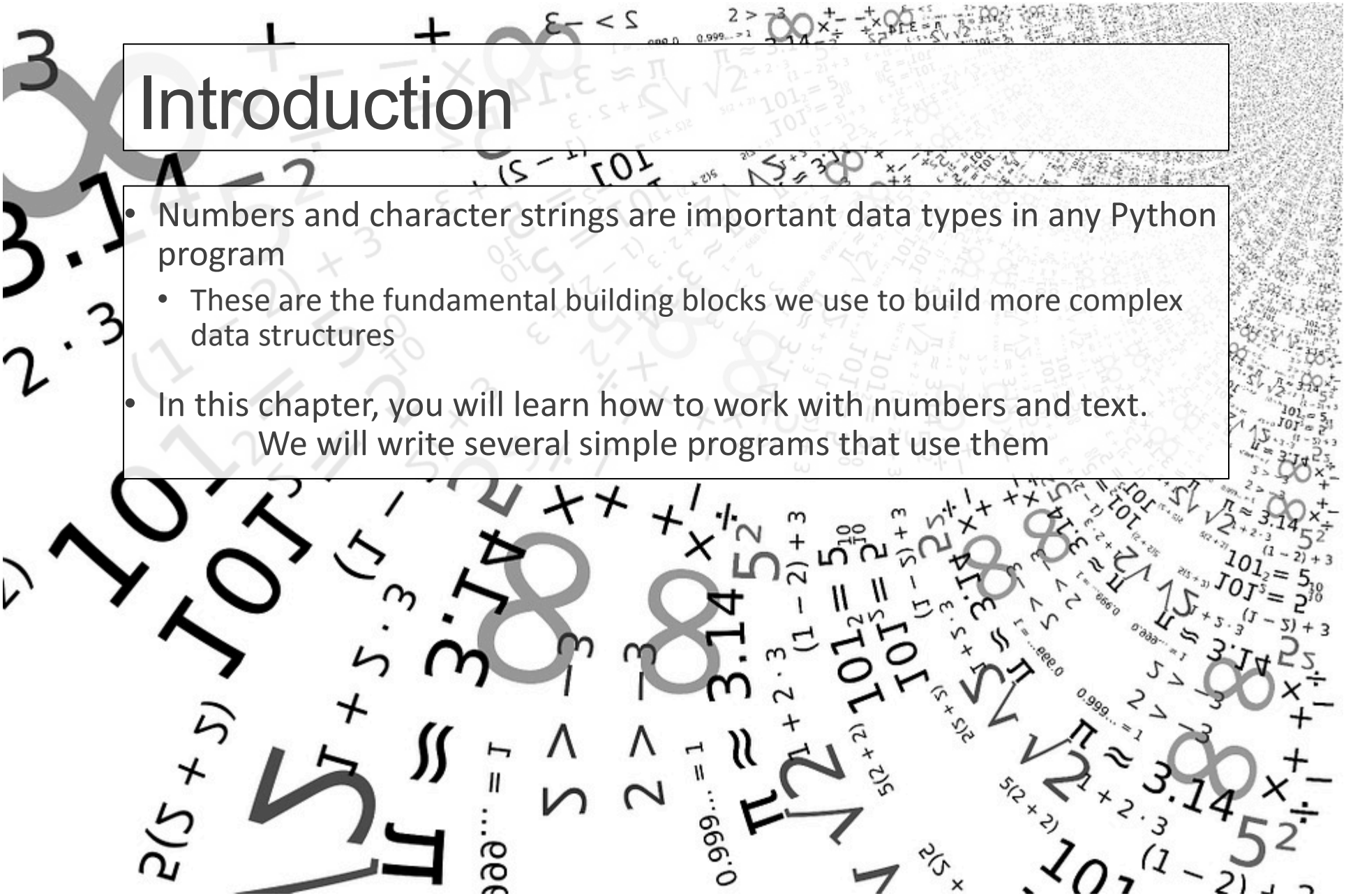
Chapter Two

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.

A variable is defined the first time it is assigned a value.

```
total = 0
   .
   .
total = bottles * BOTTLE_VOLUME
   .
   .
   .
total = total + cans * CAN_VOLUME
```

Names of previously defined variables

The expression that replaces the previous value

The same name can occur on both sides. See Figure 2.

