Chapter 2
Input, Processing, and Output

STARTING OUT WITH

Python

First Edition

by
Tony Gaddis

2.1 Designing a Program

Concept:

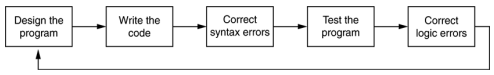
Programs must be carefully designed before they are written. During the design process, programmers use tools such as pseudocode and flowcharts to create models of programs.

2.1 Designing a Program

The Program Development Cycle

- Design the Program
- Write the Code
- Correct Syntax Errors
- Test the Program
- Correct Logic Errors

Figure 2-1 The program development cycle



2.1 Designing a Program

More About the Design Process

The process of designing a program comprises two important steps:

1. Understand the task that the program is to perform
2. Determine the steps that must be taken to perform the task

2.1 Designing a Program

More About the Design Process

1. Understand the task that the program is to perform
   a. Interview the customer
   b. Develop the ***software requirement*** (a simple task that the program must perform in order to satisfy the customer)

2.1 Designing a Program

More about the design process

2. Determine the steps that must be taken to perform the task.
   - Develop an algorithm, a set of well-defined logical steps that must be taken in order to perform a task.

   For Example:

   Write a program to calculate and display the gross pay for an hourly paid employee.

   1. *Get the number of hours worked.*
   2. *Get the hourly pay rate*
   3. *Multiply the number of hours worked by the hourly pay rate.*
   4. *Display the result of the calculation that was performed in step 3.*

2.1 Designing a Program

Pseudocode is an informal language used to develop a program's design. It has no syntax rules and is not meant to be compiled or executed.

For Example:

Write a program to calculate and display the gross pay for an hourly paid employee.

Input the hours worked

Input the hourly pay rate

Calculate gross pay as hours worked multiplied by hourly pay rate

Display the gross pay

2.1 Designing a Program

A flowchart is a tool that graphically depicts the steps that take place in a program.

Types of Symbols:

- **Ovals—appear at the top or bottom of the flowchart, and are called terminal symbols**
- **Parallelograms—used as input and output symbols**
- **Rectangles—used as processing symbols**
- **Arrows—represent the flow of the program**

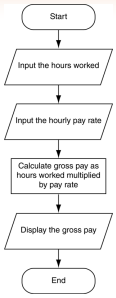


Figure 2-2 Flowchart for the pay calculating program

2.2 Input, Processing, and Output

Concept:

Input is data that the program receives. When a program receives data, it usually processes it by performing some operation with it. The result of the operation is sent out of the program as output.

2.2 Input, Processing, and Output

Computer programs perform the following three steps:

1. **Input is received**
2. **Some process is performed on the input**
3. **Output is produced**

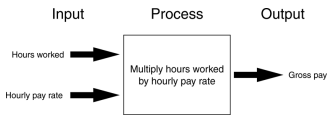


Figure 2-3 The input, processing, and output of the pay calculating program

2.3 Displaying Output with the `print` Statement

Concept:

You use the `print` statement to display output in a Python program.

2.3 Displaying Output with the `print` Statement

Program 2-1 (output.py)

```
1 print 'Kate Austen'
2 print '123 Dharma Lane'
3 print 'Asheville, NC 28899'
```

2.3 Displaying Output with the `print` Statement

Strings and String Literals

- A string is a sequence of characters that is used as data.
- When a string appears in the code of a program, it is called a string literal.
- In Python, single-quotes (‘) or double-quotes (“) can be used to enclose string literals.

2.3 Displaying Output with the `print` Statement

Strings and String Literals

If a string literal contains either a single-quote or apostrophe, then enclose the string literal in double-quotes.

Program 2-3 (apostrophe.py)

```
1  print "Don't fear!"
2  print "I'm here!"
```


2.3 Displaying Output with the `print` Statement

Strings and String Literals

If a string literal contains a double-quote, then enclose the string literal in single-quotes.

Program 2-4 (display_quote.py)

```
1  print 'Your assignment is to read "Hamlet" by tomorrow.'
```


2.4 Comments

Concept:

Comments are notes of explanations that document lines or sections of a program. Comments are part of the program, but the Python interpreter ignores them. They are intended for people who may be reading the source code.

