Chapter 2

How to write your first programs

Objectives

Applied

1. Use the IDLE shell to test numeric and string operations.
2. Code, test, and debug programs that require the skills that you've learned in this chapter. That includes the use of:

   comments for documenting your code and commenting out statements
   str, int, and float values and variables
   arithmetic expressions
   string concatenation
   special characters in strings
   the built-in print(), input(), str(), float(), int(), and round() functions
   function chaining

Objectives (cont.)

Knowledge

1. Describe the use of indentation when coding Python statements.
2. Describe the use of Python comments, including "commenting out" portions of Python code.
3. Describe these data types: str, int, float.
4. List two recommendations for creating a Python variable name.
5. Distinguish between underscore notation and camel case.
6. Describe the evaluation of an arithmetic expression, including order of precedence and the use of parentheses.
7. Distinguish among these arithmetic operators: /, //, and %.
8. Describe the use of += operator in a compound arithmetic expression.
9. Describe the use of escape sequences when working with strings.

Objectives (cont.)

10. Describe the syntax for calling one of Python's built-in functions.
11. Describe the use of the print(), input(), str(), float(), int(), and round() functions.
12. Describe what it means to chain functions.

The Python code for a Test Scores program

```
#!/usr/bin/env python3

counter = 0
score_total = 0
test_score = 0

while test_score != 999:
    test_score = int(input("Enter test score: "))
    if test_score >= 0 and test_score <= 100:
        score_total += test_score
        counter += 1

average_score = round(score_total / counter)

print("Total Score: " + str(score_total))
print("Average Score: " + str(average_score))
```

An indentation error

```
print("Total Score: " + str(score_total))
 print("Average Score: " + str(average_score))
```

Two ways to continue one statement over two or more lines

Implicit continuation

```
print("Total Score: " + str(score_total)
    + "\nAverage Score: " + str(average_score))
```

Explicit continuation

```
print("Total Score: " + str(score_total) \
    + "\nAverage Score: " + str(average_score))
```

Coding rules

- Python relies on proper indentation. Incorrect indentation causes an error.
- The standard indentation is four spaces.
- With *implicit continuation*, you can divide statements after parentheses, brackets, and braces, and before or after operators like plus or minus signs.
- With *explicit continuation*, you can use the \ character to divide statements anywhere in a line.

The Test Scores program with comments

```
#!/usr/bin/env python3

# This is a tutorial program that illustrates the use of
# the while and if statements

# initialize variables
counter = 0
score_total = 0
test_score = 0

# get scores
while test_score != 999:
    test_score = int(input("Enter test score: "))
    if test_score >= 0 and test_score <= 100:
        score_total += test_score    # add score to total
        counter += 1                 # add 1 to counter
```

The Test Scores program with comments (cont.)

```
# calculate average score
#average_score = score_total / counter
#average_score = round(average_score)
average_score = round(           # implicit continuation
    score_total / counter)       # same results as
                                 # commented out statements

# display the result
print("======================")
print("Total Score: " + str(score_total) # implicit cont.
      + "\nAverage Score: " + str(average_score))
```

Guidelines for using comments

- Use comments to describe portions of code that are hard to understand, but don't overdo them.
- Use comments to comment out (or disable) statements that you don't want to test.
- If you change the code that's described by comments, change the comments too.

The syntax for calling any function

```
function_name([arguments])
```

The print() function

```
print([data])
```

A script with three statements

```
print("Hello out there!")
print()
print("Goodbye!")
```

The console after the program runs

```
Hello out there!

Goodbye!
```

Three Python data types

Data type	Name	Examples
`str`	String	`"Mike"` `"40"` `'Please enter name: '`
`int`	Integer	`21` `450` `0` `-25`
`float`	Floating-point	`21.9` `450.25` `0.01` `-25.2` `3.1416`

Code that initializes variables and assigns data

```
first_name = "Mike"     # sets first_name to a str of "Mike"
quantity1 = 3           # sets quantity1 to an int of 3
quantity2 = 5           # sets quantity2 to an int of 5
list_price = 19.99      # sets list_price to a float of 19.99
```

