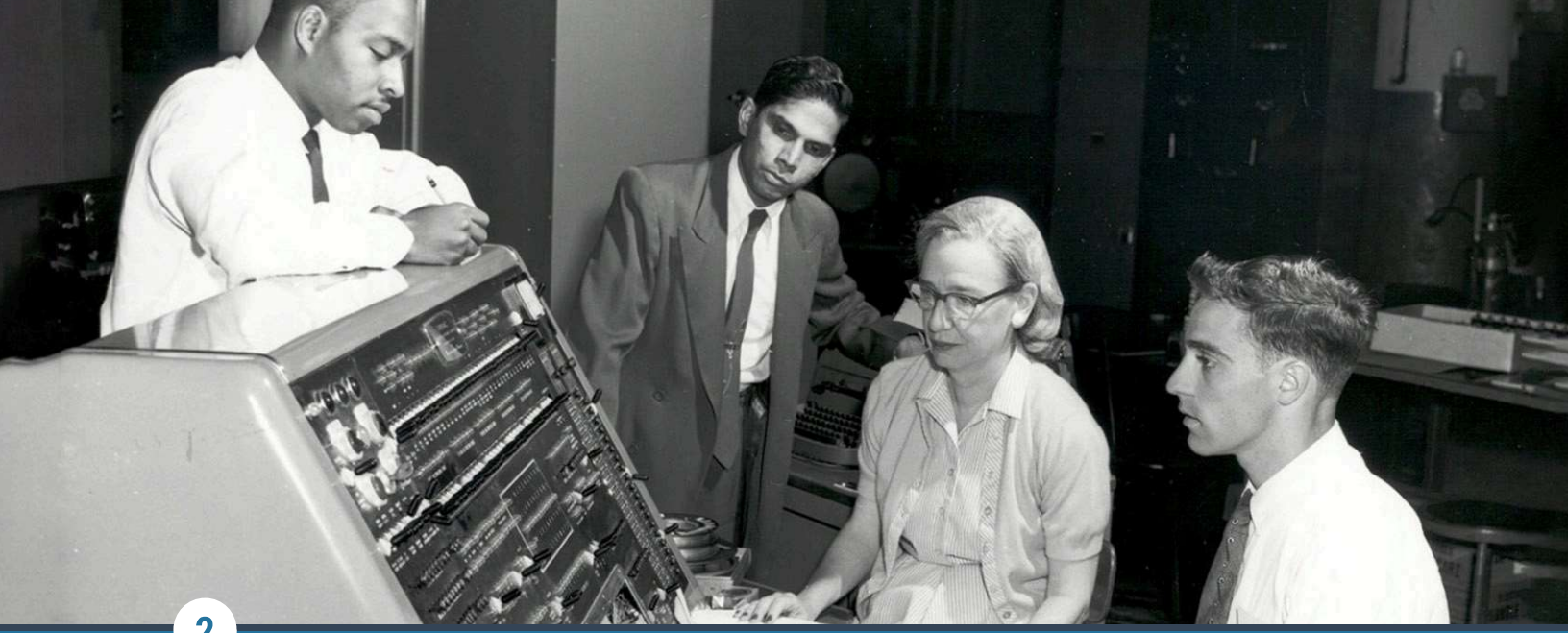

2 Expressions

Figure 2.1 credit: modification of work "Grace Hopper at Univac I console", courtesy of the Computer History Museum

Chapter Outline

Introduction

A computer program is a sequence of statements that run one after the other. In Python, many statements consist of one or more expressions. An **expression** represents a single value to be computed. Ex: The expression `3*x - 5` evaluates to `7` when x is `4`. Learning to recognize expressions opens the door for programming all kinds of interesting calculations.

Expressions are often a combination of literals, variables, and operators. In the previous example, `3` and `5` are literals, x is a variable, and * and - are operators. Expressions can be arbitrarily long, consisting of many calculations. Expressions can also be as short as one value. Ex: In the assignment statement `x = 5`, the literal `5` is an expression.

The Statements chapter introduced simple expressions like `1 * 2` and `"Hi " + "there"`. This chapter explores other kinds of expressions for working with numbers and strings. The first section shows a great way to experiment with expressions using a Python shell. Later sections present more details about integers and floating-point numbers, explain how to import and use the `math` module, and show how to make long lines of code easier to read.

2.1 The Python shell

Learning objectives

By the end of this section you should be able to

- Use a Python shell to run statements and expressions interactively.
- Explain the function of the up and down arrow keyboard shortcuts.

The interpreter

Python is a high-level language, meaning that the source code is intended for humans to understand. Computers, on the other hand, understand only low-level machine code made up of 1's and 0's. Programs written in high-level languages must be translated into machine code to run. This translation process can happen all at once, or a little at a time, depending on the language.

Python is an interpreted language: the source code is translated one line at a time while the program is running. The Python **interpreter** translates source code into machine code and runs the resulting program. If and when an error occurs, the interpreter stops translating the source code and displays an error message.

Most development environments include a Python shell for experimenting with code interactively. A **shell**, also called a console or terminal, is a program that allows direct interaction with an interpreter. The interpreter usually runs an entire program all at once. But the interpreter can run one line of code at a time within a Python shell.

CHECKPOINT

Running a Python shell

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-1-the-python-shell)

CONCEPTS IN PRACTICE

Using a Python shell

1. Python is a _____ language.
   a. high-level
   b. low-level

2. Which of the following is the most basic line of code the interpreter can run?
   a. `print(1 + 1)`
   b. `1 + 1`
   c. `1`

3. What result does the shell display after running the line `name = input()`?
   a. the name that was input
   b. nothing (except for >>>)

The arrow keys

A Python shell is convenient for exploring and troubleshooting code. The user can try something, look at the

results, and then try something else. When an error occurs, an error message is displayed, but the program keeps running. That way, the user can edit the previous line and correct the error interactively.

