Chapter Two

PROGRAMMING WITH NUMBERS AND STRINGS

Introduction

- Numbers and character strings are important data types in any Python program
  - These are the fundamental building blocks we use to build more complex data structures
- In this chapter, you will learn how to work with numbers and text.
  We will write several simple programs that use them

Chapter Goals

- To declare and initialize variables and constants
- To understand the properties and limitations of integers and floating-point numbers
- To appreciate the importance of comments and good code layout
- To write arithmetic expressions and assignment statements
- To create programs that read, and process inputs, and display the results
- To learn how to use Python strings
- To create simple graphics programs using basic shapes and text

Contents

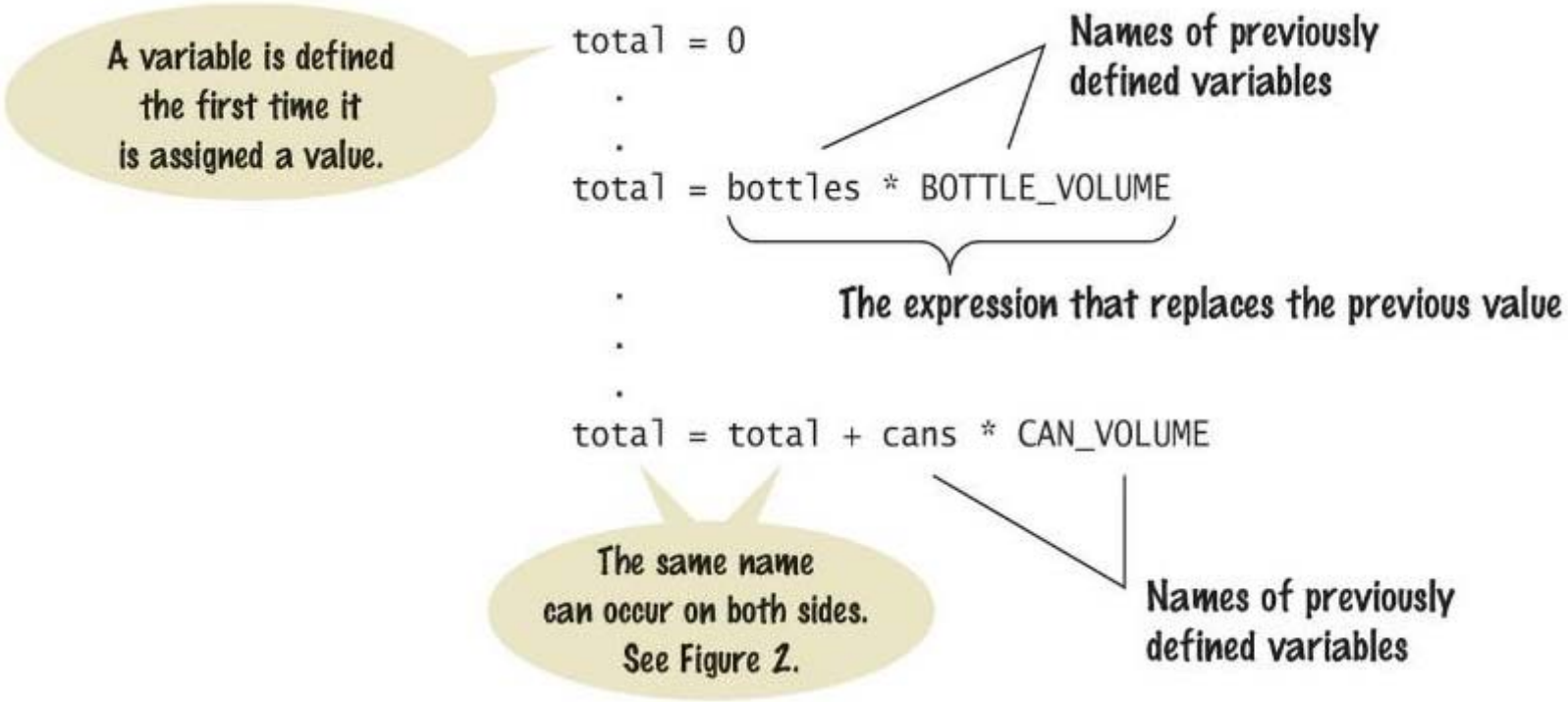
2.1 Variables

Variables

- A variable is a named storage location in a computer program
- There are many different types of variables, each type used to store different things
- You ‘define’ a variable by telling the compiler:
  - What name you will use to refer to it
  - The initial value of the variable
- You use an assignment statement to place a value into a variable

Variable Definition

- To define a variable, you must specify an initial value.



