

Python For Everyone, Enhanced eText

Cay S. Horstmann; Rance D. Necaise

Expand | Collapse

Title Page

Getting the Most from Your eText

Quick Reference

1. Introduction

2. Programming with Numbers and Strings

2.1 Variables

2.2 Arithmetic

2.3 Problem Solving: First Do It By Hand

2.4 Strings

2.5 Input and Output

2.6 Graphics: Simple Drawings

Chapter Summary

Interactive Review and Practice

End-of-Chapter Exercises

3. Decisions

4. Loops

5. Functions

6. Lists

7. Files and Exceptions

8. Sets and Dictionaries

9. Objects and Classes

The assignment operator = does not denote mathematical equality.

The = sign does not mean that the left-hand side is equal to the right-hand side. Instead, the value on the right-hand side is placed into the variable on the left.

Do not confuse this assignment operator with the = used in algebra to denote equality. Assignment is an instruction to do something—namely, place a value into a variable.

For example, in Python, it is perfectly legal to write

```
cansPerPack = cansPerPack + 2
```

The second statement means to look up the value stored in the variable `cansPerPack`, add 2 to it, and place the result back into `cansPerPack`. (See Figure 2.) The net effect of executing this statement is to increment `cansPerPack` by 2. If `cansPerPack` was 8 before execution of the statement, it is set to 10 afterwards. Of course, in mathematics it would make no sense to write that $x = x + 2$. No value can equal itself plus 2.

1 Compute the value of the right-hand side

```
cansPerPack = 8
```

```
cansPerPack + 2
```

10

2 Store the value in the variable

```
cansPerPack = 10
```

Figure 2 Executing the Assignment `cansPerPack = cansPerPack + 2`

SELF CHECK

1. Which of the following are valid variable definitions? (Assume that any variable correctly defined in the activity is defined in the following lines.)

Valid Invalid

```
cats = 3
```

Defines variable cats and initializes it to 3.

Valid Invalid

```
dogs = cats + 1
```

The initial value need not be a literal value.

Valid Invalid

```
parakeets = parakeets + 1
```

Error: All variables on the right-hand side of the = must have been defined before they are used.

Valid Invalid

```
10 = hamsters
```

Error: The left-hand side of the = must be a variable.

Valid Invalid

```
goldfish = cats
```

The value of a previously defined variable can be used to initialize a new variable.

5 correct, 0 errors, 100%

Back to Page

24 / 554

2.1

2.1 - Variables

Python For Everyone, Enhanced eText

Cay S. Horstmann; Rance D. Necaise

Expand | Collapse

Title Page

Getting the Most from Your eText

Quick Reference

1. Introduction

2. Programming with Numbers and Strings

2.1 Variables

2.2 Arithmetic

2.3 Problem Solving: First Do It By Hand

2.4 Strings

2.5 Input and Output

2.6 Graphics: Simple Drawings

Chapter Summary

Interactive Review and Practice

End-of-Chapter Exercises

3. Decisions

4. Loops

5. Functions

6. Lists

7. Files and Exceptions


8. Sets and Dictionaries

9. Objects and Classes

2.2 Arithmetic

In the following sections, you will learn how to carry out arithmetic calculations in Python.

2.2.1 Basic Arithmetic Operations



Python supports the same four basic arithmetic operations as a calculator—addition, subtraction, multiplication, and division—but it uses different symbols for multiplication and division. You must write `a * b` to denote multiplication. Unlike in mathematics, you cannot write `a b`, `a · b`, or `a × b`. Similarly, division is always indicated with `a /`, never `a ÷` or a fraction bar. For example, $\frac{a+b}{2}$ becomes `(a + b) / 2`.

The symbols `+`, `-`, `*`, `/` for the arithmetic operations are called *operators*. The combination of variables, literals, operators, and parentheses is called an **expression**. For example, `(a + b) / 2` is an expression. Parentheses are used just as in algebra: to indicate in which order the parts of the expression should be computed. For example, in the expression `(a + b) / 2`, the sum `a + b` is computed first, and then the sum is divided by 2. In contrast, in the expression `a + b / 2`, only `b` is divided by 2, and then the sum of `a` and `b / 2` is formed. As in regular algebraic notation, multiplication and division have a *higher precedence* than addition and subtraction. For example, in the expression `a + b / 2`, the `/` is carried out first, even though the `+` operation occurs further to the left. Again, as in algebra, operators with the same precedence are executed left-to-right. For example, `10 - 2 - 3` is `8 - 3` or 5.