The acronym REPL (pronounced "rep ul") is often used when referring to a shell. **REPL** stands for "read-eval-print loop," which describes the repetitive nature of a shell:

1. **R**ead/input some code
2. **E**valuate/run the code
3. **P**rint any results
4. **L**oop back to step 1

Most shells maintain a history of every line of code the user types. Pressing the up or down arrow key on the keyboard displays the history. The **up arrow** displays the previous line; the **down arrow** displays the next line. That way, the user can repeat a line without having to type the line again.

CHECKPOINT

Correcting a typo

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-1-the-python-shell)

CONCEPTS IN PRACTICE

Using the arrow keys

4. Which arrow keys were needed to edit the typo?
   a. only the up arrow key
   b. the up and down arrows
   c. the up and left arrows

5. What keys would the user press to go back two lines?
   a. press the up arrow twice
   b. press the down arrow twice
   c. press the left arrow twice

TRY IT

Exploring the shell

Running code interactively is a great way to learn how Python works. Open a Python shell on your computer, or use the one at python.org/shell (https://openstax.org/r/100pythonshell). Then enter any Python code, one line at a time, to see the result. Here are a few expressions to try:

- `x = 5`
- `3*x - 5`
- `3 * (x-5)`
- `x`
- `type(1)`
- `type('1')`

- `str(1)`
- `int('1')`
- `abs(-5)`
- `abs(5)`
- `len("Yo")`
- `len("HoHo")`
- `round(9.49)`
- `round(9.50)`

Note: These functions (`type`, `str`, `int`, `len`, and `round`) will be explored in more detail later in the chapter. You can read more about the built-in functions (https://openstax.org/r/100builtin) in the Python documentation.

TRY IT

Correcting mistakes

Open a Python shell on your computer, or use the one at python.org/shell (https://openstax.org/r/100pythonshell). Run the following two statements in the shell:

- `x = 123`
- `y = 456`

Making mistakes is common while typing in a shell. The following lines include typos and other errors. For each line: (1) run the line in a shell to see the result, (2) press the up arrow to repeat the line, and (3) edit the line to get the correct result.

- `print("Easy as", X)`
- `print("y divided by 2 is", y / 0)`
- `name = intput("What is your name? ")`
- `print(name, "is", int(name), "letters long.")`
- `print("That's all folks!)`

The expected output, after correcting typos, should look like:

- `Easy as 123`
- `y divided by 2 is 228.0`
- `(no error/output)`
- `Stacie is 6 letters long.`
- `That's all folks!`

2.2 Type conversion

Learning objectives

By the end of this section you should be able to

- Explain how the interpreter uses implicit type conversion.
- Use explicit type conversion with `int()`, `float()`, and `str()`.

Implicit type conversion

Common operations update a variable such that the variable's data type needs to be changed. Ex: A GPS first assigns `distance` with `250`, an integer. After a wrong turn, the GPS assigns `distance` with `252.5`, a float. The Python interpreter uses **implicit type conversion** to automatically convert one data type to another. Once `distance` is assigned with `252.5`, the interpreter will convert `distance` from an integer to a float without the programmer needing to specify the conversion.

CHECKPOINT

Example: Book ratings

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-2-type-conversion)

CONCEPTS IN PRACTICE

Implicit type conversion in practice

Consider the example above.

1. What is `book_rating`'s data type on line 7?
   a. float
   b. integer

2. What would `book_rating`'s data type be if `update = 1.0` instead of `0.5`?
   a. float
   b. integer

3. What is the data type of x after the following code executes?

```
x = 42.0
x = x * 1
```

   a. float
   b. integer

Explicit type conversion

A programmer often needs to change data types to perform an operation. Ex: A program should read in two values using `input()` and sum the values. Remember `input()` reads in values as strings. A programmer can use **explicit type conversion** to convert one data type to another.

- **int()** converts a data type to an integer. Any fractional part is removed. Ex: `int(5.9)` produces `5`.
- **float()** converts a data type to a float. Ex: `float(2)` produces `2.0`.
- **str()** converts a data type to a string. Ex: `str(3.14)` produces `"3.14"`.

CHECKPOINT

Example: Ordering pizza

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-2-type-conversion)

CONCEPTS IN PRACTICE

Example: Ordering pizza

Consider the example above.

4. Which function converts the input number of slices to a data type that can be used in the calculation?
   a. `float()`
   b. `input()`
   c. `int()`

5. How could line 3 be changed to improve the program overall?
   a. Use `float()` instead of `int()`.
   b. Add `1` to the result of `int()`.
   c. Add `str()` around `int()`.

CONCEPTS IN PRACTICE

Using int(), float(), and str()

Given `x = 4.5` and `y = int(x)`, what is the value of each expression?

6. `y`
   a. `4`
   b. `5`

7. `str(x)`
   a. `4.5`
   b. `"4.5"`

8. `float(y)`
   a. `4.0`
   b. `4.5`

TRY IT

Grade average

The following program computes the average of three predefined exam grades and prints the average twice. Improve the program to read the three grades from input and print the average first as a float, and

then as an integer, using explicit type conversion. Ignore any differences that occur due to rounding.