Code that assigns new data to the variables above

```
first_name = "Joel"     # sets first_name to a str of "Joel"
quantity1 = 10          # sets quantity1 to an int of 10
quantity1 = quantity2   # sets quantity1 to an int of 5
quantity1 = "15"        # sets quantity1 to a str of "15"
```

Code that causes an error due to incorrect case

```
quantity1 = Quantity2  # NameError: 'Quantity2' is not
defined
```

How to code literal values

- To code a *literal value* for a string, enclose the characters of the string in single or double quotation marks. This is called a *string literal*.
- To code a literal value for a number, code the number without quotation marks. This is called a *numeric literal*.

Rules for naming variables

- A variable name must begin with a letter or underscore.
- A variable name can’t contain spaces, punctuation, or special characters other than the underscore.
- A variable name can’t begin with a number, but can use numbers later in the name.
- A variable name can’t be the same as a *keyword* that’s reserved by Python.

Python keywords

```
and         except     lambda      while
as          False      None        with
assert      finally    nonlocal    yield
break       for        not
class       from       or
continue    global     pass
def         if         raise
del         import     return
elif        in         True
else        is         try
```

Two naming styles for variables

```
variable_name        # underscore notation
variableName         # camel case
```

Recommendations for naming variables

- Start all variable names with a lowercase letter.
- Use underscore notation or camel case.
- Use meaningful names that are easy to remember.
- Don't use the names of built-in functions, such as print().

Python's arithmetic operators

Operator	Name
+	Addition
-	Subtraction
*	Multiplication
/	Division
//	Integer division
%	Modulo / Remainder
**	Exponentiation

Examples with two operands

Example	Result
`5 + 4`	`9`
`25 / 4`	`6.25`
`25 // 4`	`6`
`25 % 4`	`1`
`3 ** 2`	`9`

The order of precedence

Order	Operators	Direction
1	`**`	Left to right
2	`*` `/` `//` `%`	Left to right
3	`+` `-`	Left to right

Examples that show the order of precedence and use of parentheses

Example	Result
`3 + 4 * 5`	`23` (the multiplication is done first)
`(3 + 4) * 5`	`35` (the addition is done first)

Code that calculates sales tax

```
subtotal = 200.00
tax_percent = .05
tax_amount = subtotal * tax_percent        # 10.0
grand_total = subtotal + tax_amount        # 210.0
```

Code that calculates the perimeter of a rectangle

```
width = 4.25
length = 8.5
perimeter = (2 * width) + (2 * length)    # 25.5
```

The most useful compound assignment operators

```
+=
-=
*=
```

Two ways to increment the number in a variable

```
counter = 0
counter = counter + 1        # counter = 1
counter += 1                 # counter = 2
```

Code that adds two numbers to a variable

```
score_total = 0              # score_total = 0
score_total += 70            # score_total = 70
score_total += 80            # score_total = 150
```

