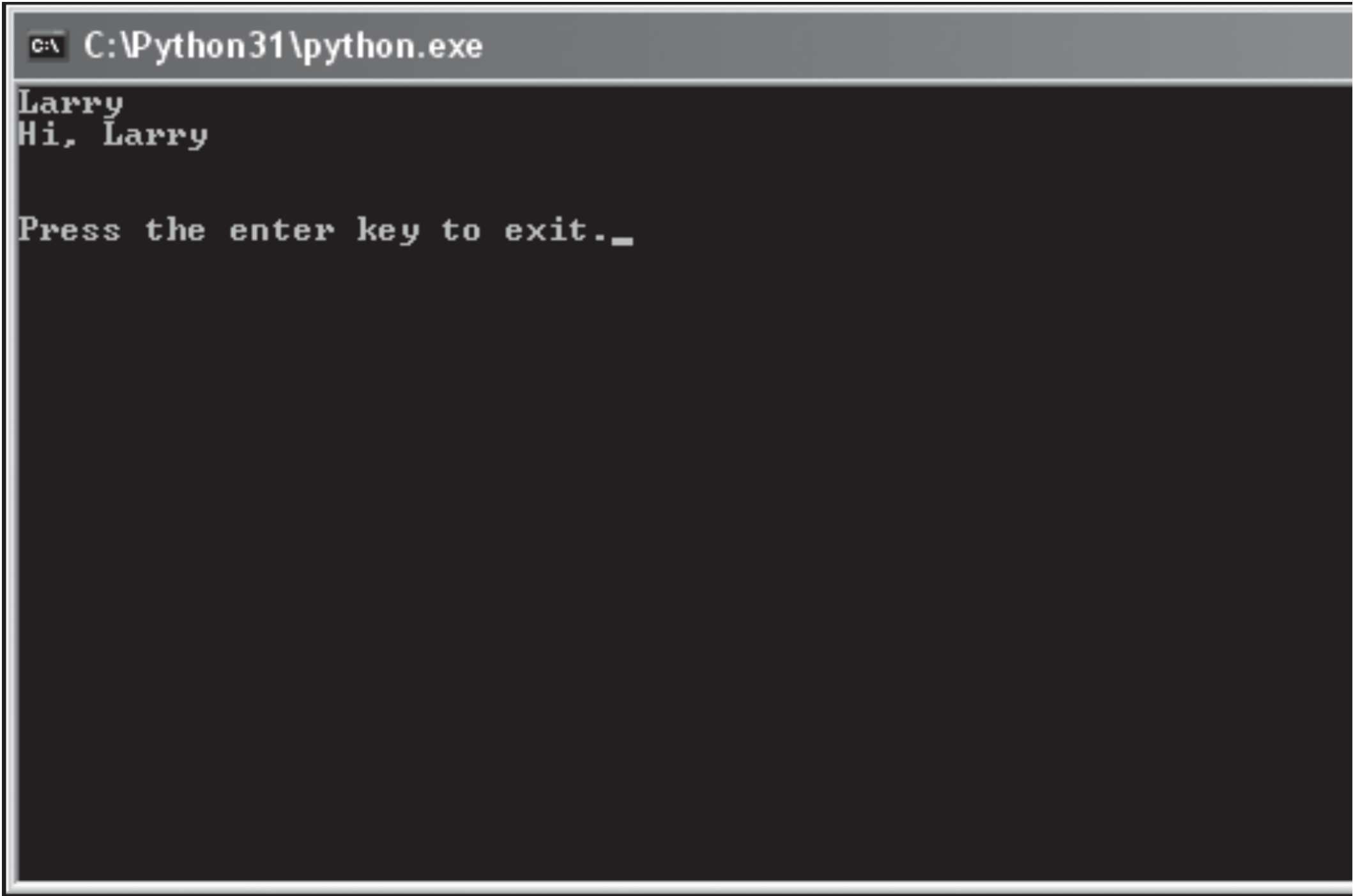
- Integers are whole numbers—numbers with no fractional part, eg, 1, 27, -100, 0, etc.

- Floats are numbers with a decimal point, like 2.376, - 99.1, and 1.0.