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-2-type-conversion)

TRY IT

Cups of water

The following program should read in the ounces of water the user drank today and compute the number of cups drank and the number of cups left to drink based on a daily goal. Assume a cup contains 8 ounces. Fix the code to calculate `cups_drank` and `cups_left` and match the following:

- `ounces` is an integer representing the ounces the user drank.
- `cups_drank` is a float representing the number of cups of water drank.
- `cups_left` is an integer representing the number of cups of water left to drink (rounded down) out of the daily goal of 8 cups.

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-2-type-conversion)

TRY IT

Product as float

The following program reads two integers in as strings. Calculate the product of the two integers, and print the result as a float.

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-2-type-conversion)

2.3 Mixed data types

Learning objectives

By the end of this section you should be able to

- Identify the data types produced by operations with integers, floats, and strings.
- Use operators and type conversions to combine integers, floats, and strings.

Combining integers and floats

Programmers often need to combine numbers of different data types. Ex: A program computes the total for an online shopping order:

```
quantity = int(input())
price = float(input())
total = quantity * price
print(total)
```

`quantity` is an integer, and `price` is a float. So what is the data type of `total`? For input `3` and `5.0`, `total` is

a float, and the program prints `15.0`.

Combining an integer and a float produces a float. A float is by default printed with at least one figure after the decimal point and has as many figures as needed to represent the value. Note: Division using the `/` operator always produces a float.

CHECKPOINT

Operations combining integers and floats

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-3-mixed-data-types)

CONCEPTS IN PRACTICE

Operations combining integers and floats

1. `8 * 0.25`
   a. `2`
   b. `2.0`

2. `2 * 9`
   a. `18`
   b. `18.0`

3. `20 / 2`
   a. `10`
   b. `10.0`

4. `7 / 2`
   a. `3.0`
   b. `3.5`

5. `12.0 / 4`
   a. `3`
   b. `3.0`

6. `8 - 1.0`
   a. `7.0`
   b. `7`

7. `5 - 0.25`
   a. `4.5`
   b. `4.75`

Combining numeric types and strings

Easy type conversion in Python can lead a programmer to assume that any data type can be combined with another. Ex: Noor's program reads in a number from input and uses the number in a calculation. This results in

an error in the program because the `input()` function by default stores the number as a string. Strings and numeric data types are incompatible for addition, subtraction, and division. One of the operands needs to be explicitly converted depending on the goal of arithmetic or string concatenation.

The * operator also serves as the repetition operator, which accepts a string operand and an integer operand and repeats the string. Ex: `"banjo" * 3` produces `"banjobanjobanjo"`.

CHECKPOINT

Adding a string and an integer

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-3-mixed-data-types)

CONCEPTS IN PRACTICE

Operations combining numeric types and strings

8. `int('34') + 56`
   a. `'3456'`
   b. `90`
   c. `'90'`

9. `str(12) + ' red roses'`
   a. `'12 red roses'`
   b. `'12'`
   c. `Error`

10. `'50' * 3`
    a. `'150'`
    b. `150`
    c. `'505050'`

11. `str(5.2) + 7`
    a. `12.2`
    b. `'12.2'`
    c. `Error`

12. `80.0 + int('100')`
    a. `180`
    b. `180.0`
    c. `'180'`

13. `str(3.14) + '159'`
    a. `162.14`
    b. `'3.14159'`
    c. `Error`

14. `2.0 * 'this'`

a. `'this'`
b. `'thisthis'`
c. `Error`

TRY IT

After the point

Write a program that reads in a string of digits that represents the digits after the decimal point of a number, num. Concatenate the input string together with `'.'` and `num`, and print the result. Ex: If input is `345`, the program will print `2.345`.

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-3-mixed-data-types)

TRY IT

Print n times

Write a program that reads in two strings, `str1` and `str2`, and an integer, `count`. Concatenate the two strings with a space in between and a newline (`"\n"`) at the end. Print the resulting string `count` times.

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-3-mixed-data-types)

2.4 Floating-point errors

Learning objectives

By the end of this section you should be able to

- Explain numerical inaccuracies related to floating-point representation.
- Use the `round()` function to mitigate floating-point errors in output.

Floating-point errors

Computers store information using 0's and 1's. All information must be converted to a string of 0's and 1's. Ex: 5 is converted to 101. Since only two values, 0 or 1, are allowed the format is called binary.

Floating-point values are stored as binary by Python. The conversion of a floating point number to the underlying binary results in specific types of floating-point errors.

A **round-off error** occurs when floating-point values are stored erroneously as an approximation. The difference between an approximation of a value used in computation and the correct (true) value is called a round-off error.

Ex: Storing the float $(0.1)_{10}$ results in binary values that actually produce $(0.1000000000000000055511151231257827021181583404541015625)_{10}$ when converted back, which is not

exactly equal to $(0.1)_{10}$.

`# Print floats with 30 decimal places`
`print(f'{0.1:.30f}') # prints 0.1`
`print(f'{0.2:.30f}') # prints 0.2`
`print(f'{0.4:.30f}') # prints 0.4`
	`0.100000000000000005551115123126`
	`0.200000000000000011102230246252`
	`0.400000000000000022204460492503`

Table 2.1 Round-off error. (The example above shows a formatted string or f-string, which are introduced in the Objects chapter.)

An **overflow error** occurs when a value is too large to be stored. The maximum and minimum floating-point values that can be represented are 1.8×10^{308} and -1.8×10^{308}, respectively. Attempting to store a floating-point value outside the range $\left(-1.8 \times 10^{308}, 1.8 \times 10^{308}\right)$ leads to an overflow error.

Below, 3.0^{256} and 3.0^{512} can be represented, but 3.0^{1024} is too large and causes an overflow error.