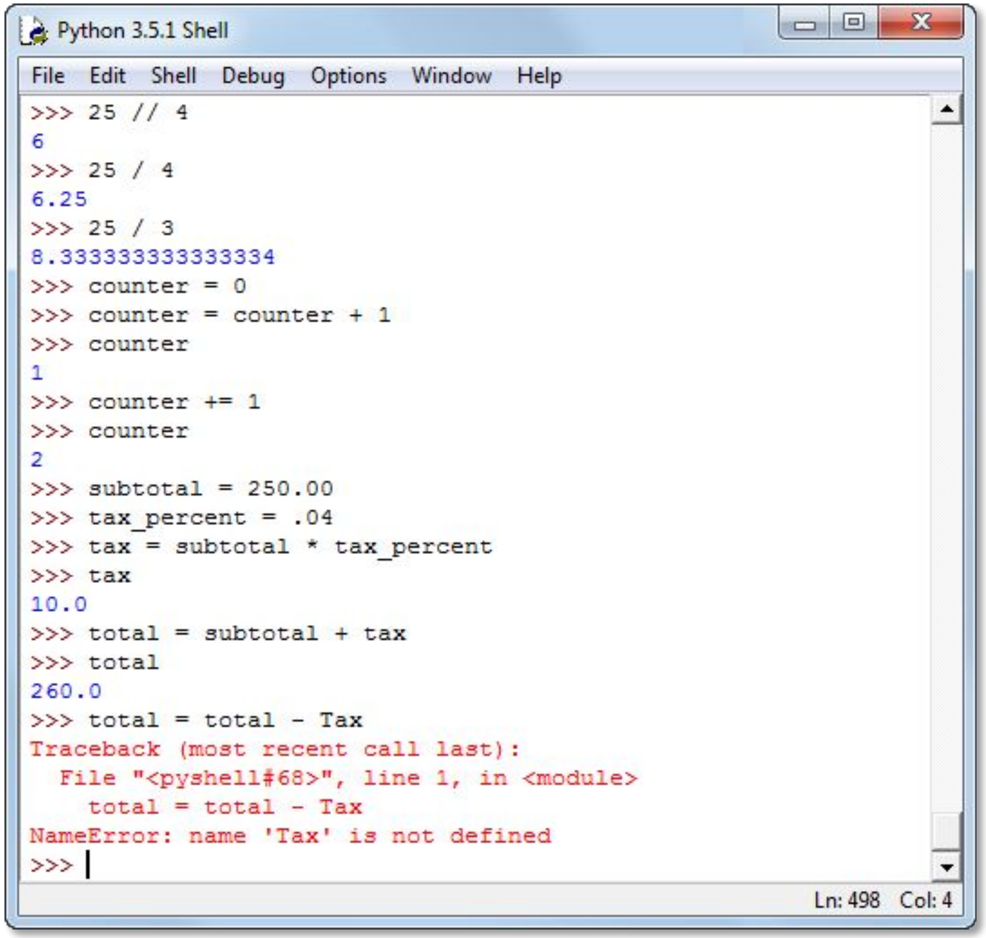
More compound assignment operator examples

```
total = 1000.0
total += 100.0                 # total = 1100.0
counter = 10
counter -= 1                   # counter = 9
price = 100
price *= .8                    # price = 80.0
```

A floating-point result that isn't precise

```
subtotal = 74.95                   # subtotal = 74.95
tax = subtotal * .1                # tax = 7.495000000000001
```

The Python shell after some numeric testing



```
>>> 25 // 4
6
>>> 25 / 4
6.25
>>> 25 / 3
8.333333333333334
>>> counter = 0
>>> counter = counter + 1
>>> counter
1
>>> counter += 1
>>> counter
2
>>> subtotal = 250.00
>>> tax_percent = .04
>>> tax = subtotal * tax_percent
>>> tax
10.0
>>> total = subtotal + tax
>>> total
260.0
>>> total = total - Tax
Traceback (most recent call last):
  File "<pyshell#68>", line 1, in <module>
    total = total - Tax
NameError: name 'Tax' is not defined
>>> 
```

How to use the shell

- To test a statement, type it at the prompt and press the Enter key. You can also type the name of a variable at the prompt to see what its value is.
- Any variables that you create remain active for the current session. As a result, you can use them in statements that you enter later in the same session.
- To retype your previous entry, press Alt+p (Windows) or Command+p (OS X).
- To cycle through all of the previous entries, continue pressing the Alt+p (Windows) or Command+p (OS X) keystroke until the entry you want is displayed at the prompt.

How to assign strings to variables

```
first_name = "Bob"                     # Bob
last_name = 'Smith'                    # Smith
name = ""                              # (empty string)
name = "Bob Smith"                     # Bob Smith
```

How to join three strings with the + operator

```
name = last_name + ", " + first_name    # Smith, Bob
```

The str() function

```
str(data)
```

How to join a string and a number

```
name = "Bob Smith"
age = 40
message = name + " is " + str(age) + " years old."
```

What happens if you don't use the str() function

```
message = name + " is " + age + " years old."
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    message = name + " is " + age + " years old."
```

Common escape sequences

Sequence	Character
`\n`	New line
`\t`	Tab
`\r`	Return
`\"`	Quotation mark in a double quoted string
`\'`	Quotation mark in a single quoted string
`\\`	Backslash

