Chapter 9

How to work with numbers

Objectives

Applied

1. Code, test, and debug programs that work with numbers. That includes the use of:

   the math module
   f-strings for formatting numbers
   the locale module for formatting currency values for specific countries
   the decimal module

Knowledge

1. Describe how the use of floating-point numbers can lead to inaccurate results.
2. Describe the purposes of the math, locale, and decimal modules.
3. Describe two ways to eliminate the types of errors that can occur when using floating-point numbers.

Two numeric data types

Type	Bytes	Use
`int`	`4`	Integers from -2,147,483,648 to 2,147,483,647.
`float`	`8`	Floating-point numbers from -1.7E308 to +1.7E308 with up to 16 significant digits.

Examples of float values

```
21.5       # a positive float value
-124.82    # a negative float value
-3.7e-9    # floating-point notation for -0.0000000037
```

Variables that are assigned values using scientific notation

```
value1 = 2.382E+5      # 2.382 * 10^5 (or 238,200)
value2 = 3.25E-8       # 3.25 * 10^-8 (or .0000000325)
```

An example of a floating-point error

```
balance = 100.10
balance += 100.10
balance += 100.10
print("Balance =", balance)
```

The result

```
Balance = 300.29999999999995
```

Code that fixes the floating-point error

```
balance = round(balance, 2)
print("Balance =", balance)
```

The result

```
Balance = 300.3
```

Some common functions of the math module

```
pow(num, pow)
sqrt(num)
ceil(num)
floor(num)
```

A constant of the math module

```
pi
```

How to import the math module

```
import math as m
```

The pow() and sqrt() functions

```
result = m.pow(2, 3)       # 8.0 (the cube of 2)
result = m.sqrt(16)        # 4.0 (square root)
result = m.pow(125, 1/3)   # 4.999999999999999 (cube root)
```

The pi constant

```
radius = 12
circumference = m.pi * radius * 2     # 75.39822368615503
area = m.pi * m.pow(radius, 2)        # 452.3893421169302
area = m.pi * radius**2               # 452.3893421169302
```

The floor() and ceil() functions

```
result = m.floor(12.545)      # 12
result = m.ceil(12.545)       # 13
result = m.floor(-3.432)      # -4
result = m.ceil(-3.432)       # -3
```

The ceil() function with decimal places

```
m.ceil(2.0083)                      # 3
m.ceil(2.0083 * 10) / 10            # 2.1
m.ceil(2.0083 * 100) / 100          # 2.01
```

The floor() function with decimal places

```
m.floor(2.0083)                     # 2
m.floor(2.0083 * 10) / 10           # 2.0
m.floor(2.0083 * 1000) / 1000       # 2.008
```

The syntax of an f-string with a format specification

```
"f{value:format_specification}"
```

The syntax of the format specification

```
[field_width][comma][.decimal_places][type_code]
```

Common type codes

Code	Meaning
`d`	Integer
`f`	Floating-point number
`%`	Percent
`e`	Scientific notation

F-strings with format specifications

```
fp_number = 12345.6789
print(f"{fp_number:.2f}")          # 12345.68
print(f"{fp_number:,.2f}")         # 12,345.68
print(f"{fp_number:15,.2f}")       #       12,345.68
print(f"{fp_number:.2e}")          # 1.23e+04

fp_number = .12345
print(f"{fp_number:.0%}")          # 12%
print(f"{fp_number:.1%}")          # 12.3%

int_number = 12345
print(f"{int_number:d}")           # 12345
print(f"{int_number:,d}")          # 12,345
```

How to format a string literal

```
# enclose the literal in single quotes
print(f"{'Description':15}")
```

How to use a variable in a format specification

```
spec = 15
# enclose the variable in brackets
print(f"{'Description':{spec}}")
```

How to use field widths to align results

```
print(f"{'Description':15} {'Price':>10} {'Qty':>5}")
print(f"{'Hammer':15} {9.99:10.2f} {3:5d}")
```

The console

```
Description          Price  Qty
Hammer                9.99    3
```

Commonly used functions of the locale module

```
setlocale(category, locale)
currency(num[, grouping])
format(format, num[, grouping])
```

Codes for working with locales

Locale	Short code	Long code	Currency Format
English/United States	us	en_US	$12,345.15
English/United Kingdom	uk	en_UK	£12,345.15
German/Germany	de	de_DE	+12.345,15 €

How to import the locale module into the lc namespace

```
import locale as lc
```

How to set the locale to English/United States

```
lc.setlocale(lc.LC_ALL, "us")          # Windows
lc.setlocale(lc.LC_ALL, "en_US")       # macOS and Windows
```

How to set the locale on most Windows systems

```
lc.setlocale(lc.LC_ALL, "")
```

The currency() function

```
print(lc.currency(12345.15, grouping=True))  # $12,345.15 (US)
```

The format() function

```
print(lc.format("%d", 12345, grouping=True)) # 12,345    (US)
print(lc.format("%.2f", 12345.15, grouping=True))
                                             # 12,345.15 (US)
```

The user interface for the Invoice program with incorrect results

```
Enter order total: 100.05

Order total:             100.05
Discount amount:          10.01
Subtotal:                 90.05
Sales tax:                 4.50
Invoice total:            94.55

Continue? (y/n):
```