The assignment statement

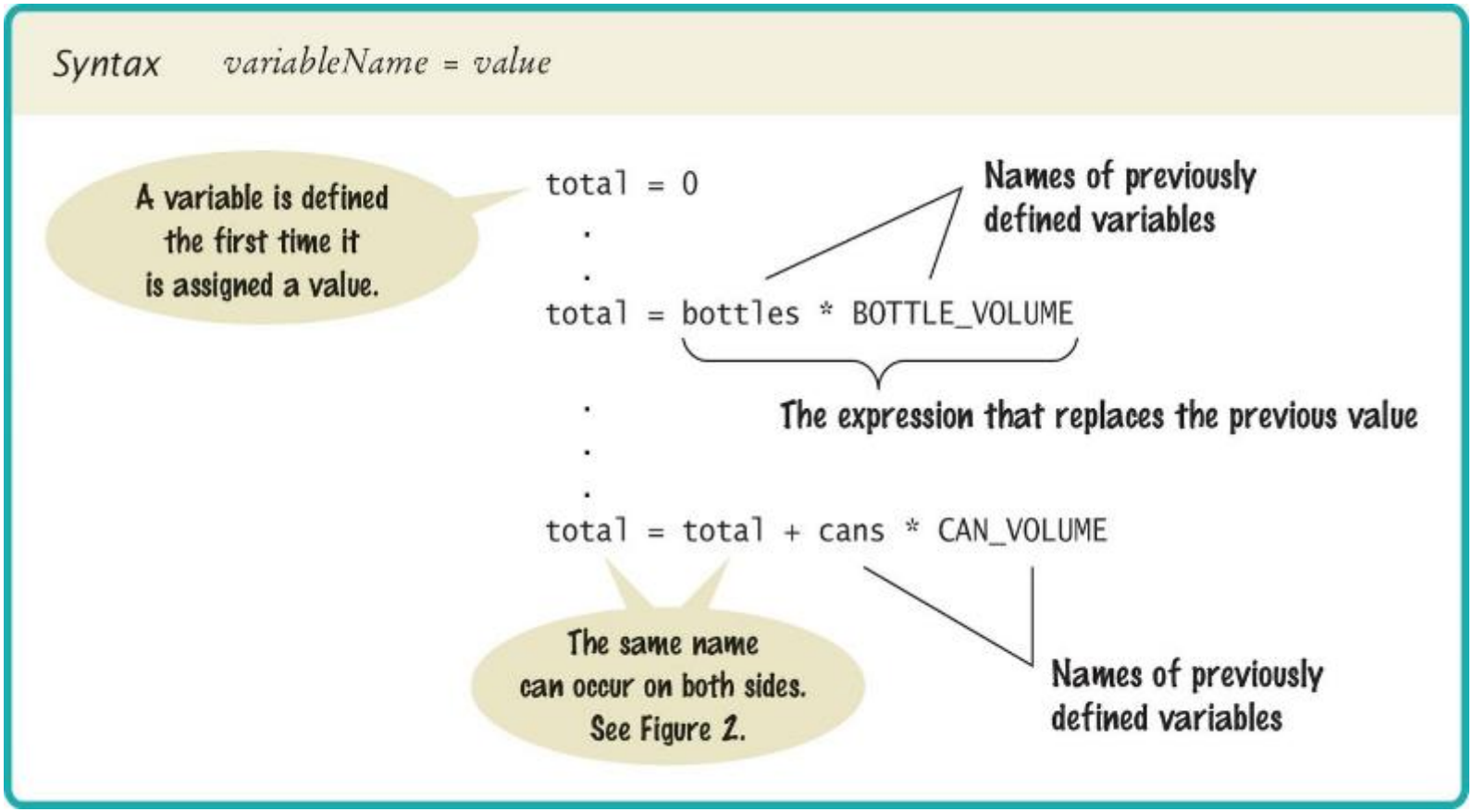
- Use the **assignment statement** '=' to place a new value into a variable

```
cansPerPack = 6   # define & initializes the variable cansPerPack
```

- Beware: The "=" sign is NOT used for comparison:
  - It copies the value on the right side into the variable on the left side
  - You will learn about the comparison operator in the next chapter

Assignment syntax

- The value on the right of the '=' sign is assigned to the variable on the left



An example: soda deal

- Soft drinks are sold in cans and bottles. A store offers a six-pack of 12-ounce cans for the same price as a two-liter bottle. Which should you buy? (12 fluid ounces equal approximately 0.355 liters.)

List of variables:	Type of Number
Number of cans per pack	Whole number
Ounces per can	Whole number
Ounces per bottle	Number with fraction

Why different types?

- There are three different types of data that we will use in this chapter:

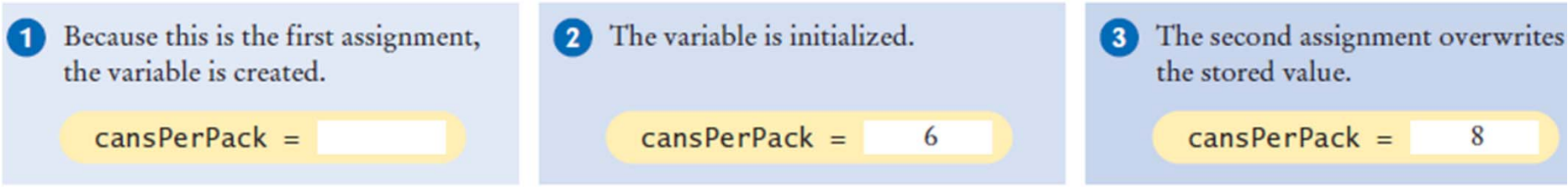
1.	A whole number (no fractional part)	`7`	`(integer or int)`
2.	A number with a fraction part	`8.88`	`(float)`
3.	A sequence of characters	`"Bob”`	`(string)`

- The data type is associated with the **value**, not the **variable**:

```
cansPerPack = 6     # int
canVolume = 12.0    # float
```

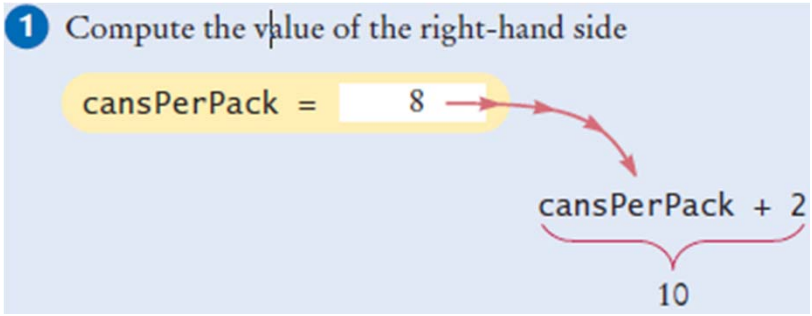
Updating a Variable (assigning a value)

- If an existing variable is assigned a new value, that value replaces the previous contents of the variable.
- For example:
  - cansPerPack = 6
  - cansPerPack = 8

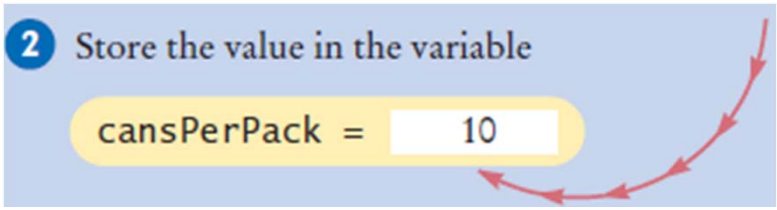


Updating a Variable (computed)

- Executing the Assignment:
  cansPerPack = cansPerPack + 2
- Step by Step:
- Step 1: Calculate the right hand side of the assignment. Find the value of cansPerPack, and add 2 to it.