The new line character

```
print("Title: Python Programming\nQuantity: 5")
```

Displayed on the console

```
Title: Python Programming
Quantity: 5
```

The tab and new line characters

```
print("Title:\t\tPython Programming\nQuantity:\t5")
```

Displayed on the console

```
Title:          Python Programming
Quantity:       5
```

The backslash in a Windows path

```
print("C:\\murach\\python")
```

Displayed on the console

```
C:\murach\python
```

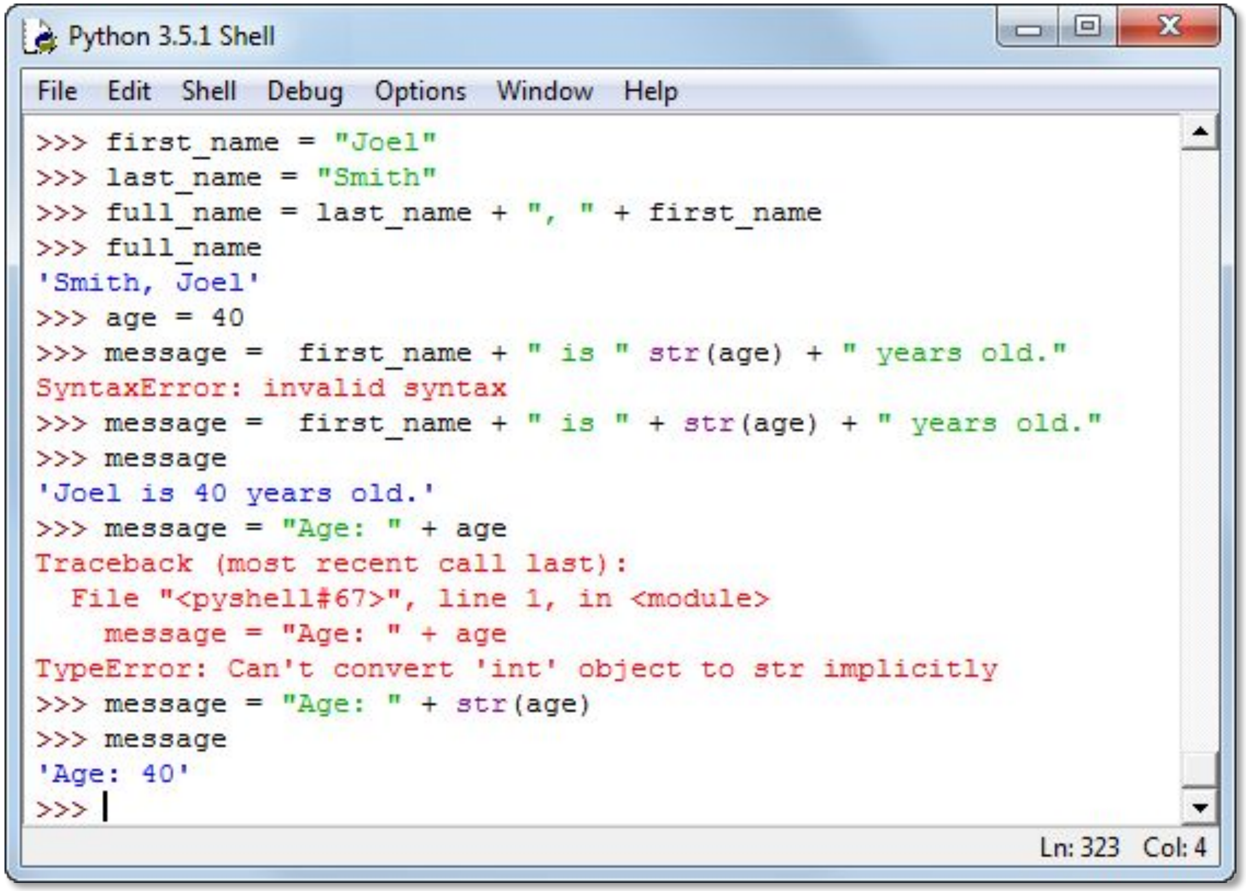
Four ways to include quotation marks in a string

```
"Type \"x\" to exit"   # String is: Type "x" to exit.
'Type \'x\' to exit'   # String is: Type 'x' to exit.
"Type 'x' to exit"     # String is: Type 'x' to exit.
'Type "x" to exit'     # String is: Type "x" to exit.
```