`print('3.0 to the power of 256 =', 3.0**256)`
`print('3.0 to the power of 512 = ', 3.0**512)`
`print('3.0 to the power of 1024 = ', 3.0**1024)`
	`3.0 to the power of 256 = 1.3900845237714473e+122`
	`3.0 to the power of 512 = 1.9323349832288915e+244`
	`3.0 to the power of 1024 =`
	`Traceback (most recent call last):`
	`  File "<stdin>", line 3, in <module>`
	`  print('3.0 to the power of 1024 = ', 3.0**1024)`
	`OverflowError: (34, 'Numerical result out of range')`

Table 2.2 Overflow error.

CONCEPTS IN PRACTICE

Floating-point errors

For each situation, which error occurs?

1. The statement `result = 2.0 * (10.0 ** 500)` assigns the variable result with too large of a value.
   a. round-off
   b. overflow

2. `0.12345678901234567890123456789 * 0.1` produces `0.012345678901234568430878013601`.
   a. round-off
   b. overflow

Floating point round() function

Python's **round()** function is used to round a floating-point number to a given number of decimal places. The function requires two arguments. The first argument is the number to be rounded. The second argument decides the number of decimal places to which the number is rounded. If the second argument is not provided, the number will be rounded to the closest integer. The `round()` function can be used to mitigate floating-point errors.

Ex:

- `round(2.451, 2) = 2.45`
- `round(2.451) = 2`

CHECKPOINT

Examples of round() function

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-4-floating-point-errors)

CONCEPTS IN PRACTICE

Examples of round() function

3. What is the output of `round(6.6)`?
   a. `6`
   b. `6.6`
   c. `7`

4. What is the output of `round(3.5, 2)`?
   a. `3`
   b. `3.5`
   c. `3.50`

5. What is the output of `round(12)`?
   a. `12.0`
   b. `12`
   c. `12.00`

6. What is the output of `round(0.1, 1)`?
   a. `0.1`
   b. `0.10`
   c. `0.1000000000000000055511151231257827021181583404541015625`

TRY IT

Inaccurate tips

The following code calculates the tip amount, given a bill amount and the tip ratio. Experiment with the

following bill amounts and tip ratios and see if any inaccuracies may result in calculating the tip amount.

- bill amount: 22.70 and 33.33
- tip ratio: 0.15, 0.18, and 0.20

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-4-floating-point-errors)

TRY IT

Area of a triangle

Complete the following steps to calculate a triangle's area, and print the result of each step. The area of a triangle is $\frac{bh}{2}$, where b is the base and h is the height.

1. Calculate the area of a triangle with base = 7 and height = 3.5.
2. Round the triangle's area to one decimal place.
3. Round the triangle's area to the nearest integer value.

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-4-floating-point-errors)

2.5 Dividing integers

Learning objectives

By the end of this section you should be able to

- Evaluate expressions that involve floor division and modulo.
- Use the modulo operator to convert between units of measure.

Division and modulo

Python provides two ways to divide numbers:

- **True division** (`/`) converts numbers to floats before dividing. Ex: `7 / 4` becomes `7.0 / 4.0`, resulting in `1.75`.
- **Floor division** (`//`) computes the quotient, or the number of times divided. Ex: `7 // 4` is `1` because 4 goes into 7 one time, remainder 3. The **modulo operator** (`%`) computes the remainder. Ex: `7 % 4` is `3`.

Note: The `%` operator is traditionally pronounced "mod" (short for "modulo"). Ex: When reading `7 % 4` out loud, a programmer would say "seven mod four."

CHECKPOINT

Quotient and remainder

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-5-dividing-integers)

CONCEPTS IN PRACTICE

Division and modulo

What is the value of each expression?

1. `13 / 5`
   a. `2`
   b. `2.6`
   c. `3`

2. `13 % 5`
   a. `2`
   b. `2.6`
   c. `3`

3. `1 // 4`
   a. `0`
   b. `0.25`
   c. `1`

4. `2 % 0`
   a. `0`
   b. `2`
   c. `Error`

Unit conversions

Division is useful for converting one unit of measure to another. To convert centimeters to meters, a variable is divided by 100. Ex: 300 centimeters divided by 100 is 3 meters.

Amounts often do not divide evenly as integers. 193 centimeters is 1.93 meters, or 1 meter and 93 centimeters. A program can use floor division and modulo to separate the units:

- The quotient, 1 meter, is `193 // 100`.
- The remainder, 93 centimeters, is `193 % 100`.

Programs often use floor division and modulo together. If one line of code floor divides by `m`, the next line will likely modulo by `m`. The unit `m` by which an amount is divided is called the **modulus**. Ex: When converting centimeters to meters, the modulus is 100.

CHECKPOINT

Money and time

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-5-dividing-integers)

CONCEPTS IN PRACTICE

Unit conversions

5. What is the modulus for converting minutes to hours?
   a. 40
   b. 60
   c. 280

6. A program has the line `pounds = ounces // 16`. What is likely the next line of code?
   a. `ounces = ounces % 16`
   b. `pounds = ounces % 16`
   c. `ounces = ounces - pounds * 16`

TRY IT

Arrival time

Having a mobile device can be a lifesaver on long road trips. Programs like Google Maps find the shortest route and estimate the time of arrival. The time of arrival is based on the current time plus how long the trip will take.