Mixing integers and floating-point values in an arithmetic expression yields a floating-point value. If you mix integer and floating-point values in an arithmetic expression, the result is a floating-point value. For example, `7 + 4.0` is the floating-point value 11.0.

SELF CHECK

•• 1. Assuming that `x` and `y` are variables that store floating-point values, translate the following expressions into Python.

GOOD JOB! ✓


Mathematical Expression	Python Code	Explanation
$x - y$	<code>x - y</code>	Use <code>+</code> and <code>-</code> for addition and subtraction.
xy	<code>x * y</code>	Use <code>*</code> for multiplication.
$x \times y$	<code>x * y</code>	Use <code>*</code> , not the <code>x</code> or <code>·</code> symbol.
$x + 2y$	<code>x + 2 * y</code>	As in mathematics, <code>*</code> binds more strongly than <code>+</code> .
$2(x + y)$	<code>2 * (x + y)</code>	Use parentheses to control the order of evaluation.
$\frac{x+y}{2}$	<code>(x + y) / 2</code>	Use <code>/</code> for division, not the fraction bar.
$(x + y) \div 2$	<code>(x + y) / 2</code>	Use <code>/</code> instead of <code>÷</code> .

7 correct, 0 errors, 100%, 34 seconds

Start over

Back to Page

31 / 554



Python For Everyone, Enhanced eText

Cay S. Horstmann; Rance D. Necaise

Expand | Collapse

Title Page

Getting the Most from Your eText

Quick Reference

1. Introduction

2. Programming with Numbers and Strings

2.1 Variables

2.2 Arithmetic

2.3 Problem Solving: First Do It By Hand

2.4 Strings

2.5 Input and Output

2.6 Graphics: Simple Drawings

Chapter Summary

Interactive Review and Practice

End-of-Chapter Exercises

3. Decisions

4. Loops

5. Functions

6. Lists

7. Files and Exceptions

8. Sets and Dictionaries

9. Objects and Classes

10. Inheritance

11. Recursion


12. Sorting and Searching

Appendices

Glossary

Illustration Credits

Wiley End User License Agreement



SELF CHECK

• 1. Select the assignment statement that correctly implements the algorithmic step below:
number of pairs = integer part of (total width - tile width) / (2 x tile width)

☐ `numberOfPairs = (totalWidth - tileWidth) / (2.0 * tileWidth)`

☐ `numberOfPairs = (totalWidth - tileWidth) // 2 * tileWidth`

☒ `numberOfPairs = (totalWidth - tileWidth) // (2 * tileWidth)`

☐ `numberOfPairs = (totalWidth - tileWidth) / 2 * tileWidth`

One correct, 0 errors, 100%

•• 2. Your task is to design a program that models inflating a spherical balloon. First the balloon is inflated to have a certain diameter (which is provided as an input). Then inflate the balloon so that the diameter grows by an inch, and display the amount the volume has grown. Repeat that step: grow the diameter by another inch and show the growth of the volume. *Hint:* The volume of a sphere is $\frac{4}{3}\pi r^3$.
Assuming that the input is 10, what are the two outputs, when rounded to the nearest integer?

☒ 173 and 208.

The specified output is the *growth* of the volume after the diameter grows by an inch, not the volume itself. The volume is a function of the radius, which grows by .5 inch each time.

☐ 697 and 905.

☐ 1386 and 1663.

☐ 5575 and 7238.

One correct, 0 errors, 100%

•• 3. Rearrange the following lines of code to produce pseudocode for the task of [the preceding exercise](#).

GOOD JOB! ✓

Prompt for diameter and read user input.