- Step 2: Store the result in the variable named on the left side of the assignment operator



A Warning…

- Since the data type is associated with the value and not the variable:
  - A variable can be assigned different values at different places in a program

taxRate = 5 # an **int**

Then later…

```
taxRate = 5.5              # a float
```

And then

```
taxRate = “Non- taxable” # a string
```

- If you use a variable and it has an unexpected type an error will occur in your program

Table 1: Number Literals in Python

Table 1 Number Literals in Python

Number	Type	Comment
`6`	`int`	An integer has no fractional part.
`-6`	`int`	Integers can be negative.
`0`	`int`	Zero is an integer.
`0.5`	`float`	A number with a fractional part has type `float`.
`1.0`	`float`	An integer with a fractional part .0 has type `float`.
`1E6`	`float`	A number in exponential notation: 1×10^6 or 1000000. Numbers in exponential notation always have type `float`.
`2.96E-2`	`float`	Negative exponent: $2.96 \times 10^{-2} = 2.96 / 100 = 0.0296$
`100,000`		**Error:** Do not use a comma as a decimal separator.
`3 1/2`		**Error:** Do not use fractions; use decimal notation: 3.5.

Naming variables

- Variable names should describe the purpose of the variable
  - ‘canVolume’ is better than ‘cv’
- Use These Simple Rules
  1. Variable names must start with a letter or the underscore (_) character
     1. Continue with letters (upper or lower case), digits or the underscore
  2. You cannot use other symbols (? or %...) and spaces are not permitted
  3. Separate words with ‘camelCase’ notation
     1. Use upper case letters to signify word boundaries
  4. Don’t use ‘reserved’ Python words (see Appendix C, pages A6 and A7)

Table 2: Variable Names in Python

Table 2 Variable Names in Python

Variable Name	Comment
`canVolume1`	Variable names consist of letters, numbers, and the underscore character.
`x`	In mathematics, you use short variable names such as *x* or *y*. This is legal in Python, but not very common, because it can make programs harder to understand (see Programming Tip 2.1 on page 36).
⚠ `CanVolume`	**Caution:** Variable names are case sensitive. This variable name is different from `canVolume`, and it violates the convention that variable names should start with a lowercase letter.
⊘ `6pack`	**Error:** Variable names cannot start with a number.
⊘ `can volume`	**Error:** Variable names cannot contain spaces.
⊘ `class`	**Error:** You cannot use a reserved word as a variable name.
⊘ `ltr/fl.oz`	**Error:** You cannot use symbols such as / or.

Programming Tip: Use Descriptive Variable Names

- Choose descriptive variable names
- Which variable name is more self descriptive?

```
canVolume = 0.35
```

```
cv = 0.355
```

- This is particularly important when programs are written by more than one person.

constants

- In Python a **constant** is a variable whose value ***should not*** be changed after it's assigned an initial value.
  - It is a good practice to use all caps when naming constants

```
BOTTLE_VOLUME = 2.0
```

- It is good style to use named constants to explain numerical values to be used in calculations
  - Which is clearer?

```
totalVolume = bottles * 2
```

```
totalVolume = bottles * BOTTLE_VOLUME
```

- A programmer reading the first statement may not understand the significance of the "2"
- Python will let you change the value of a **constant**
  - Just because you can do it, doesn't mean you should do it

Constants: Naming & Style

- It is customary to use all UPPER_CASE letters for constants to distinguish them from variables.
  - It is a nice visual way cue

```
BOTTLE_VOLUME = 2    # Constant
MAX_SIZE = 100       # Constant
taxRate = 5          # Variable
```

Python comments

- Use comments at the beginning of each program, and to clarify details of the code
- Comments are a courtesy to others and a way to document your thinking
  - Comments to add explanations for humans who read your code.
- The compiler ignores comments.

Commenting Code: 1st Style

```
##
#  This program computes the volume (in liters) of a six-pack of soda
#  cans and the total volume of a six-pack and a two-liter bottle
#

# Liters in a 12-ounce can
CAN_VOLUME = 0.355

# Liters in a two-liter bottle.
BOTTLE_VOLUME = 2

# Number of cans per pack.
cansPerPack = 6

# Calculate total volume in the cans.
totalVolume = cansPerPack * CAN_VOLUME
print("A six-pack of 12-ounce cans contains", totalVolume, "liters.")

# Calculate total volume in the cans and a 2-liter bottle.
totalVolume = totalVolume + BOTTLE_VOLUME
print("A six-pack and a two-liter bottle contain", totalVolume,
"liters.")
```

Commenting Code: 2nd Style

```
##
#  This program computes the volume (in liters) of a six-pack of soda
#  cans and the total volume of a six-pack and a two-liter bottle
#
## CONSTANTS ##
CAN_VOLUME = 0.355   # Liters in a 12-ounce can
BOTTLE_VOLUME = 2     # Liters in a two-liter bottle

# Number of cans per pack.
cansPerPack = 6

# Calculate total volume in the cans.
totalVolume = cansPerPack * CAN_VOLUME
print("A six-pack of 12-ounce cans contains", totalVolume, "liters.")

# Calculate total volume in the cans and a 2-liter bottle.
totalVolume = totalVolume + BOTTLE_VOLUME
print("A six-pack and a two-liter bottle contain", totalVolume,
"liters.")
```

Undefined Variables

- You must define a variable before you use it: (i.e. it must be defined somewhere above the line of code where you first use the variable)

```
canVolume = 12 * literPerOunce
literPerOunce = 0.0296
```

- The correct order for the statements is:

```
literPerOunce = 0.0296
canVolume = 12 * literPerOunce
```