Using Mathematical Operators

Operator	Description	Example	Evaluates To
+	Addition	7 + 3	10
-	Subtraction	7 - 3	4
*	Multiplication	7 * 3	21
/	Division (True)	7 / 3	2.3333333333333335
//	Division (Integer)	7 // 3	2
%	Modulus	7 % 3	1

Introducing the Greeter Program



greeter.py

```
# Greeter
# Demonstrates the use of a variable

name = "Larry"

print(name)

print("Hi,", name)

input("\n\nPress the enter key to exit.")
```

Creating Variables

- A *variable* provides a way to label and access information.

- Before you use a variable, you have to create it,

name = "Larry"

- This line is called an *assignment statement*. It creates a variable called name and assigns it a value so that it references the string "Larry" .

- In general, assignment statements assign a value to a variable.

Using Variables

- The convenience of a variable is that it can be used just like the value to which it refers.

- print(name) prints the string "Larry" just like the statement print("Larry") does.

- print("Hi,", name) prints the string "Hi," followed by a space, followed by "Larry" .

Naming Variables

- There are only a few rules that you have to follow to create legal variable names.

- Create an illegal one and Python will let you know about it with an error.

- The 2 most important rules:

1. A variable name can contain only numbers, letters, and underscores.

2. A variable name can’t start with a number.

Guidelines for good variable names

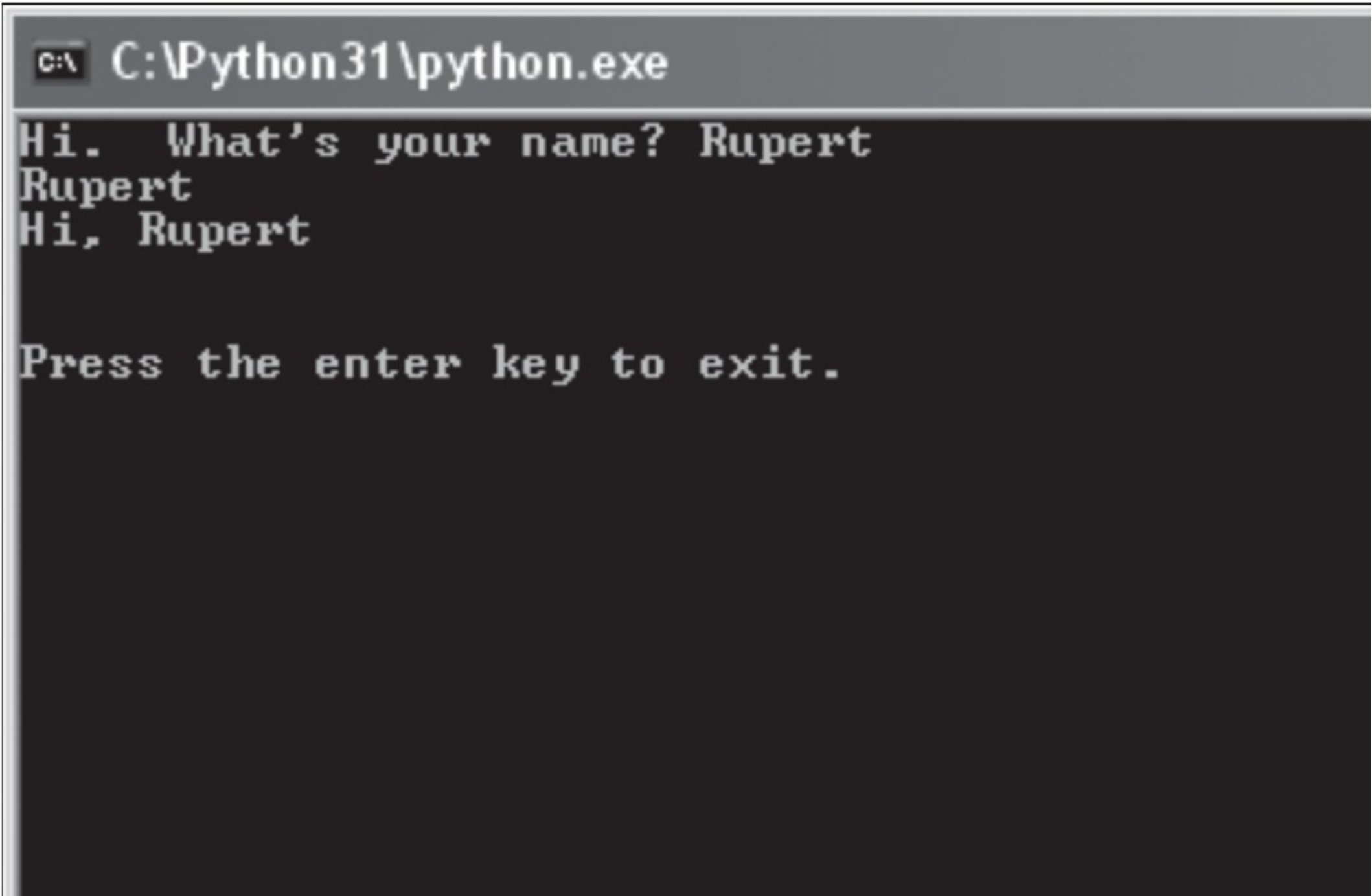
- **Choose descriptive names**: Use score instead of s .