The code that yields incorrect results

```
order_total = float(input("Enter order total: "))
print()

# determine the discount percent
if order_total > 0 and order_total < 100:
    discount_percent = 0
elif order_total >= 100 and order_total < 250:
    discount_percent = .1
elif order_total >= 250:
    discount_percent = .2

# calculate the results
discount = order_total * discount_percent
subtotal = order_total - discount
sales_tax = subtotal * .05
invoice_total = subtotal + sales_tax

# display the results
print(f"Order total:       {order_total:10,.2f}")
print(f"Discount amount:   {discount:10,.2f}")
print(f"Subtotal:          {subtotal:10,.2f}")
print(f"Sales tax:         {sales_tax:10,.2f}")
print(f"Invoice total:     {invoice_total:10,.2f}")
print()
```

The code that fixes this problem

```
# calculate the results with rounding
discount = round(order_total * discount_percent, 2)
subtotal = order_total - discount
sales_tax = round(subtotal * .05, 2)
invoice_total = subtotal + sales_tax
```

The user interface for the Invoice program with correct results

```
Enter order total: 100.05

Order total:              100.05
Discount amount:           10.01
Subtotal:                  90.04
Sales tax:                  4.50
Invoice total:             94.54

Continue? (y/n):
```

How to create Decimal objects and use them in calculations

```
from decimal import Decimal

order_total = Decimal("100.05")
discount_percent = Decimal(".1")
discount = order_total * discount_percent  # 10.005

subtotal = order_total - discount          # 90.045
tax_percent = Decimal(".05")
sales_tax = subtotal * tax_percent         #  4.50225
invoice_total = subtotal + sales_tax       # 94.54725

test1 = subtotal * 2     # Legal. You can mix Decimal and int
test2 = subtotal * 3.5   # Error! You can't mix Decimal and float
```

The syntax of the quantize() method of a Decimal object

```
object.quantize(Decimal("positions_code")[,
    rounding_constant])
```

Three of the rounding constants

```
ROUND_HALF_UP
ROUND_HALF_DOWN
ROUND_HALF_EVEN
```

How to specify the number of decimal places

```
discount = Decimal("10.005")
discount = discount.quantize(Decimal("1.00"))    # 10.00
```

How to override the default rounding mode

```
from decimal import ROUND_HALF_UP

discount = Decimal("10.005")
discount = discount.quantize(Decimal("1.00"),
                             ROUND_HALF_UP)      # 10.01
```

The code for the Invoice program (part 1)

```
from decimal import Decimal
from decimal import ROUND_HALF_UP

choice = "y"
while choice == "y":

    # get the user entry
    order_total = Decimal(input("Enter order total:     "))
    order_total = order_total.quantize(Decimal("1.00"),
                                       ROUND_HALF_UP)
    print()

    # determine the discount percent
    if order_total > 0 and order_total < 100:
        discount_percent = Decimal("0")
    elif order_total >= 100 and order_total < 250:
        discount_percent = Decimal(".1")
    elif order_total >= 250:
        discount_percent = Decimal(".2")
```

The code for the Invoice program (part 2)

```
        # calculate the results
        discount = order_total * discount_percent
        discount = discount.quantize(Decimal("1.00"), ROUND_HALF_UP)
        subtotal = order_total - discount
        tax_percent = Decimal(".05")
        sales_tax = subtotal * tax_percent
        sales_tax = sales_tax.quantize(Decimal("1.00"),
                                       ROUND_HALF_UP)
        invoice_total = subtotal + sales_tax

        # display the results
        print(f"Order total:       {order_total:10,}")
        print(f"Discount amount:   {discount:10,}")
        print(f"Subtotal:          {subtotal:10,}")
        print(f"Sales tax:         {sales_tax:10,}")
        print(f"Invoice total:     {invoice_total:10,}")
        print()

        choice = input("Continue? (y/n): ")
        print()

print("Bye!")
```

The user interface for the Future Value program

```
Enter monthly investment:    100
Enter yearly interest rate: 12.5
Enter number of years:       10

Monthly investment:      $100.00
Interest rate:              12.5
Years:                        10
Future value:          $23,938.13

Continue? (y/n):
```

The code for the Future Value program (part 1)

```
from decimal import Decimal
import locale as lc

def get_future_value(monthly_investment, yearly_interest, years):
    monthly_interest_rate = yearly_interest / 12 / 100
    months = years * 12
    future_value = Decimal("0.00")
    for i in range(months):
        future_value += monthly_investment
        monthly_interest = future_value * monthly_interest_rate
        future_value += monthly_interest
    future_value = future_value.quantize(Decimal("1.00"))
    return future_value
```

The code for the Future Value program (part 2)

```
def main():
    choice = "y"
    while choice.lower() == "y":
        # convert user input to Decimal and int values
        monthly_investment = Decimal(
            input("Enter monthly investment:   "))
        yearly_interest = Decimal(
            input("Enter yearly interest rate: "))
        years = int(input("Enter number of years:      "))

        future_value = get_future_value(
            monthly_investment, yearly_interest, years)
        print()
```

The code for the Future Value program (part 3)

```
        lc.setlocale(lc.LC_ALL, "en_US")
        monthly_investment = lc.currency(monthly_investment,
                                         grouping=True)
        future_value = lc.currency(future_value, grouping=True)

        s1 = 20
        s2 = ">10"
        print(f"{'Monthly investment:':{s1}} {monthly_investment:{s2}}")
        print(f"{'Interest rate:':{s1}} {yearly_interest:{s2}}")
        print(f"{'Years:':{s1}} {years:{s2}}")
        print(f"{'Future value:':{s1}} {future_value:{s2}}")
        print()

        choice = input("Continue? (y/n): ")
        print()

if __name__ == "__main__":
    main()
```