2.2 Arithmetic

Basic Arithmetic Operations

- Python supports all of the basic arithmetic operations:
  - Addition "+"
  - Subtraction "-"
  - Multiplication "*"
  - Division "/"
- You write your expressions a bit differently

$$\frac{a+b}{2}$$

```
(a + b) / 2
```

Precedence

- Precedence is similar to Algebra:
  - PEMDAS
    - Parenthesis, Exponent, Multiply/Divide, Add/Subtract

Mixing numeric types

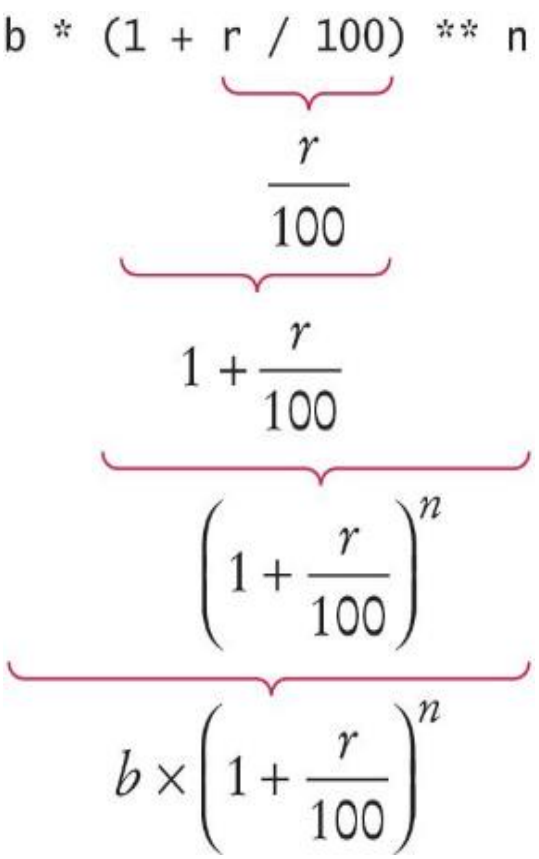
- If you mix integer and floating-point values in an arithmetic expression, the result is a floating-point value.
- 7 + 4.0 # Yields the floating value 11.0
- Remember from our earlier example:
  - If you mix stings with integer or floating point values the result is an error

Powers

- Double stars ** are used to calculate an exponent
- Analyzing the expression:

$$b \times \left(1 + \frac{r}{100}\right)^n$$

- Becomes:
  - b * ((1 + r / 100) ** n)



Floor division

- When you divide two integers with the / operator, you get a floating-point value. For example,

7 / 4

- Yields 1.75
- We can also perform **floor division** using the // operator.
  - The "//" operator computes the quotient and discards the fractional part

7 // 4

- Evaluates to 1 because 7 divided by 4 is 1.75 with a fractional part of 0.75, which is discarded.

Calculating a remainder

- If you are interested in the remainder of dividing two integers, use the "%" operator (called modulus):

remainder = 7 % 4

- The value of remainder will be 3
- Sometimes called modulo divide

A Simple Example:

- Open a new file the IDE:
- Type in the following:

```
# Convert pennies to dollars and cents
pennies = 1729
dollars = pennies // 100  # Calculates the number of dollars
cents = pennies % 100     # Calculates the number of pennies
print("I have", dollars, "and", cents, "cents")
```

- Save the file
- Run the file
- What is the result?

Integer Division and Remainder Examples

Table 3 Floor Division and Remainder

Expression (where n = 1729)	Value	Comment
`n % 10`	9	For any positive integer n, `n % 10` is the last digit of n.
`n // 10`	172	This is n without the last digit.
`n % 100`	29	The last two digits of n.
`n % 2`	1	`n % 2` is 0 if n is even, 1 if n is odd (provided n is not negative)
`-n // 10`	-173	–173 is the largest integer ≤ –172.9. We will not use floor division for negative numbers in this book.

Calling functions

- Recall that a function is a collection of programming instructions that carry out a particular task.
- The print() function can display information, but there are many other functions available in Python.
- When calling a function you must provide the correct number of arguments
  - The program will generate an error message if you don't

Calling functions that return a value

- Most functions return a value. That is, when the function completes its task, it passes a value back to the point where the function was called.
- For example:
  - The call abs(-173) returns the value 173.
  - The value returned by a function can be stored in a variable:
    - distance = abs(x)
- You can use a function call as an argument to the **print** function
- For instance:

```
print(abs(-173))
```

Built-in Functions

- **Built-in** functions are a small set of functions that are defined as a part of the Python language
- They can be used without importing any modules
- For example, Floating-point to integer conversion

You can use the function int() and float() to convert between integer and floating point values:

```
balance = total + tax   # balance: float
dollars = int (balance) # dollars: integer
```

You lose the fractional part of the floating-point value (no rounding occurs)

So int(10.6) and int(10.3) both return 10

Built in Mathematical Functions

Table 4 Built-in Mathematical Functions

Function	Returns
`abs(`x`)`	The absolute value of x.
`round(`x`)` `round(`x`, `n`)`	The floating-point value x rounded to a whole number or to n decimal places.
`max(`x_1`, `x_2`, ..., `x_n`)`	The largest value from among the arguments.
`min(`x_1`, `x_2`, ..., `x_n`)`	The smallest value from among the arguments.

Python libraries (modules)

- A **library** is a collection of code, written and compiled by someone else, that is ready for you to use in your program
- A **standard library** is a library that is considered part of the language and must be included with any Python system.
- Python's standard library is organized into **modules**.
  - Related functions and data types are grouped into the same module.
  - Functions defined in a module must be explicitly loaded into your program before they can be used.

Using functions from the Math Module

- For example, to use the sqrt() function, which computes the square root of its argument:

```
# First include this statement at the top of your
# program file.
from math import sqrt

# Then you can simply call the function as
y = sqrt(x)
```

Functions from the Math Module

Table 5 Selected Functions in the `math` Module

Function	Returns
`sqrt(`x`)`	The square root of x. ($x \geq 0$)
`trunc(`x`)`	Truncates floating-point value x to an integer.
`cos(`x`)`	The cosine of x in radians.
`sin(`x`)`	The sine of x in radians.
`tan(`x`)`	The tangent of x in radians.
`exp(`x`)`	e^x
`degrees(`x`)`	Convert x radians to degrees (i.e., returns $x \cdot 180/\pi$)
`radians(`x`)`	Convert x degrees to radians (i.e., returns $x \cdot \pi/180$)
`log(`x`)` `log(`x`, `*base*`)`	The natural logarithm of x (to base e) or the logarithm of x to the given *base*.