Write a program that (1) inputs the current time and estimated length of a trip, (2) calculates the time of arrival, and (3) outputs the results in hours and minutes. Your program should use the following prompts (user input in bold):

13

```
    Current minute (0-59)? 25
    Trip time (in minutes)? 340

    Current hour (0-23)? 13
    Current minute (0-59)? 25
    Trip time (in minutes)? 340
```

In this example, the current time is 13:25 (1:25pm). The trip time is 340 minutes (5 hours and 40 minutes). 340 minutes after 13:25 is 19:05 (7:05pm). Your program should output the result in this format:

```
    Arrival hour is 19
    Arrival minute is 5
```

The arrival hour must be between 0 and 23. Ex: Adding 120 minutes to 23:00 should be 1:00, not 25:00. The arrival minute must be between 0 and 59. Ex: Adding 20 minutes to 8:55 should be 9:15, not 8:75.

Hint: Multiply the current hour by 60 to convert hours to minutes. Then, calculate the arrival time, in total minutes, as an integer.

Your code must not use Python keywords from later chapters, such as `if` or `while`. The solution requires

only addition, multiplication, division, and modulo.

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-5-dividing-integers)

TRY IT

Change machine

Self-checkout aisles are becoming increasingly popular at grocery stores. Customers scan their own items, and a computer determines the total purchase amount. Customers who pay in cash insert dollar bills, and a machine automatically dispenses change in coins.

That's where this program comes into the story. Your task is to calculate how many of each coin to dispense. Your program should use the following prompts (user input in bold):

```
18.76
    Cash payment? 20

    Total amount? 18.76
    Cash payment? 20
```

You may assume that the cash paid will always be a whole number (representing dollar bills) that is greater than or equal to the total amount. The program should calculate and output the amount of change due and how many dollars, quarters, dimes, nickels, and pennies should be dispensed:

```
    Change Due $1.24

     Dollars: 1
    Quarters: 0
       Dimes: 2
     Nickels: 0
     Pennies: 4
```

Hint: Calculate the total change, in cents, as an integer. Use the `round()` function to avoid floating-point errors.

Your code must not use Python keywords from later chapters, such as `if` or `while`. The solution requires only subtraction, multiplication, division, and modulo.

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-5-dividing-integers)

2.6 The math module

Learning objectives

By the end of this section you should be able to

- Distinguish between built-in functions and math functions.
- Use functions and constants defined in the math module.

Importing modules

Python comes with an extensive standard library (https://openstax.org/r/100pythlibrary) of modules. A **module** is previously written code that can be imported in a program. The **import statement** defines a variable for accessing code in a module. Import statements often appear at the beginning of a program.

The standard library also defines built-in functions such as `print()`, `input()`, and `float()`. A **built-in function** is always available and does not need to be imported. The complete list of built-in functions (https://openstax.org/r/100builtin) is available in Python's official documentation.

A commonly used module in the standard library is the math module (https://openstax.org/r/100mathmodule). This module defines functions such as `sqrt()` (square root). To call `sqrt()`, a program must `import math` and use the resulting `math` variable followed by a dot. Ex: `math.sqrt(25)` evaluates to `5.0`.

The following program imports and uses the math module, and uses built-in functions for input and output.

EXAMPLE 2.1

Calculating the distance between two points

```
import math

x1 = float(input("Enter x1: "))
y1 = float(input("Enter y1: "))
x2 = float(input("Enter x2: "))
y2 = float(input("Enter y2: "))

distance = math.sqrt((x2-x1)**2 + (y2-y1)**2)
print("The distance is", distance)
```

CHECKPOINT

Importing math in a Python shell

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-6-the-math-module)

CONCEPTS IN PRACTICE

Built-in functions and math module

1. In the above example, when evaluating math, why did the interpreter raise a `NameError`?
   a. The math module was not available.

b. The variable pie was spelled incorrectly.
c. The math module was not imported.

2. Which of these functions is builtin and does not need to be imported?
   a. `log()`
   b. `round()`
   c. `sqrt()`

3. Which expression results in an error?
   a. `math.abs(1)`
   b. `math.log(1)`
   c. `math.sqrt(1)`

Mathematical functions

Commonly used math functions and constants are shown below. The complete math module listing (https://openstax.org/r/100mathmodule) is available in Python's official documentation.

Constant	Value	Description
`math.e`	$e = 2.71828\ldots$	Euler's number: the base of the natural logarithm.
`math.pi`	$\pi = 3.14159\ldots$	The ratio of the circumference to the diameter of a circle.
`math.tau`	$\tau = 6.28318\ldots$	The ratio of the circumference to the radius of a circle. Tau is equal to 2π.

Table 2.3 Example constants in the math module.

Function	Description	Examples
Number-theoretic		
`math.ceil(x)`	The ceiling of x: the smallest integer greater than or equal to x.	`math.ceil(7.4)` → `8` `math.ceil(-7.4)` → `-7`
`math.floor(x)`	The floor of x: the largest integer less than or equal to x.	`math.floor(7.4)` → `7` `math.floor(-7.4)` → `-8`
Power and logarithmic		
`math.log(x)`	The natural logarithm of x (to base e).	`math.log(math.e)` → `1.0` `math.log(0)` → `ValueError: math domain error`

Table 2.4 Example functions in the math module.

Function	Description	Examples
`math.log(x, base)`	The logarithm of x to the given base.	`math.log(8, 2)` → `3.0` `math.log(10000, 10)` → `4.0`
`math.pow(x, y)`	x raised to the power y. Unlike the ** operator, `math.pow()` converts x and y to type float.	`math.pow(3, 0)` → `1.0` `math.pow(3, 3)` → `27.0`

`math.sqrt(x)`	The square root of x.	`math.sqrt(9)` → `3.0`
		`math.sqrt(-9)` →
		`ValueError: math domain error`

Trigonometric		
`math.cos(x)`	The cosine of x radians.	`math.cos(0)` → `1.0` `math.cos(math.pi)` → `-1.0`
`math.sin(x)`	The sine of x radians.	`math.sin(0)` → `0.0` `math.sin(math.pi/2)` → `1.0`

`math.tan(x)`	The tangent of x radians.	`math.tan(0)` → `0.0`
		`math.tan(math.pi/4)` →
		`0.999`
		(Round-off error; the result should be 1.0.)