`volume1 = $\pi \times \text{diameter} \times \text{diameter} \times \text{diameter} / 6$`

Increment diameter

`volume2 = $\pi \times \text{diameter} \times \text{diameter} \times \text{diameter} / 6$`

`growth = volume2 - volume1`

Print growth

Increment diameter

`volume3 = $\pi \times \text{diameter} \times \text{diameter} \times \text{diameter} / 6$`

`growth = volume3 - volume2`

Print growth

10 correct, 0 errors, 100%

Start over

Back to Page

39 / 554

Python For Everyone, Enhanced
eText
Cay S. Horstmann; Rance D. Necaise

Expand | Collapse

Title Page

Getting the Most from Your eText

Quick Reference

1. Introduction

2. Programming with Numbers and Strings

2.1 Variables

2.2 Arithmetic

2.3 Problem Solving: First Do It By Hand

2.4 Strings

2.5 Input and Output

2.6 Graphics: Simple Drawings

Chapter Summary

Interactive Review and Practice

End-of-Chapter Exercises

3. Decisions

4. Loops

5. Functions

6. Lists

7. Files and Exceptions

8. Sets and Dictionaries

9. Objects and Classes

10. Inheritance

11. Recursion

12. Sorting and Searching

Appendices

Glossary

Illustration Credits

Wiley End User License Agreement

```
print("Hello")

A string can be stored in a variable
greeting = "Hello"

and later accessed when needed just as numerical values can be:
print(greeting)

A string literal denotes a particular string
print("This is a string.", 'So is this.')
```

A **string literal** denotes a particular string (such as "Hello"), just as a **number literal** (such as 2) denotes a particular number. In Python, string literals are specified by enclosing a sequence of characters within a matching pair of either single or double quotes.

By allowing both types of delimiters, Python makes it easy to include an apostrophe or quotation mark within a string.

In this book, we use double quotation marks around strings because this is a common convention in many other programming languages. However, the interactive Python interpreter always displays strings with single quotation marks.

The `len` function returns the number of characters in a string.

The number of characters in a string is called the *length* of the string. For example, the length of "Harry" is 5. You can compute the length of a string using Python's `len` function:

```
length = len("World!") # length is 6
```

A string of length 0 is called the *empty string*. It contains no characters and is written as "" or ''.

1. What does the following code segment print?

```
text = "Python Program"
n = len(text)
print(n)
```

☐ 10

☐ 13

☒ 14

☐ 15

The space counts as a character.

One correct, 0 errors, 100%

2. For each of the following, indicate whether the statement is valid or invalid.

Valid

Invalid

msg = "Welcome!"

Valid

Invalid

A literal string can be assigned to a variable ...

Valid

Invalid

size = len("Hello")

Valid

Invalid

... or passed to a function.

Valid

Invalid

text = "ABC & 456"

Valid

Invalid

A string can contain any sequence of letters, numbers, punctuation, and spaces.

Valid

Invalid

name = John Smith

Valid

Invalid

A literal string must be enclosed within either double or single quotes. In this book, we use double quotation marks for all literal strings.

Valid

Invalid

reponse = "The result ="

Valid

Invalid

A literal string must begin and end with the same type of quotation mark.

Valid

Invalid

answer = ""

Valid

Invalid

The empty string is a string that contains no characters.

6 correct, 0 errors, 100%

Back to Page

42 / 554

Python For Everyone, Enhanced eText

Cay S. Horstmann; Rance D. Necaise

Expand | Collapse

Title Page

Getting the Most from Your eText

Quick Reference

1. Introduction

2. Programming with Numbers and Strings

2.1 Variables

2.2 Arithmetic

2.3 Problem Solving: First Do It By Hand

2.4 Strings

2.5 Input and Output

2.6 Graphics: Simple Drawings

Chapter Summary

Interactive Review and Practice

End-of-Chapter Exercises

3. Decisions

4. Loops

5. Functions

6. Lists

7. Files and Exceptions

8. Sets and Dictionaries

9. Objects and Classes

10. Inheritance

11. Recursion

12. Sorting and Searching

Appendices

Glossary

Illustration Credits

Wiley End User License Agreement

sec09_01/initials2.py

```
1 ## This program obtains two names from the user and prints a pair of initials.
2 #
3 # Obtain the two names from the user.
4 first = input("Enter your first name: ")
5 second = input("Enter your significant other's first name: ")
6 first = "Walter"
7 second = "White"
8 secretIdentity = input("Say my name")
9 secretIdentity = "Heisenberg"
10
11 # Compute and display the initials.
12 initials = first[0] + " " + second[0]
13 print(initials)
14 print("Can't change code due to vitalsource bug... First name should show as Walter, Second as White")
15 print(secretIdentity)
16
```

Input


1 Sally

2 Harry

Run Reset

Output

Enter your first name: Sally
Enter your significant other's first name: Harry
WS
Can't change code due to vitalsource bug... First name should show as Walter, Second as White
Heisenberg

 Still stuck?