Arithmetic Expressions

Table 6 Arithmetic Expression Examples

Mathematical Expression	Python Expression	Comments
$\frac{x + y}{2}$	`(x + y) / 2`	The parentheses are required; `x + y / 2` computes $x + \frac{y}{2}$.
$\frac{xy}{2}$	`x * y / 2`	Parentheses are not required; operators with the same precedence are evaluated left to right.
$\left(1 + \frac{r}{100}\right)^n$	`(1 + r / 100) ** n`	The parentheses are required.
$\sqrt{a^2 + b^2}$	`sqrt(a ** 2 + b ** 2)`	You must import the `sqrt` function from the `math` module.
π	`pi`	`pi` is a constant declared in the `math` module.

Roundoff Errors

- Floating point values are not exact
  - This is a limitation of binary values; not all floating point numbers have an exact representation

- Open a new file and type in:

```
price = 4.35
quantity = 100
total = price * quantity
# Should be 100 * 4.35 = 435.00
print(total)
```

- You can deal with roundoff errors by
  - rounding to the nearest integer (see Section 2.2.4)
  - or by displaying a fixed number of digits after the decimal separator (see Section 2.5.3).

Unbalanced Parentheses

- Consider the expression
  `((a + b) * t / 2 * (1 - t)`
  - What is wrong with the expression?
- Now consider this expression.
  `(a + b) * t) / (2 * (1 - t)`
  - This expression has three "(" and three ")", but it still is not correct
- At any point in an expression the count of "(" must be greater than or equal to the count of ")"
- At the end of the expression the two counts must be the same

Additional Programming Tips

- Use Spaces in expressions

```
totalCans = fullCans + emptyCans
```

- Is easier to read than

```
totalCans=fullCans+emptyCans
```

- Other ways to import modules:

```
From math import, sqrt, sin, cos  # imports the functions listed
From math import *  # imports all functions from the module
Import math # imports all functions from the module
```

- If you use the last style you have to add the module name and a "." before each function call

```
y = math.sqrt(x)
```

2.4 Strings

Strings

- Start with some simple definitions:
  - Text consists of **characters**
  - **Characters** are letters, numbers, punctuation marks, spaces, ….
  - A **string** is a sequence of **characters**
- In Python, string literals are specified by enclosing a sequence of **characters** within a matching pair of either single or double quotes.

```
print("This is a string.", 'So is this.')
```

- By allowing both types of delimiters, Python makes it easy to include an apostrophe or quotation mark within a string.
  - message = 'He said "Hello"‘
  - Remember to use matching pairs of quotes, single with single, double with double

String Length

- 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 ''.

String Concatenation (“+”)

- You can ‘add’ one String onto the end of another

```
firstName = "Harry"
lastName = "Morgan"
name  = firstName + lastName  # HarryMorgan
print(“my name is:”, name)
```

- You wanted a space in between the two names?

```
name = firstName + " " + lastName  # Harry Morgan
```

Using “+” to concatenate strings is an example of a concept called operator overloading. The “+” operator performs different functions of variables of different types

String repetition (“*”)

- You can also produce a string that is the result of repeating a string multiple times.
- Suppose you need to print a dashed line.
- Instead of specifying a literal string with 50 dashes, you can use the * operator to create a string that is comprised of the string "-" repeated 50 times.

dashes = "-" * 50

- results in the string
- "--“

The “*” operator is also overloaded.

Converting Numbers to Strings

- Use the str() function to convert between numbers and strings.
- Open a new file and type in:

```
balance = 888.88
dollars = 888
balanceAsString = str(balance)
dollarsAsString = str(dollars)
print(balanceAsString)
print(dollarsAsString)
```

- To turn a string containing a number into a numerical value, we use the int() and float() functions:

```
id = int("1729")
price = float("17.29")
print(id)
print(price)
```

- **This conversion is important when the strings come from user input**.

Strings and Characters

- **strings** are sequences of **characters**
  - Python uses **Unicode** characters
    - **Unicode** defines over 100,000 characters
    - **Unicode** was designed to be able to encode text in essentially all written languages
  - Characters are stored as integer values
    - See the ASCII subset on Unicode chart in Appendix A
    - For example, the letter ‘H’ has a value of 72

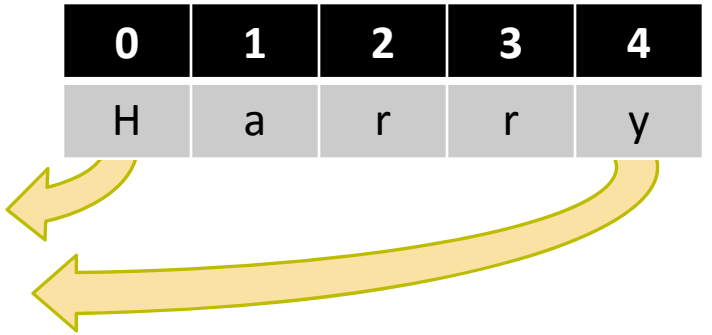
Copying a character from a String

- Each char inside a String has an index number:

0	1	2	3	4	5	6	7	8	9
c	h	a	r	s		h	e	r	e

- The first char is index zero (0)
- The [] operator returns a char at a given index inside a String:

```
name = "Harry”
start = name[0]
last = name[4]
```



String Operations

Table 7 String Operations

Statement	Result	Comment
`string = "Py"` `string = string + "thon"`	`string` is set to `"Python"`	When applied to strings, + denotes concatenation.
`print("Please" +` `" enter your name: ")`	Prints `Please enter your name:`	Use concatenation to break up strings that don't fit into one line.
`team = str(49) + "ers"`	`team` is set to `"49ers"`	Because 49 is an integer, it must be converted to a string.
`greeting = "H & S"` `n = len(greeting)`	`n` is set to 5	Each space counts as one character.
`string = "Sally"` `ch = string[1]`	`ch` is set to `"a"`	Note that the initial position is 0.
`last = string[len(string) - 1]`	`last` is set to the string containing the last character in `string`	The last character has position `len(string) - 1`.