- **Be consistent**: It's not important what the style of naming is, as long as you're consistent.

- **Follow the traditions of the language**: Avoid using an underscore as the 1st character of variable names. Names starting with an underscore have special meaning in Python.

- **Keep the length in check**: try to keep your variable names under 15 characters

Introducing the Personal Greeter Program



personal_greeter.py

```
# Personal Greeter
# Demonstrates getting user input

name = input("Hi. What's your name? ")

print(name)

print("Hi,", name)

input("\n\nPress the enter key to exit.")
```

Using the input() Function

- **name = input("Hi. What's your name? ")**

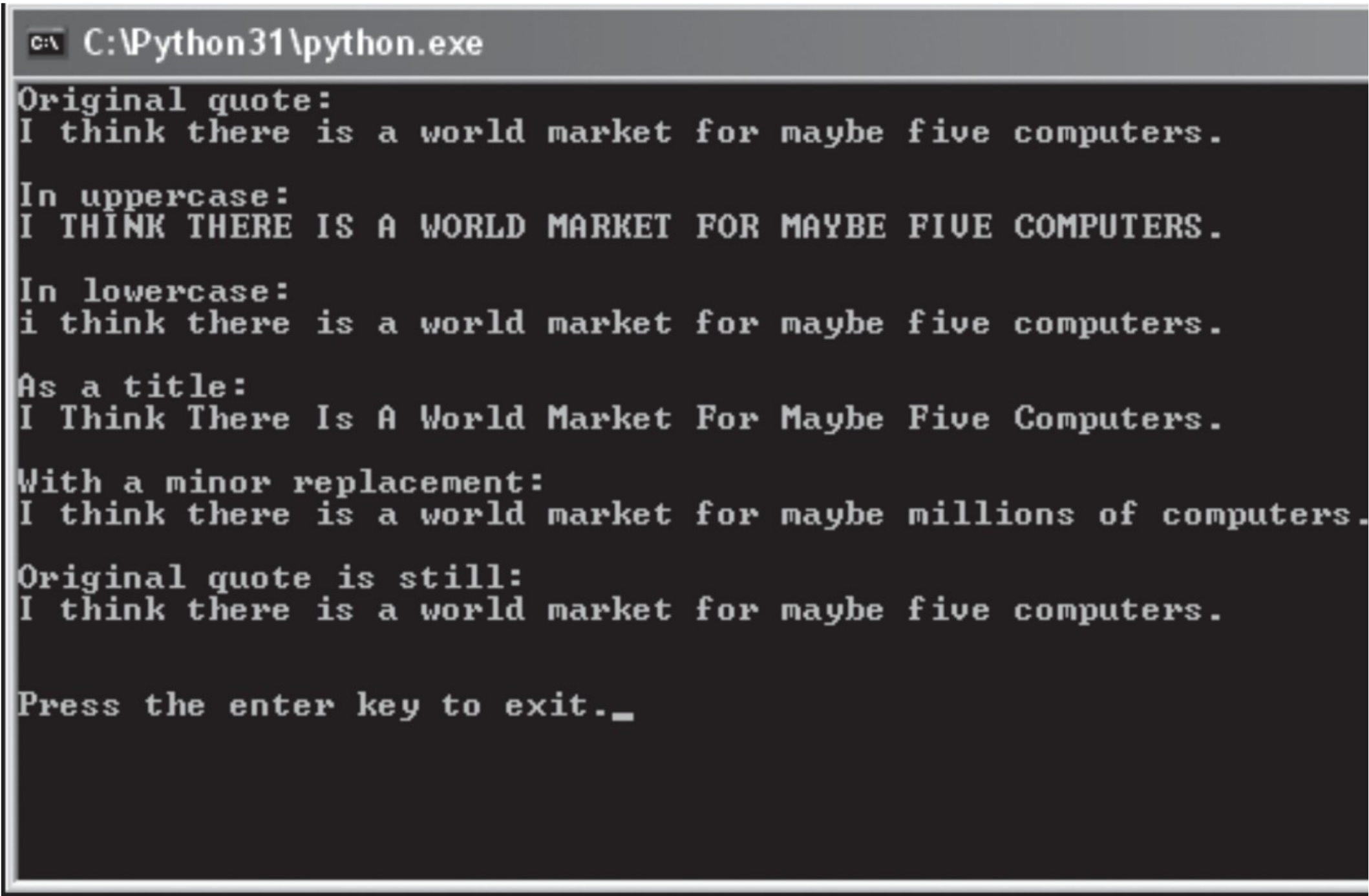
- **input()** gets some text from the user. It takes a string argument that it uses to prompt the user for this text.