1. Which of the following statements is *most* appropriate for reading a person's full name into a single variable?

☐ input("Enter full name: ")

☐ name = input()

☒ name = input("Enter full name: ")

☐ name = int(input("Enter full name: "))

One correct, 0 errors, 100%

2. Write a program that reads in the first and last name of a person and prints the last name, followed by a comma, a space, and the first name. Note the space following the colon in each of the input prompts. For example:

Please enter your first name: Harry
Please enter your last name: Morgan
Morgan, Harry

name.py

```
1 # Read in the first and last name
2 firstName = input("Please enter your first name: ")
3 lastName = input("Please enter your last name: ")
4
5 # Print the last name, a comma, a space, and the first name
6 print("%s, %s" % (lastName, firstName))
```

CodeCheck Reset

Running name.py

Test 1

Please enter your first name: Harry
Please enter your last name: Morgan
Morgan, Harry

pass

Test 2

Please enter your first name: Suzanne
Please enter your last name: Martinez
Martinez, Suzanne

pass

Score

2/2

48 / 554

Python For Everyone, Enhanced eText
Cay S. Horstmann, Rance D. Necaise

Expand | Collapse

1. Introduction 1

2. Programming with Numbers and Strings 23

2.1 Variables 24

2.2 Arithmetic 31

2.3 Problem Solving: First Do It By Hand 39

2.4 Strings 41

2.5 Input and Output 48

2.6 Graphics: Simple Drawings 58

Chapter Summary 71

Interactive Review and Practice 72

End-of-Chapter Exercises EX2.1

3. Decisions 73

4. Loops 123

5. Functions 183

6. Lists 245

7. Files and Exceptions 299

8. Sets and Dictionaries 357

9. Objects and Classes 393

10. Inheritance 443

11. Recursion 489

12. Sorting and Searching 523

Appendices A-1

Glossary B-1

Illustration Credits A-22

Wiley End User License Agreement B-22

sec06_01/window.py

```
1 # This program creates a graphics window with a rectangle. It provides the
2 # template used with all of the graphical programs used in the book..
3
4
5
6 from egraphics import GraphicsWindow
7
8 # Create the window and access the canvas.
9 win = GraphicsWindow()
10 canvas = win.canvas()
11
12 # Draw on the canvas.
13 canvas.drawRect(15, 10, 20, 30)
14
15 # Wait for the user to click the window.
16 win.wait()
```

Run Reset

Output

Images




Table 10: GraphicsWindow Methods

Method	Description
w = GraphicsWindow()	Creates a new graphics window with an empty canvas. The size of the canvas is 400 x 400 unless another size is specified.
w = GraphicsWindow(width, height)	Creates a new graphics window with an empty canvas. The size of the canvas is width x height unless another size is specified.
w.canvas()	Returns the object representing the canvas contained in the graphics window.
w.wait()	Keeps the graphics window open and waits for the user to click the "close" button.

1. Which of the following statements will create a graphics window that contains a canvas 300 pixels wide and 300 pixels high?

☒ win = GraphicsWindow(300, 300)

If you use win = GraphicsWindow(), the canvas is 400 x 400.

☐ win = GraphicsWindow(300, 300)

☐ win = GraphicsWindow()

☐ win = GraphicsWindow(width, height)

One correct, 0 errors, 100%

2. Rearrange the following lines of code to produce a program that creates a graphics window containing a simple drawing.

GOOD JOB!

```
from egraphics import GraphicsWindow
win = GraphicsWindow(400, 400)
canvas = win.canvas()
canvas.drawText(50, 50, "My Drawing")
canvas.drawRect(10, 10, 50, 200)
canvas.drawLine(200, 200, 400, 200)
win.wait()
```

7 correct, 0 errors, 100%

Start over

2.6

2.6 - Graphics: Simple Drawings (Optional)