Methods

- In computer programming, an object is a software entity that represents a value with certain behavior.
  - The value can be simple, such as a string, or complex, like a graphical window or data file.
- The behavior of an object is given through its **methods**.
  - A method is a collection of programming instructions to carry out a specific task – similar to a function
- But unlike a **function**, which is a standalone operation, a **method** can only be applied to an object of the type for which it was defined.
  - Methods are specific to a type of object
  - Functions are general and can accept arguments of different types
- You can apply the upper() method to any string, like this:
  - name = "John Smith"
  - # Sets uppercaseName to "JOHN SMITH"
  - uppercaseName = name.upper()

Some Useful String Methods

Table 8 Useful String Methods

Method	Returns
`s.lower()`	A lowercase version of string *s*.
`s.upper()`	An uppercase version of *s*.
`s.replace(`*old*`, `*new*`)`	A new version of string *s* in which every occurrence of the substring *old* is replaced by the string *new*.

String Escape Sequences

- How would you print a double quote?
  - Preface the " with a “\” inside the double quoted String

```
print("He said \"Hello\"")
```

- OK, then how do you print a backslash?
  - Preface the \ with another \

```
System.out.print("“C:\\Temp\\Secret.txt“")
```

- Special characters inside Strings
  - Output a newline with a ‘\n’

```
print("*\n**\n***\n")
```

```
*
**
***
```

2.5 Input and Output

Input and Output

- You can read a String from the console with the input() function:
  - name = input("Please enter your name")
- Converting a String variable to a number can be used if numeric (rather than string input) is needed
  - age = int(input("Please enter age: "))
  - The above is equivalent to doing it two steps (getting the input and then converting it to a number):
  - aString = input("Please enter age: ") # String input
  - age = int(aString) # Converted to
  - # int

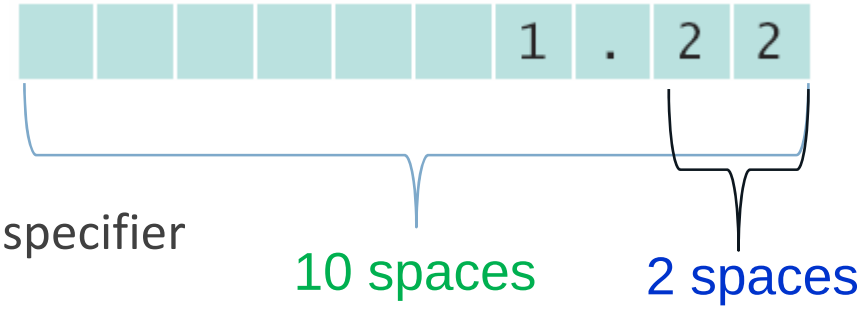
Formatted output

- Outputting floating point values can look strange:

```
Price per liter:  1.21997
```

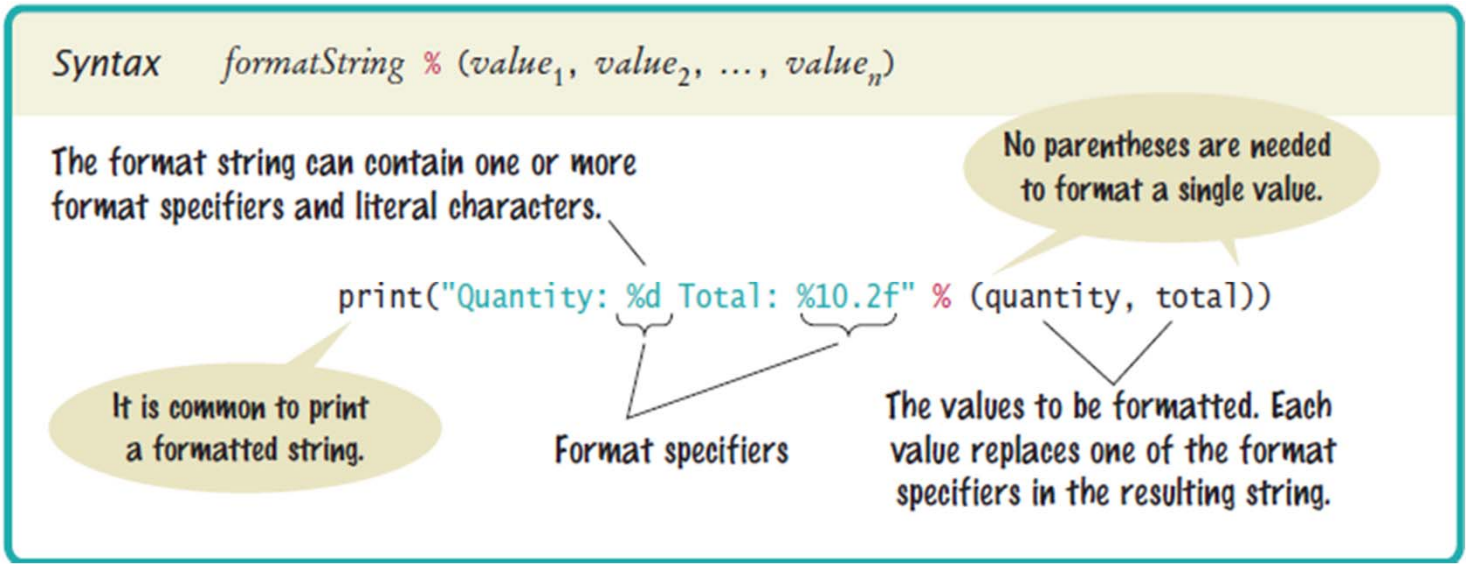
- To control the output appearance of numeric variables, use formatted output tools such as:

```
print("Price per liter %.2f"  %(price))
Price per liter: 1.22
print("Price per liter %10.2f"  %(price))
Price per liter:       1.22
```