Names of previously defined variables

The assignment statement

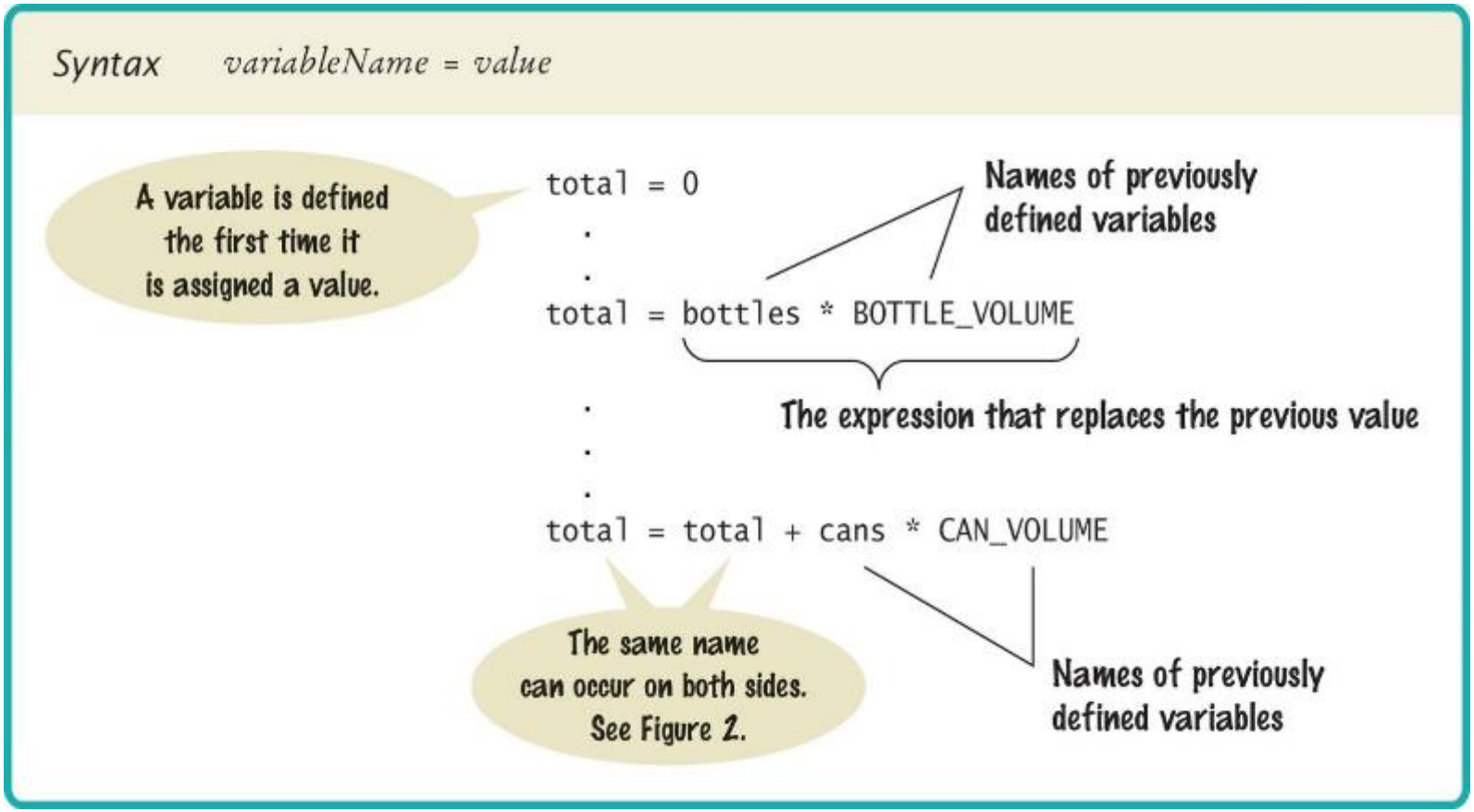
- Use the **assignment statement** '=' to place a new value into a variable

```
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

1 Because this is the first assignment, the variable is created.

```
cansPerPack =
```

2 The variable is initialized.

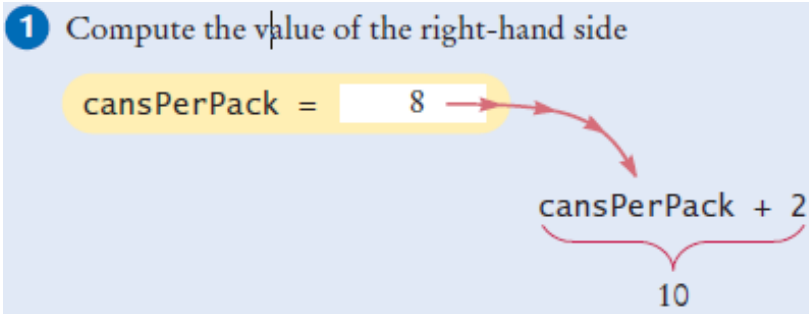
```
cansPerPack = 6
```

3 The second assignment overwrites the stored value.

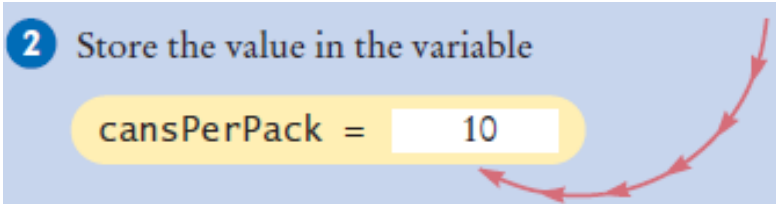
```
cansPerPack = 8
```

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

Our First Program of the Day…

- Open PyCharm (our IDE) and create a new file
  - type in the following
  - save the file as typetest.py
  - Run the program

```
# Testing different types in the same variable
taxRate = 5  # int
print(taxRate)
taxrate = 5.5  # float
print(taxRate)
taxRate = "Non-taxable" # string
print(taxRate)
print(taxRate + 5)
```

- So…
  - Once you have initialized a variable with a value of a particular type you should take great care to keep storing values of the same type in the variable

A Minor Change

- Change line 8 to read:

```
print(taxRate + “??”)
```

- Save your changes
- Run the program

- What is the result?

- When you use the “+” operator with strings the second argument is concatenated to the end of the first
  - We’ll cover string operations in more detail later in this chapter

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)

```
b * (1 + r / 100) ** n
```

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

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 in the PyCharm 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.