2.4 Comments

Two types of comments:

1. Full line comment
   ```
   # This program calculates net pay
   ```
2. End-line comment
   ```
   print "John Smith"   # Display the name
   ```

Python begins a comment with the # character.

2.5 Variables

Concept:

A variable is a name that represents a value stored in the computer’s memory.

2.5 Variables

Creating Variables with Assignment Statements

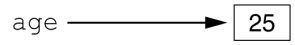
Assignment Statement

variable = *expression*

where,

variable	name of the variable
=	assignment operator
expression	value or piece of code that results in a value

Figure 2-4 The age variable references the value 25

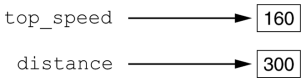


2.5 Variables

Program 2-8 (variable_demo2.py)

```
1  # Create two variables: top_speed and distance.
2  top_speed = 160
3  distance = 300
4
5  # Display the values referenced by the variables.
6  print 'The top speed is'
7  print top_speed
8  print 'The distance traveled is'
9  print distance
```

Figure 2-5 Two variables



2.5 Variables

Variable Naming Rules

- Can not use Python key words
- Can not contain spaces
- First character must be one of the letters *a* through *z*, *A* through *Z*, or an underscore character(_)
- After the first character use letters *a* through *z*, *A* through *Z*, digits *0* through *9*, or underscore
- Uppercase and lowercase characters are distinct (We say that Python is a "case-sensitive" programming language.)

2.5 Variables

Variable Naming Rules

For readability use one of the below styles for multiword variable name:

- Use the underscore character to represent a space
  - gross_pay
  - hot_dogs_sold_today
- *camelCase* naming convention (recommended if you will continue on to Java)
  - grossPay
  - hotDogsSoldToday

2.5 Variables

Variable Naming Rules

Table 2-1 Sample variable names

Variable Name	Legal or Illegal?
`units_per_day`	Legal
`dayOfWeek`	Legal
`3dGraph`	Illegal. Variable names cannot begin with a digit.
`June1997`	Legal
`Mixture#3`	Illegal. Variable names may only use letters, digits, or underscores.

2.5 Variables

Displaying Multiple Items with the `print` Statement

Use a comma to separate the items.

Program 2-9 (variable_demo3.py)

```
1  # This program demonstrates a variable.
2  room = 503
3  print 'I am staying in room number', room
```

2.5 Variables

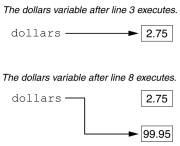
Variable Reassignment

Variables are called "variables" because they can reference different values while a program is running.

Program 2-10 (variable_demo4.py)

```
1 # This program demonstrates variable reassignment.
2 # Assign a value to the dollars variable.
3 dollars = 2.75
4 print 'I have', dollars, 'in my account.'
5
6 # Reassign dollars so it references
7 # a different value.
8 dollars = 99.95
9 print 'But now I have', dollars, 'in my account!'
```

Figure 2-6 Variable reassignment in Program 2-10



2.5 Variables

Numeric Data Types and Literals

A number that is written into a program's code is called a *numeric literal*.

- A numeric literal that is written as a whole number with no decimal point is considered an `int`.
  For example:
  7124, 503, and -9
- A numeric literal that is written with a decimal point is considered a `float`.
  For example:
  1.5, 3.1415, and 5.0

2.5 Variables

Storing Strings with the `str` Data Type

The `str` data type is used to store strings in memory.

Program 2-11 (string_variable.py)

```
1 # Create variables to reference two strings.
2 first_name = 'Kathryn'
3 last_name = 'Marino'
4
5 # Display the values referenced by the variables.
6 print first_name, last_name
```


2.6 Reading Input from the Keyboard

Concept:

Programs commonly need to read input typed by the user on the keyboard. We will use the Python functions to do this.

2.6 Reading Input from the Keyboard

Python uses built-in functions to read input from the keyboard.

A ***function*** is a piece of prewritten code that performs an operation and then returns a value back to the program.

The `input` function can be used to read numeric data from the keyboard.