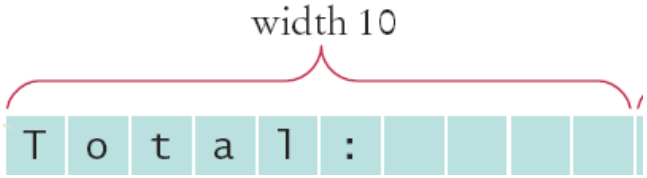
- The %10.2f is called a format specifier

Syntax: formatting strings

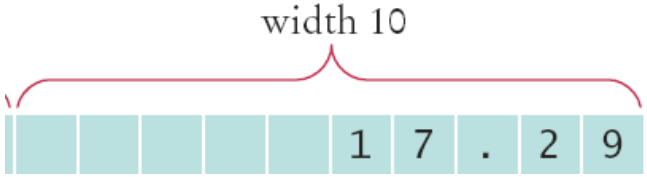


Format flag examples

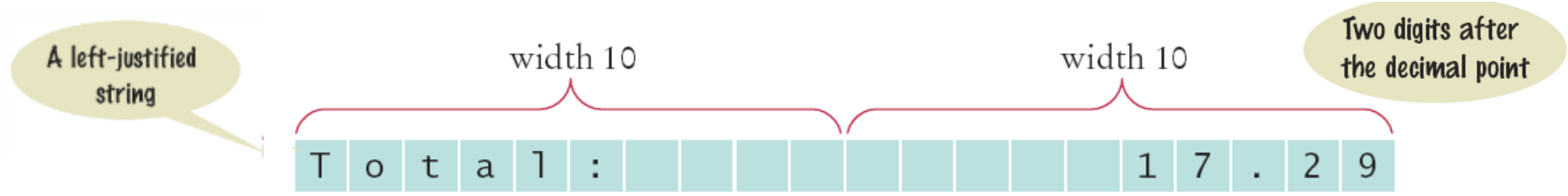
- Left Justify a String:
- print("%-10s" %("Total:"))



- Right justify a number with two decimal places
  - print("%10.2f" %(price))



- And you can print multiple values:
  - print("%-10s%10.2f" %("Total: ", price))



Volume2.py

ch02/volume2.py

```
1  ##
2  #  This program prints the price per ounce for a six-pack of cans.
3  #
4
5  # Define constant for pack size.
6  CANS_PER_PACK = 6
7
8  # Obtain price per pack and can volume.
9  userInput = input("Please enter the price for a six-pack: ")
10 packPrice = float(userInput)
11
12 userInput = input("Please enter the volume for each can (in ounces): ")
13 canVolume = float(userInput)
14
15 # Compute pack volume.
16 packVolume = canVolume * CANS_PER_PACK
17
18 # Compute and print price per ounce.
19 pricePerOunce = packPrice / packVolume
20 print("Price per ounce: %8.2f" % pricePerOunce)
```

Format Specifier Examples

Table 9 Format Specifier Examples

Format String	Sample Output	Comments
`"%d"`	`24`	Use `d` with an integer.
`"%5d"`	`   24`	Spaces are added so that the field width is 5.
`"%05d"`	`00024`	If you add `0` before the field width, zeroes are added instead of spaces.
`"Quantity:%5d"`	`Quantity:   24`	Characters inside a format string but outside a format specifier appear in the output.
`"%f"`	`1.219977`	Use `f` with a floating-point number.
`"%.2f"`	`1.22`	Prints two digits after the decimal point.
`"%7.2f"`	`   1.22`	Spaces are added so that the field width is 7.
`"%s"`	`Hello`	Use `s` with a string.
`"%d %.2f"`	`24 1.22`	You can format multiple values at once.
`"%9s"`	`    Hello`	Strings are right-justified by default.
`"%-9s"`	`Hello    `	Use a negative field width to left-justify.
`"%d%%"`	`24%`	To add a percent sign to the output, use `%%`.

2.6 Graphics (Optional)

SIMPLE DRAWINGS

Drawing Simple Graphics

- To help you create simple drawings, we have included a graphics module with the book that is a simplified version of Python's more complex library module.
- The module code and usage instructions are available with the source code for the book on its companion web site.

Using the graphics module (1)

- To create a graphical application using the graphics module, carry out the following at the top of your program:

```
from graphics import GraphicsWindow
```

- Create a graphics window (640 x 480 pixels):

```
win = GraphicsWindow(640, 480)
```

- Access the canvas contained in the graphics window:

```
canvas = win.canvas()
```

Using the graphics module (2)

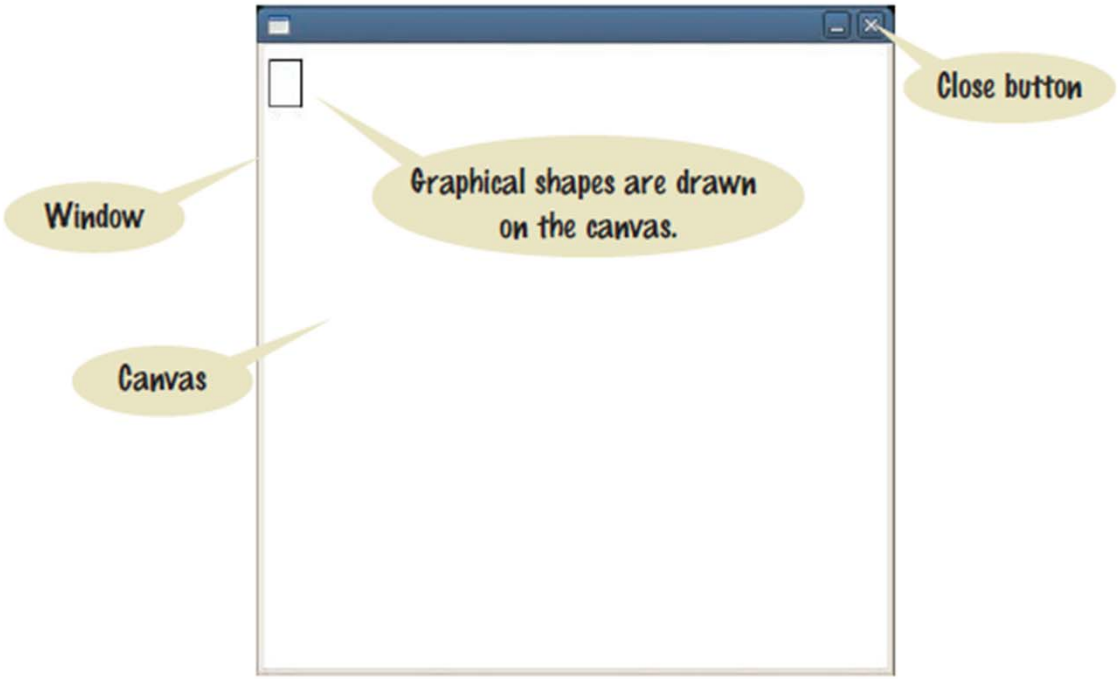
- Create your drawing.

```
canvas.drawRect(15, 10, 20, 30)
```