- **input()** waits for the user to enter something. Once the user presses the Enter key, input() returns whatever the user typed as a string. That string, the *return value* of the function call, is what name gets.

- **input("\n\nPress the enter key to exit.")**

- The goal of the line waits for the user to press the Enter key. If the return value isn't assigned to a variable, the computer just ignores it. So once the user presses the Enter key, the call to the **input()** function ends, the program ends, and the console window closes.

The Quotation Manipulation Program



quotation_manipulation.py

```
# Quotation Manipulation
# Demonstrates string methods

# quote from IBM Chairman, Thomas Watson
quote = "I think there is a world market for maybe five
computers."

print("Original quote:")
print(quote)

print("\nIn uppercase:")
print(quote.upper())

print("\nIn lowercase:")
print(quote.lower())

print("\nAs a title:")
print(quote.title())
```

```
print("\nWith a minor replacement:")
print(quote.replace("five", "millions of"))

print("\nOriginal quote is still:")
print(quote)

input("\n\nPress the enter key to exit.")
```

Creating New Strings with String Methods

- print(quote**.upper()**) prints a version of quote in all uppercase letters. The line does this through the use of a string method, **upper()**.

- The line becomes equivalent to the following line:

print("I THINK THERE IS A WORLD MARKET FOR MAYBE FIVE COMPUTERS.")

- A *string method* is like an ability a string has. Methods are similar to built-in functions.

- You kick off a method, or *invoke* it, by adding a dot, followed by the name of the method, followed by a pair of parentheses

- print(quote**.lower()**) invokes the **lower()** method of quote to create an all-lowercase-letters version of the string, which the method returns.

- print(quote**.title()**) prints a version of quote that's like a title.

- print(quote**.replace(**"five", "millions of"**)**) prints a new string, where every occurrence of "five" in quote is replaced with "millions of" .