2.6 Reading Input from the Keyboard

Reading Numbers with the `input` Function

Use the `input` function in an assignment statement:

variable = input (prompt)

where,

variable	name of the variable that will reference the data
=	assignment operator
input	name of the function
prompt	string that is displayed on the screen

For example:

hours = input ('How many hours did you work?')

2.6 Reading Input from the Keyboard

Reading Strings with the `raw_input` Function

The `raw_input` function retrieves all keyboard input as a string.

Program 2-13 (string_input.py)

```
1 # Get the user's first name.
2 first_name = raw_input('Enter your first name: ')
3
4 # Get the user's last name.
5 last_name = raw_input('Enter your last name: ')
6
7 # Print a greeting to the user.
8 print 'Hello', first_name, last_name
```


2.7 Performing Calculations

Concept:

Python has numerous operators that can be used to perform mathematical calculations.

2.7 Performing Calculations

A programmer's tools for performing calculations are ***math operators***, and the following are provided by Python:

Table 2-2 Python math operators

Symbol	Operation	Description
+	Addition	Adds two numbers
-	Subtraction	Subtracts one number from another
*	Multiplication	Multiplies one number by another
/	Division	Divides one number by another and gives the quotient
%	Remainder	Divides one number by another and gives the remainder
**	Exponent	Raises a number to a power

2.7 Performing Calculations

For Example: A retail business is planning to have a storewide sale where the prices of all items will be 20 percent off. We have been asked to write a program to calculate the sale price of an item.

Program 2-15 (sale_price.py)

```
1  # This program gets an item's original price and
2  # calculates its sale price, with a 20% discount.
3
4  # Get the item's original price.
5  original_price = input("Enter the item's original price: ")
6
7  # Calculate the amount of the discount.
8  discount = original_price * 0.2
9
10 # Calculate the sale price.
11 sale_price = original_price - discount
12
13 # Display the sale price.
14 print 'The sale price is', sale_price
```


2.7 Performing Calculations

Integer Division

When an integer is divided by an integer the result will also be an integer. This behavior is known as *integer division*.

For Example:

```
num_add = 17 + 5          # the answer will be 22
num_div_int = 3 / 2       # the answer is 1; 0.5 is the fractional part
                          # and it is truncated.
num_div_real = 3.0 / 2.0  # the answer is 1.5
```


2.7 Performing Calculations

Operator Precedence

The precedence of the math operators, from highest to lowest, are:

1. Operations that are enclosed in parentheses.
2. Exponentiation **
3. Multiplication *, division /, and remainder %
4. Addition + and subtraction -

Table 2-3 Some expressions

Expression	Value
5 + 2 * 4	13
10 / 2 - 3	2
8 + 12 * 2 - 4	28
6 - 3 * 2 + 7 - 1	6

2.7 Performing Calculations

Grouping with Parentheses

Table 2-4 More expressions and their values

Expression	Value
`(5 + 2) * 4`	28
`10 / (5 – 3)`	5
`8 + 12 * (6 – 2)`	56
`(6 – 3) * (2 + 7) / 3`	9

2.7 Performing Calculations

The Exponent and Remainder Operators

**** is the exponent operator. Its purpose is to raise a number to a power.**

For Example:

```
area = length**2
```

% is the remainder operator. Its purpose is to perform division and return the remainder.

For Example:

```
leftover = 17 % 3    #remainder is 2
```


2.7 Performing Calculations

Converting Math Formulas to Programming Statements

Algebraic Expression

$$x = \frac{a + b}{3}$$

Programming Expression

```
x = (a + b) / 3
```

Table 2-6 Algebraic and programming expressions

Algebraic Expression	Python Statement
$y = 3\frac{x}{2}$	`y = 3 * x / 2`
$z = 3bc + 4$	`z = 3 * b * c + 4`
$a = \frac{x + 2}{b - 1}$	`a = (x + 2) / (b - 1)`

2.7 Performing Calculations

Data Type Conversion

Python follows the following rules when evaluating mathematical expressions:

- **When an operation is performed on two `int` values, the result will be an `int`.**
- **When an operation is performed on two `float` values, the result will be a `float`.**
- **When an operation is performed on an `int` and a `float`, the `int` value will be temporarily converted to a `float` and the result of the operation will be a `float`.**

2.7 Performing Calculations

Data Type Conversion

Python built-in `float()` and `int()` functions.