- Have the program wait for the user to close the window (by clicking the close button).
  - Without this statement, the program would terminate immediately and the graphics window would disappear, leaving no time for you to see your drawing.

```
win.wait()
```

A graphics window



A complete drawing example

ch02/window.py

```
1  ##
2  #  This program creates a graphics window with a rectangle. It provides the
3  #  template used with all of the graphical programs used in the book.
4  #
5
6  from graphics 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(5, 10, 20, 30)
14
15 # Wait for the user to close the window.
16 win.wait()
```

Table 10: GraphicsWindow Methods

Table 10 GraphicsWindow Methods

Method	Description
`w = GraphicsWindow()` `w = GraphicsWindow(`*width*`, `*height*`)`	Creates a new graphics window with an empty canvas. The size of the canvas is 400 × 400 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.

Drawing shapes

- Basic shapes have 4 properties: **x coordinate**, **y coordinate**, **width** and **height**.
- Example:

```
canvas.drawRect(15, 10, 20, 30)
```

- This statement draws a rectangle with the upper top left corner at point (x = 15, y = 10) in the window with a height of 20 and a width of 30.
- Common shapes that can be drawn include: rectangles, squares, circles and ovals.

Drawing lines

- Lines require 4 slightly different properties than drawing shapes:
  - Point 1(x coordinate, y coordinate)
  - Point 2(x coordinate, y coordinate)

Table 13: Common Shapes, Lines and Text

Table 13 GraphicsCanvas Drawing Methods

Method	Result	Notes
`c.drawLine(`x_1, y_1, x_2, y_2`)`		(x_1, y_1) and (x_2, y_2) are the endpoints.
`c.drawRect(`*x*, *y*, *width*, *height*`)`		(x, y) is the top left corner.
`c.drawOval(`*x*, *y*, *width*, *height*`)`		(x, y) is the top-left corner of the box that bounds the ellipse. To draw a circle, use the same value for *width* and *height*.
`c.drawText(`*x*, *y*, *text*`)`		(x, y) is the anchor point.

The canvas and shapes can be colored

- If you use the default setting (not changing the fill or outline), shapes are outlined in black and there is no fill color.
- The fill color and outline can be set to different colors with the method calls:

```
setFill(<color name>)
```

OR

```
setFill(<red level>, <green level>, <blue level>)
```

```
setOutline(<color name>)
```

OR

```
setOutline(<red level>, <green level>, <blue level>)
```

Example of setting color

- The following statements draw a rectangle that is outlined in black and filled with green.

```
canvas.setOutline("black")
canvas.setFill(0, 255, 0)
canvas.drawRect(10, 20, 100, 50)
```

Table 11: Common Color Names

Table 11 Common Color Names			
Color Name	Color Name	Color Name	Color Name
"black"	"magenta"	"maroon"	"pink"
"blue"	"yellow"	"dark blue"	"orange"
"red"	"white"	"dark red"	"sea green"
"green"	"gray"	"dark green"	"light gray"
"cyan"	"gold"	"dark cyan"	"tan"

Table 12: GraphicsCanvas Color Methods

Table 12 GraphicsCanvas Color Methods

Method	Description
`c.setColor(`*name*`)` `c.setColor(`*red*`, `*green*`, `*blue*`)`	Sets both the fill and outline color to the same color. Color can be set by the color's *name* or by values for its *red*, *green*, and *blue* components.

`c.setFill()`
`c.setFill(`*name*`)`
`c.setFill(`*red*`, `*green*`, `*blue*`)`
	Sets the color used to fill a geometric shape. If no argument is given, the fill color is cleared.

`c.setOutline()`
`c.setOutline(`*name*`)`
`c.setOutline(`*red*`, `*green*`, `*blue*`)`
	Sets the color used to draw lines and text. If no argument is given, the outline color is cleared.

Summary: variables

- A variable is a storage location with a name.
- When defining a variable, you must specify an initial value.
- By convention, variable names should start with a lower case letter.
- An assignment statement stores a new value in a variable, replacing the previously stored value.

Summary: operators

- The assignment operator = does not denote mathematical equality.
- Variables whose initial value should not change are typically capitalized by convention.
- The / operator performs a division yielding a value that may have a fractional value.
- The // operator performs a division, the remainder is discarded.
- The % operator computes the remainder of a floor division.

Summary: python overview

- The Python library declares many mathematical functions, such as sqrt() and abs()
- You can convert between integers, floats and strings using the respective functions: int(), float(), str()
- Python libraries are grouped into modules. Use the import statement to use methods from a module.
- Use the input() function to read keyboard input in a console window.

Summary: python overview

- Use the format specifiers to specify how values should be formatted.

Summary: Strings

- Strings are sequences of characters.
- The len() function yields the number of characters in a String.
- Use the + operator to concatenate Strings; that is, to put them together to yield a longer String.
- In order to perform a concatenation, the + operator requires both arguments to be strings. Numbers must be converted to strings using the str() function.
- String index numbers are counted starting with 0.

Summary: Strings

- Use the [ ] operator to extract the elements of a String.