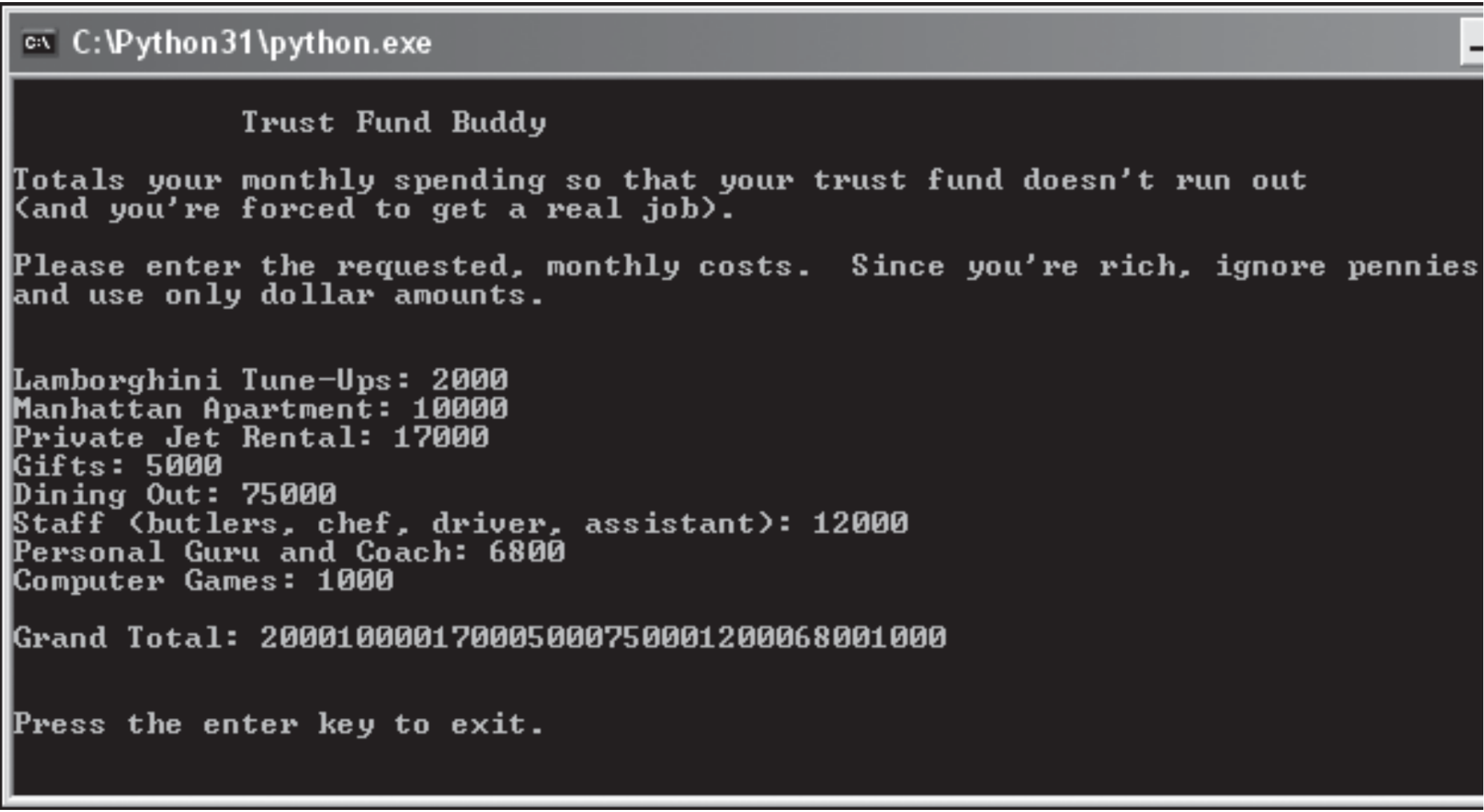
- **replace()** needs at least 2 pieces of information: the old text to be replaced, and the new text that replaces it.

- You can add an optional 3rd argument, an **integer**, that tells the method the maximum number of times to make the replacement.

- String methods create a new string. They don't affect the original one.

Method	Description
`upper()`	Returns the uppercase version of the string.
`lower()`	Returns the lowercase version of the string.
`swapcase()`	Returns a new string where the case of each letter is switched. Uppercase becomes lowercase and lowercase becomes uppercase.
`capitalize()`	Returns a new string where the first letter is capitalized and the rest are lowercase.
`title()`	Returns a new string where the first letter of each word is capitalized and all others are lowercase.
`strip()`	Returns a string where all the white space (tabs, spaces, and newlines) at the beginning and end is removed.
`replace(`*`old`*`, `*`new`*` [,`*`max`*`])`	Returns a string where occurrences of the string *`old`* are replaced with the string *`new`*. The optional *`max`* limits the number of replacements.