For Example:

```
x = 27.9
y = int(x)      # y will be assigned 27
x = -12.9
y = int(x)      # y will be assigned -12
y = 7
x = float(y)    # x will be assigned 7.0
```


2.7 Performing Calculations

Breaking Long Statements into Multiple Lines

Python allows for a break in a statement into multiple lines by using the **line continuation characters**, which is a backslash (\).

For Example:

```
print 'We sold', units_solld, \
      'for a total of', sales_amount
result = var1 * 2 + var2 * 3 + \
         var3 * 4 + var4 * 5
```

2.8 More About Data Output

Suppressing the `print` Statement's Newline

The `print` statement displays a string and then prints an unseen newline character.

For Example:

```
print 'One'
print 'Two'
print 'Three'
```

Output:

```
One
Two
Three
```

If you do not want to start a new line of output, you can use a trailing comma.

For Example:

```
print 'One',
print 'Two',
print 'Three'
```

Output:

```
One Two Three
```


2.8 More About Data Output

Escape Characters

A special character that is preceded with a backslash (\), appearing inside a string literal.

For Example:

```
print 'One\nTwo\nThree'
```

Output

```
One
Two
Three
```

Table 2-7 Some of Python's escape characters

Escape Character	Effect
`\n`	Causes output to be advanced to the next line.
`\t`	Causes output to skip over to the next horizontal tab position.
`\'`	Causes a single quote mark to be printed.
`\"`	Causes a double quote mark to be printed.
`\\`	Causes a backslash character to be printed.

2.8 More About Data Output

Displaying Multiple Items with the + Operator

When the + is used with two strings, it performs string concatenation.

For Example:

```
print 'this is' + 'one string.'     # one string is appended to another
```

Output:

```
This is one string
```


2.8 More About Data Output

Formatting Numbers

Floating-point Numbers:

- Without formatting the number can be displayed with up to 12 significant digits
- The % symbol is a ***string format operator*** when the operand on the left side of the % is a string
- A ***formatting specifier*** is a special set of characters that specify how a value should be formatted

For Example:

```
my_value = 7.23456
print 'The value is %.2f' % my_value
```

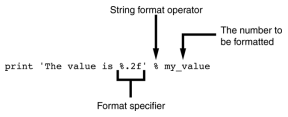
Output

```
The value is 7.23
```


2.8 More About Data Output

Formatting Numbers

Figure 2-8 Using the string format operator



Program 2-21 (formatting.py)

```
1 # This program demonstrates how a floating-point
2 # number can be formatted.
3 amount_due = 5000.0
4 monthly_payment = amount_due / 12.0
5 print 'The monthly payment is %.2f' % monthly_payment
```


2.8 More About Data Output

Formatting Numbers

Formatting Multiple Values

For Example:

```
value1 = 6.7891234
value2 = 1.2345678
print 'The values are %.1f and %.3f' %
(value1, value2)
```

Output

```
The values are 6.8 and 1.235
```


2.8 More About Data Output

Formatting Numbers

Specify a Minimum Field Width

Program 2-23 (columns.py)

```
1  # This program displays the following
2  # floating-point numbers in a column
3  # with their decimal points aligned.
4  num1 = 127.899
5  num2 = 3465.148
6  num3 = 3.776
7  num4 = 264.821
8  num5 = 88.081
9  num6 = 799.999
10
11 # Display each number in a field of 7 spaces
12 # with 2 decimal places.
13 print '%7.2f' % num1
14 print '%7.2f' % num2
15 print '%7.2f' % num3
16 print '%7.2f' % num4
17 print '%7.2f' % num5
18 print '%7.2f' % num6
```


2.8 More About Data Output

Formatting Numbers

Formatting Integers and Strings

- The %d formatting specifier is used to format an integer
- The %s formatting specifier is used to format a string

For Example:

```
name = 'Ringo'
day = 'Monday'
sales = 8450.5543
print 'Hello %s. Good to see you!' % name
print 'The sales on %s were $%.2f.' % (day, sales)
```

Output

```
Hello Ringo. Good to see you!
The sales on Monday were $8450.55.
```


Chapter 2
Input, Processing, and Output

QUESTIONS

?