Implicit continuation of a string over several lines

```
print("Total Score: "
      + str(score_total)
      + "\nAverage Score: "
      + str(average_score))
```

The Python shell with string testing



```
>>> first_name = "Joel"
>>> last_name = "Smith"
>>> full_name = last_name + ", " + first_name
>>> full_name
'Smith, Joel'
>>> age = 40
>>> message =  first_name + " is " str(age) + " years old."
SyntaxError: invalid syntax
>>> message =  first_name + " is " + str(age) + " years old."
>>> message
'Joel is 40 years old.'
>>> message = "Age: " + age
Traceback (most recent call last):
  File "<pyshell#67>", line 1, in <module>
    message = "Age: " + age
TypeError: Can't convert 'int' object to str implicitly
>>> message = "Age: " + str(age)
>>> message
'Age: 40'
>>> 
```

How to use the shell

- To test a statement, type it at the prompt and press the Enter key. You can also type the name of a variable at the prompt to see what its value is.
- Any variables that you create remain active for the current session. As a result, you can use them in statements that you enter later in the same session.
- To retype the previous statement on Windows, press Alt+p. To cycle through all of the previous statements, continue pressing Alt+p. On OS X, use Command+p.

The syntax of the print() function

```
print(data[, sep=' '][, end='\n'])
```

Three print() functions

```
print(19.99)                                    # 19.99
print("Price:", 19.99)                          # Price: 19.99
print(1, 2, 3, 4)                               # 1 2 3 4
```

Two ways to get the same result

A print() function that receives four arguments

```
print("Total Score:", score_total,
      "\nAverage Score:", average_score)
```

A print() function that receives one string as the argument

```
print("Total Score: " + str(score_total) +
     "\nAverage Score: " + str(average_score))
```

The data that's displayed by both functions

```
Total Score: 240
Average Score: 80
```

Examples that use the sep and end arguments

```
print(1,2,3,4,sep=' | ')      # 1 | 2 | 3 | 4
print(1,2,3,4,end='!!!')      # 1 2 3 4!!!
```

The input() function

```
input([prompt])
```

Code that gets string input from the user

```
first_name = input("Enter your first name: ")
print("Hello, " + first_name + "!")
```

The console

```
Enter your first name: Mike
Hello, Mike!
```

Another way to get input from the user

```
print("What is your first name?")
first_name = input()
print("Hello, " + first_name + "!")
```

The console

```
What is your first name?
Mike
Hello, Mike!
```

Code that attempts to get numeric input

```
score_total = 0
score = input("Enter your score: ")
score_total += score      # causes an error
                          # because score is a string
```

Three functions for working with numbers

```
int(data)
float(data)
round(number [,digits])
```

Code that causes an exception

```
x = 15
y = "5"
z = x + y    # TypeError: can't add an int to a str
```

How using the int() function fixes the exception

```
x = 15
y = "5"
z = x + int(y)   # z is 20
```

Code that gets an int value from the user

```
quantity = input("Enter the quantity: ")   # get str type
quantity = int(quantity)                  # convert to int type
```

How to use chaining to get the value in one statement

```
quantity = int(input("Enter the quantity: "))
```

Code that gets a float value from the user

```
price = input("Enter the price: ")       # get str type
price = float(price)                # convert to float type
```

How to use chaining to get the value in one statement

```
price = float(input("Enter the price: "))
```

Code that uses the round() function

```
miles_driven = 150
gallons_used = 5.875
mpg = miles_driven / gallons_used   # 25.53191489361702
mpg = round(mpg, 2)                 # 25.53
```

How to combine the last two statements

```
mpg = round(miles_driven / gallons_used, 2)
```