The Trust Fund Buddy—Bad Program



trust_fund_bad.py

```
# Trust Fund Buddy - Bad
# Demonstrates a logical error

print(
"""
            Trust Fund Buddy

Totals your monthly spending so that your trust fund doesn't
run out (and you're forced to get a real job).

Please enter the requested, monthly costs.  Since you're rich,
ignore pennies and use only dollar amounts.

"""
)
```

```
car =  input("Lamborghini Tune-Ups: ")
rent = input("Manhattan Apartment: ")
jet = input("Private Jet Rental: ")
gifts = input("Gifts: ")
food = input("Dining Out: ")
staff = input("Staff (butlers, chef, driver, assistant): ")
guru = input("Personal Guru and Coach: ")
games = input("Computer Games: ")

total = car + rent + jet + gifts + food + staff + guru +games

print("\nGrand Total:", total)

input("\n\nPress the enter key to exit.")
```

Tracking Down Logical Errors

- Since the program doesn't crash, you don't get the benefit of an error message to offer a clue. You have to observe the behavior of the program and investigate the code.

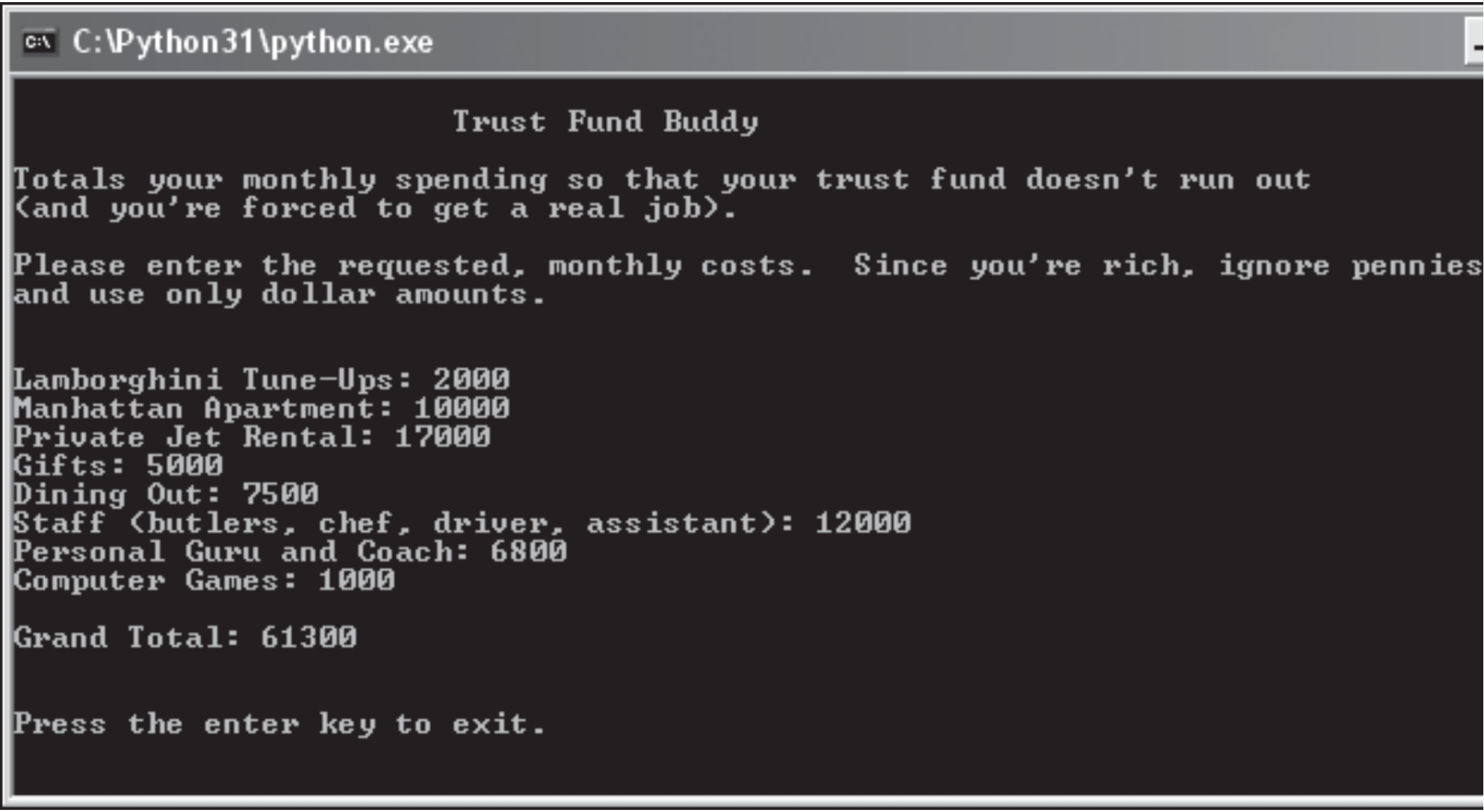
- The huge number in the output is clearly not the sum of all the numbers the user entered.