Table 2.4 Example functions in the math module.

CONCEPTS IN PRACTICE

Using math functions and constants

4. What is the value of `math.tau/2`?
   a. approximately `2.718`
   b. approximately `3.142`
   c. approximately `6.283`

5. What is the value of `math.sqrt(100)`?
   a. the float `10.0`
   b. the integer `10`
   c. `ValueError: math domain error`

6. What is πr^2 in Python syntax?
   a. `pi * r**2`

b. `math.pi * r**2`
c. `math.pi * r*2`

7. Which expression returns the integer 27?
a. `3 ** 3`
b. `3.0 ** 3`
c. `math.pow(3, 3)`

TRY IT

Quadratic formula

In algebra, a quadratic equation is written as $ax^2 + bx + c = 0$. The coefficients *a*, *b*, and *c* are known values. The variable *x* represents an unknown value. Ex: $2x^2 + 3x - 5 = 0$ has the coefficients $a = 2$, $b = 3$, and $c = -5$. The quadratic formula provides a quick and easy way to solve a quadratic equation for *x*:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The plus-minus symbol indicates the equation has two solutions. However, Python does not have a plus-minus operator. To use this formula in Python, the formula must be separated:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$
$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

Write the code for the quadratic formula in the program below. Test your program using the following values for *a*, *b*, and *c*:

Provided input			Expected output	
a	b	c	x1	x2
1	0	-4	2.0	-2.0
1	2	-3	1.0	-3.0
2	1	-1	0.5	-1.0

Table 2.5

Provided input			Expected output
0	1	1	division by zero
1	0	1	math domain error

Table 2.5

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-6-the-math-module)

TRY IT

Cylinder formulas

In geometry, the surface area and volume of a right circular cylinder can be computed as follows:

$A = 2\pi rh + 2\pi r^2$

$V = \pi r^2 h$

Write the code for these two formulas in the program below. Hint: Your solution should use both `math.pi` and `math.tau`. Test your program using the following values for r and h:

Provided input		Expected output	
r	h	area	volume
0	0	0.0	0.0
1	1	12.57	3.14
1	2	18.85	6.28
2.5	4.8	114.67	94.25
3.1	7.0	196.73	211.33

Table 2.6

If you get an error, try to look up what that error means.

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-6-the-math-module)

2.7 Formatting code

Learning objectives

By the end of this section you should be able to

- Identify good spacing for expressions and statements.
- Write multi-line statements using implicit line joining.

Recommended spacing

Most spaces in Python code are ignored when running programs; however, spaces at the start of a line are very important. The following two programs are equivalent:

- Good spacing:

```
name = input("Enter someone's name: ")
place = input("Enter a famous place: ")
print(name, "should visit", place + "!")
```

- Poor spacing:

```
name=input ("Enter someone's name: " )
place =input("Enter a famous place: ")
print( name,"should visit" , place+ "!")
```

One might argue that missing or extra spaces do not matter. After all, the two programs above run exactly the same way. However, the "poor spacing" version is more difficult to read. Code like `name=input` and `place+` might lead to confusion.

Good programmers write code that is as easy to read as possible. That way, other programmers are more likely to understand the code. To encourage consistency, the Python community has a set of guidelines about where to put spaces and blank lines, what to name variables, how to break up long lines, and other important topics.

PYTHON STYLE GUIDE

PEP 8 (https://openstax.org/r/100PEP8) is the official style guide for Python. **PEP** stands for Python Enhancement Proposal. Members of the Python community write PEPs to document best practices and propose new features. The table below is based on guidelines from PEP 8 under the heading Whitespace in Expressions and Statements (https://openstax.org/r/100whitespace).

Guideline	Example	Common Mistakes
Parentheses: no space before or after.	`print("Go team!")`	`print ("Go team!")` `print( "Go team!" )`
Commas: no space before, one space after.	`print("Hello", name)`	`print("Hello" , name)` `print("Hello",name)`

Assignment: one space before and after the =.	`name = input("Your name? ")`	`name=input("Your name? ")`
		`name= input("Your name? ")`
		`name =input("Your name? ")`

Concatenation: one space before and after the +.	`print("Hi", name + "!")`	`print("Hi", name+"!")`
		`print("Hi", name+ "!")`
		`print("Hi", name +"!")`

Arithmetic: use space to show lower precedence.	`x**2 + 5*x - 8`	`x ** 2 + 5 * x - 8`
		`x ** 2+5 * x-8`
		`x**2+5*x-8`

Table 2.7 Guidelines for spaces.

CONCEPTS IN PRACTICE

Recommended spacing

1. Which statement is formatted properly?
   a. `name = input("What is your name? ")`
   b. `name = input ("What is your name? ")`
   c. `name = input( "What is your name? " )`

2. Which statement is formatted properly?
   a. `name=name+"!"`
   b. `name = name+"!"`
   c. `name = name + "!"`

3. Which statement is formatted properly?
   a. `print("Hello",name)`
   b. `print("Hello", name)`
   c. `print("Hello " , name)`

4. Which expression is formatted properly?

a. `b**2 - 4*a*c`
b. `b ** 2 - 4 * a * c`
c. `b**2 - 4*a * c`

Automatic concatenation

Long strings make Python programs difficult to read. Ex: This program prints the first sentence of the US Declaration of Independence (https://openstax.org/r/100declaration):

print("The unanimous Declaration of the thirteen united States of America, When in the Course of human events, it becomes necessary for one people to dissolve the political bands which have connected them with another, and to assume among the powers of the earth, the separate and equal station to which the Laws of Nature and of Nature's God entitle them, a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the separation.")