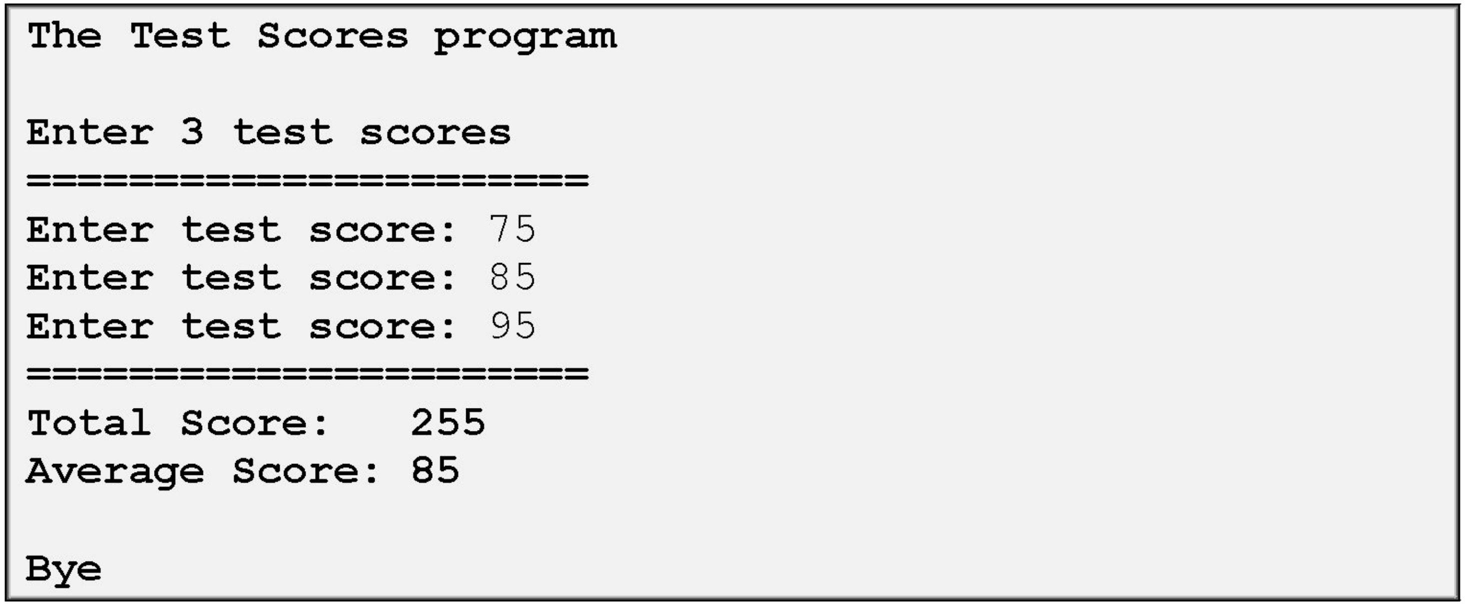
The console

```
The Miles Per Gallon program

Enter miles driven:         150
Enter gallons of gas used: 35.875

Miles Per Gallon:      4.18

Bye
```

The code

```
#!/usr/bin/env python3

# display a title
print("The Miles Per Gallon program")
print()

# get input from the user
miles_driven= float(input("Enter miles driven:\t\t"))
gallons_used = float(input("Enter gallons of gas used:\t"))

# calculate and round miles per gallon
mpg = miles_driven / gallons_used
mpg = round(mpg, 2)

# display the result
print()
print("Miles Per Gallon:\t\t" + str(mpg))
print()
print("Bye")
```

The console



```
The Test Scores program

Enter 3 test scores
======================
Enter test score: 75
Enter test score: 85
Enter test score: 95
======================
Total Score:   255
Average Score: 85

Bye
```

The code

```
#!/usr/bin/env python3

# display a title
print("The Test Scores program")
print()
print("Enter 3 test scores")
print("======================")

# get scores from the user and accumulate the total
total_score = 0       # initialize the total_score variable
total_score += int(input("Enter test score: "))
total_score += int(input("Enter test score: "))
total_score += int(input("Enter test score: "))

# calculate average score
average_score = round(total_score / 3)

# format and display the result
print("======================")
print("Total Score:  ", total_score,
      "\nAverage Score:", average_score)
print()
print("Bye")
```