- the input() function returns a string. So each "number" the user enters is treated like a string. Which means that each variable in the program has a string value associated with it.

- total=car +rent +jet +gifts +food +staff +guru +games is not adding numbers. It's concatenating strings!

- How do you fix it? Somehow those string values need to be converted to numbers.

The Trust Fund Buddy—Good Program



trust_fund_good.py

```
# Trust Fund Buddy - Good
# Demonstrates type conversion

print(
"""
                        Trust Fund Buddy

Totals your monthly spending so that your trust fund doesn't
run out (and you're forced to get a real job).

Please enter the requested, monthly costs.  Since you're rich,
ignore pennies and use only dollar amounts.

"""
)
```

```
car = input("Lamborghini Tune-Ups: ")
car = int(car)

rent = int(input("Manhattan Apartment: "))
jet = int(input("Private Jet Rental: "))
gifts = int(input("Gifts: "))
food = int(input("Dining Out: "))
staff = int(input("Staff (butlers, chef, driver, assistant): "))
guru = int(input("Personal Guru and Coach: ") )
games = int(input("Computer Games: "))

total = car + rent + jet + gifts + food + staff + guru +games

print("\nGrand Total:", total)

input("\n\nPress the enter key to exit.")
```

Converting Strings to Integers

- The function to convert a value to an integer is as

```
car = input("Lamborghini Tune-Ups: ")
car = int(car)
```

- **int()** takes the string referenced by car and returns an integer version of it. Then, car gets this new integer value.

- The following assignments are done in just one line, eg,

```
rent = int(input("Manhattan Apartment: "))
```

Function	Example	Returns	Description
`float(x)`	`float("10.0")`	`10.0`	Returns a floating-point value by converting *x*
`int(x)`	`int("10")`	`10`	Returns an integer value by converting *x*
`str(x)`	`str(10)`	`'10'`	Returns a string value by converting *x*

Using Augmented Assignment Operators

- food = food * 52 to calculate and assign the yearly amount.
- You could accomplish the same thing with food *= 52
- *= is an augmented assignment operator.
- Since assigning a new value to a variable based on its original value is what happens a lot in programming, these operators provide a nice shortcut to a common task.

Operator	Example	Is Equivalent To
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5