PEP 8 recommends that each line of code be less than 80 characters long. That way, programmers won't need to scroll horizontally to read the code. The above program can be rewritten by breaking up the original string:

```
print("The unanimous Declaration of the thirteen united States of "
      "America, When in the Course of human events, it becomes "
      "necessary for one people to dissolve the political bands "
      "which have connected them with another, and to assume among "
      "the powers of the earth, the separate and equal station to "
      "which the Laws of Nature and of Nature's God entitle them, a "
      "decent respect to the opinions of mankind requires that they "
      "should declare the causes which impel them to the separation.")
```

For convenience, Python automatically concatenates multiple strings. The + operator is not required in this situation.

CHECKPOINT

String concatenation

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-7-formatting-code)

CONCEPTS IN PRACTICE

String literal concatenation

5. Which line prints the word `"grandmother"`?
   a. `print(grandmother)`
   b. `print("grand" "mother")`
   c. `print("grand", "mother")`

6. What string is equivalent to `"Today is" "a holiday"`?

a. `'Today isa holiday'`
b. `'Today is a holiday'`
c. `'Today is" "a holiday'`

7. If name is `"Ada"`, what does `print("Hello," name)` output?
   a. `Hello,Ada`
   b. `Hello, Ada`
   c. `SyntaxError`

Multi-line statements

Most statements in a Python program need only one line of code. But occasionally longer statements need to span multiple lines. Python provides two ways to write multi-line statements:

- Explicit line joining, using \ characters:

```
decl = "The unanimous Declaration of the thirteen united States of " \
       "America, When in the Course of human events, it becomes " \
       "necessary for one people to dissolve the political bands..."
```

- Implicit line joining, using parentheses:

```
decl = ("The unanimous Declaration of the thirteen united States of "
       "America, When in the Course of human events, it becomes "
       "necessary for one people to dissolve the political bands...")
```

Implicit line joining is more common, since many statements and expressions use parentheses anyway. PEP 8 recommends avoiding the use of explicit line joining whenever possible.

CONCEPTS IN PRACTICE

Multi-line statements

8. Which character is used for explicit line joining?
   a. /
   b. \
   c. |

9. What is the best way to print a very long string?
   a. Break up the string into multiple smaller strings.
   ```
   print("..." # first part of string
   "..." # next part of string
   "...")
   ```
   b. Print the string using multiple print statements.
   ```
   print("...") # first part of string
   print("...") # next part of string
   print("...")
   ```

c. Assign the string to a variable and print the variable.

```
text = "..." # the entire string
print(text)
```

10. Which example consists of two statements?

a.
```
print("Happy "
"New Year")
```

b.
```
saying = ("Happy "
"New Year")
```

c.
```
saying= "Happy "
"New Year"
```

TRY IT

Spaced out

The following code works correctly but is formatted poorly. In particular, the code does not include spaces recommended by PEP 8. Furthermore, two of the lines are about 90 characters long. Reformat the code to follow the guidelines in this section. Be careful not to change the behavior of the code itself.

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-7-formatting-code)

TRY IT

Five quotes

Write a program that prints the following five quotes (source: BrainyQuote (https://openstax.org/r/100brainyquote)) from Guido van Rossum, the creator of Python. Your program should have exactly five print statements, one for each quote:

```
    1. "If you're talking about Java in particular, Python is about the best fit
you can get amongst all the other languages. Yet the funny thing is, from a
language point of view, JavaScript has a lot in common with Python, but it is sort
of a restricted subset."
    2. "The second stream of material that is going to come out of this project is
a programming environment and a set of programming tools where we really want to
focus again on the needs of the newbie. This environment is going to have to be
extremely user-friendly."
    3. "I have this hope that there is a better way. Higher-level tools that
actually let you see the structure of the software more clearly will be of
tremendous value."
    4. "Now, it's my belief that Python is a lot easier than to teach to students
programming and teach them C or C++ or Java at the same time because all the
details of the languages are so much harder. Other scripting languages really don't
work very well there either."
```

```
    5. "I would guess that the decision to create a small special purpose language
or use an existing general purpose language is one of the toughest decisions that
anyone facing the need for a new language must make."
```

Notice that all of these lines are longer than 80 characters, and some contain single quote marks. Format the code using multi-line statements and escape sequences as necessary.

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-7-formatting-code)

2.8 Python careers

Learning objectives

By the end of this section you should be able to

- Summarize how Python is used in fields other than CS and IT.
- Describe two different kinds of applications made with Python.

Fields and applications

Learning Python opens the door to many programming-related careers. Example job titles include software engineer, data scientist, web developer, and systems analyst. These jobs often require a degree in computer science (CS), information technology (IT), or a related field. However, programming is not limited to these fields and careers.

Many professionals use Python to support the work they do. Python programs can automate tasks and solve problems quickly. Table 2.8 shows a few examples of Python outside of computing fields. Knowing how to program is a useful skill that can enhance any career.

Python is a versatile language that supports many kinds of applications. Table 2.9 shows a few examples of programs that can be written in Python. Given Python's usefulness and popularity, Python is a great language to learn. A supportive community of professionals and enthusiasts is ready to help.