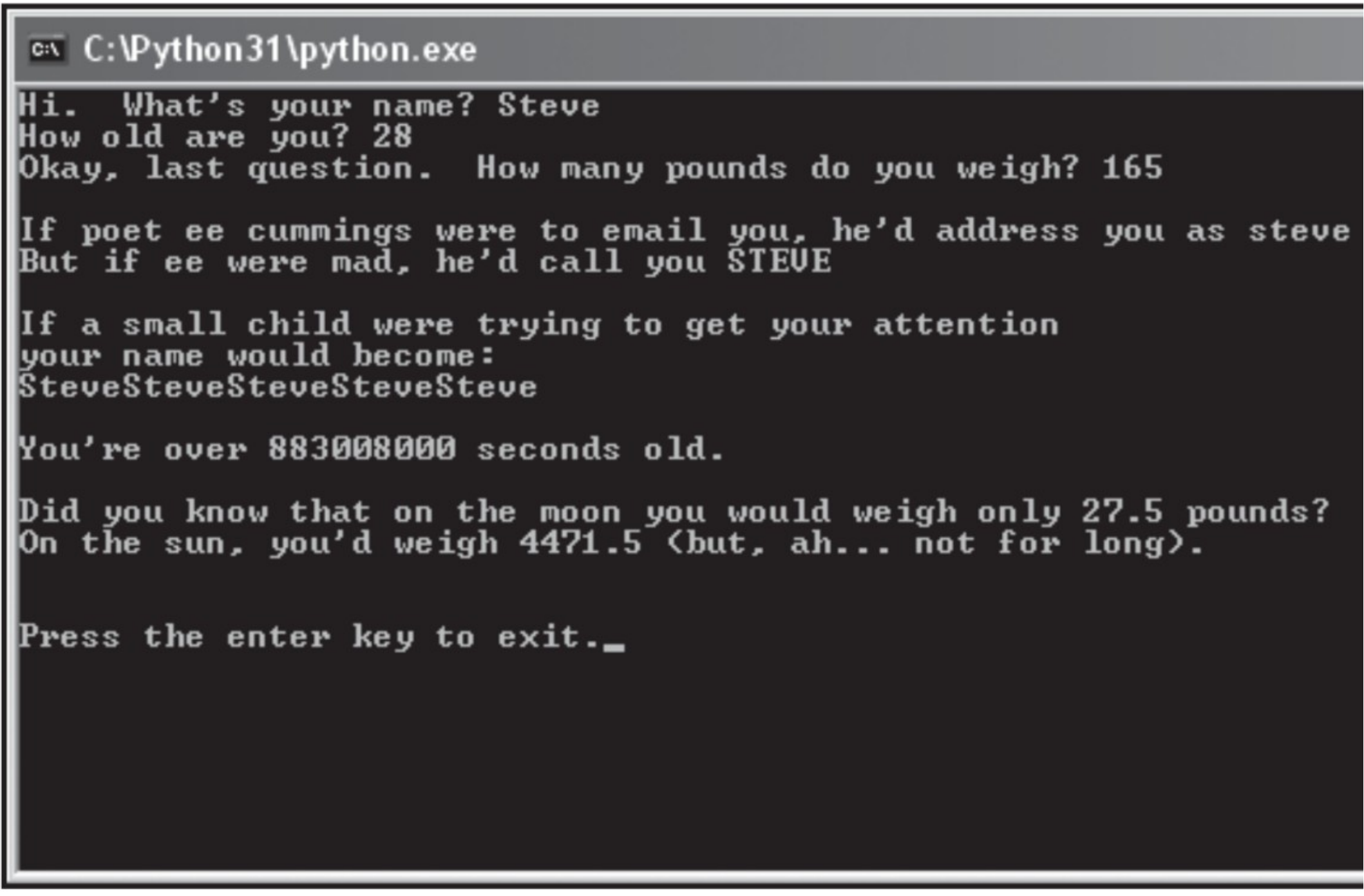
useless_trivia.py

```
# Useless Trivia
#
# Gets personal information from the user and then
# prints true but useless information about him or her

name = input("Hi.  What's your name? ")

age = input("How old are you? ")
age = int(age)

weight = int(input("Okay, last question.  How many pounds
do you weigh? "))

print("\nIf poet ee cummings were to email you, he'd address
you as", name.lower())
print("But if ee were mad, he'd call you", name.upper())
```

```
called = name * 5
print("\nIf a small child were trying to get your attention",)
print("your name would become:")
print(called)

seconds = age * 365 * 24 * 60 * 60
print("\nYou're over", seconds, "seconds old.")

moon_weight = weight / 6
print("\nDid you know that on the moon you would weigh
only", moon_weight, "pounds?")

sun_weight = weight * 27.1
print("On the sun, you'd weigh", sun_weight, "(but, ah... not
for long).")

input("\n\nPress the enter key to exit.")
```

- The program takes 3 pieces of personal information from the user: name, age, and weight. From these mundane items, the program is able to produce some amusing but trivial facts about the person.

the Result of the Useless Trivia Program



Quiz 2: Write a Tipper program where the user enters a restaurant bill. The program should then display two amounts: (bill + a 15 percent tip) and (bill + a 20 percent tip).