Field	Example use of Python
Business	An accountant writes a Python program to generate custom sales reports.
Education	A teacher writes a Python program to organize homework submissions.
Fine arts	An artist writes a Python program to operate an interactive art display.
Humanities	A linguist writes a Python program to analyze changes in slang word usage.
Science	A biologist writes a Python program to analyze DNA sequences for cancer.

Table 2.8 Python outside of CS and IT.

Application	Example use of Python
Artificial intelligence	An engineer develops models to support image and voice recognition. Ex: TensorFlow library (https://openstax.org/r/100tensorflow).
Data visualization	A statistician creates charts to make sense of large amounts of data. Ex: Matplotlib library (https://openstax.org/r/100matplotlib).
General purpose	Most programs in this book are general. Ex: Read inputs, perform calculations, print results.
Scientific computing	Python is very useful for conducting experiments and analyzing data. Ex: The SciPy project (https://openstax.org/r/100scipyproject).
Web development	Python can run interactive websites. Ex: Instagram is built with Django (https://openstax.org/r/100django), a Python framework.

Table 2.9 Applications built with Python.

CONCEPTS IN PRACTICE

Fields and applications

1. Which of the following fields use Python to support their work?
   a. geography
   b. health care
   c. political science
   d. all of the above

2. Which of the following Python libraries creates charts and plots?
   a. Matplotlib
   b. SciPy
   c. TensorFlow

3. Which of the following applications were built with Python?
   a. Facebook
   b. Instagram
   c. WhatsApp

EXPLORING FURTHER

For more examples of applications that can be built with Python, see "Top 12 Fascinating Python Applications in Real-World" (https://openstax.org/r/100top12apps) by Rohit Sharma. For more information about Python related careers, see "What Does a Python Developer Do?" (https://openstax.org/r/100careers) in BrainStation's career guide.

2.9 Chapter summary

Highlights from this chapter include:

- Expressions and statements can be run interactively using a shell.
- Input strings can be converted to other types. Ex: `int(input())`.
- Strings can be concatenated with other types. Ex: `"$" + str(cost)`.
- Floats are subject to round-off and overflow errors.
- Integers can be divided exactly using `//` and `%`.
- Modules like `math` provide many useful functions.
- Formatting long lines helps improve readability.

At this point, you should be able to write programs that ask for input of mixed types, perform mathematical calculations, and output results with better formatting. The programming practice below ties together most topics presented in the chapter.

Function	Description
`abs(x)`	Returns the absolute value of x.
`int(x)`	Converts x (a string or float) to an integer.
`float(x)`	Converts x (a string or integer) to a float.
`str(x)`	Converts x (a float or integer) to a string.
`round(x, ndigits)`	Rounds x to `ndigits` places after the decimal point. If `ndigits` is omitted, returns the nearest integer to x.
Operator	**Description**
`s * n` (Repetition)	Creates a string with n copies of s. Ex: `"Ha" * 3` is `"HaHaHa"`.
`x / y` (Real division)	Divides x by y and returns the entire result as a float. Ex: `7 / 4` is `1.75`.
`x // y` (Floor division)	Divides x by y and returns the quotient as an integer. Ex: `7 // 4` is `1`.
`x % y` (Modulo)	Divides x by y and returns the remainder as an integer. Ex: `7 % 4` is `3`.

Table 2.10 Chapter 2 reference.

TRY IT

Baking bread

The holidays are approaching, and you need to buy ingredients for baking many loaves of bread. According to a recipe by King Arthur Flour (https://openstax.org/r/100kingarthurflr), you will need the following ingredients for each loaf:

- 1 1/2 teaspoons instant yeast
- 1 1/2 teaspoons salt
- 1 1/2 teaspoons sugar
- 2 1/2 cups all-purpose flour
- 2 cups sourdough starter
- 1/2 cup lukewarm water

Write a program that inputs the following variables: `bread_weight` (float), `serving_size` (float), and `num_guests` (int). The output will look like the following:

Note: The measures the program comes up with are exact, but to bake, the baker would have to use some approximation. Ex: 9.765625 cups all-purpose flour really means 9 and 3/4 cups.

```
For 25 people, you will need 3.90625 loaves of bread:

  5.859375 teaspoons instant yeast
  5.859375 teaspoons salt
  5.859375 teaspoons sugar
  9.765625 cups all-purpose flour
  7.8125 cups sourdough starter
  1.953125 cups lukewarm water
```

In the above output, `bread_weight` is 16.0 ounces, `serving_size` is 2.5 ounces, and `num_guests` is 25 people. Use these three variables to calculate the number of loaves needed.

Make sure your output matches the above example exactly. Notice that each line of the ingredients begins with two spaces.

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-9-chapter-summary)

TRY IT

Tip calculator

Google has a variety of search tricks (https://openstax.org/r/100googletricks) that present users with instant results. If you search on Google for tip calculator (https://openstax.org/r/100tipcalculatr), an interactive tool is included at the top of the results. The goal of this exercise is to implement a similar tip calculator.

Begin by prompting the user to input the following values (user input in bold):

```
43.21
    Percentage to tip: 18
    Number of people: 2

    Enter bill amount: 43.21
    Percentage to tip: 18
    Number of people: 2
```

Then calculate the tip amount and total amount for the bill, based on the user input. Output the results using this format:

```
    Tip amount: $7.78
    Total amount: $50.99

    Tip per person: $3.89
    Total per person: $25.49
```

Your program should output all dollar amounts rounded to two decimal places. The output should be exactly six lines, as shown above. Notice the blank line before each section of the output. Notice also the space before but not after the dollar sign.

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/2-9-chapter-summary)