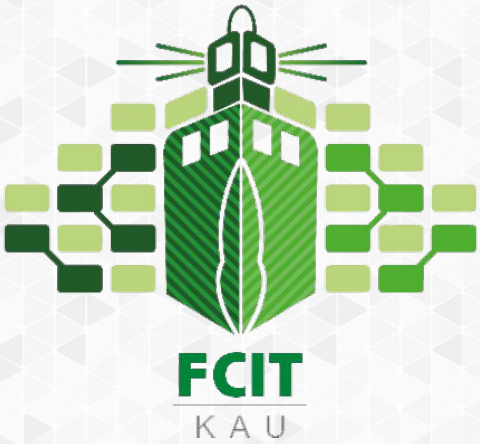
FACULTY OF COMPUTING
& INFORMATION TECHNOLOGY
KING ABDULAZIZ UNIVERSITY



كلية الحاسبات
وتقنية المعلومات
جامعة الملك عبدالعزيز

Chapter 2
Elementary Programming

CPIT 110 (Problem-Solving and Programming)

Introduction to Programming Using Python, By: Y. Daniel Liang

Sections

Python Online IDE

Programs

- Program 1: Compute Area
- Program 2: Compute Area With Console Input
- Program 3: Compute Average
- Program 4: Compute Average With Simultaneous Assignment
- Program 5: Compute Area with a Constant
- Program 6: Convert Time
- Problem 7: Keeping Two Digits After Decimal Points
- Problem 8: Displaying Current Time
- Problem 9: Computing Loan Payments

Check Points

- Section 2.2
  - #1
  - #2
- Section 2.3
  - #3
- Section 2.4
  - #4
- Section 2.6
  - #5
- Section 2.8
  - #6
  - #7
- Section 2.9
  - #8
  - #9
- Section 2.10
  - #10
- Section 2.11
  - #11
  - #12

Objectives

- To write programs that perform simple computations (2.2).
- To obtain input from a program's user by using the input function (2.3).
- To use identifiers to name elements such as variables and functions (2.4).
- To assign data to variables (2.5).
- To perform simultaneous assignment (2.6).
- To define named constants (2.7).
- To use the operators +, -, *, /, //, %, and ** (2.8).
- To write and evaluate numeric expressions (2.9).
- To use augmented assignment operators to simplify coding (2.10).
- To perform numeric type conversion and rounding with the int and round functions (2.11).
- To obtain the current system time by using time.time() (2.12).
- To describe the software development process and apply it to develop a loan payment program (2.13).
- To compute and display the distance between two points (2.14).

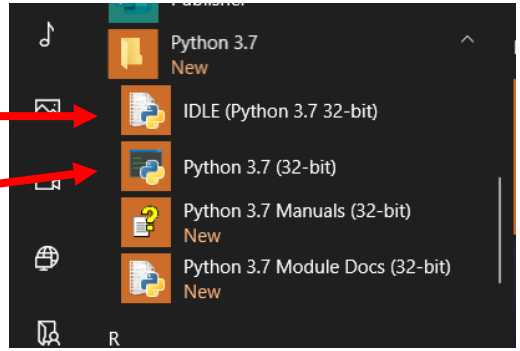
Get Ready

Get Ready

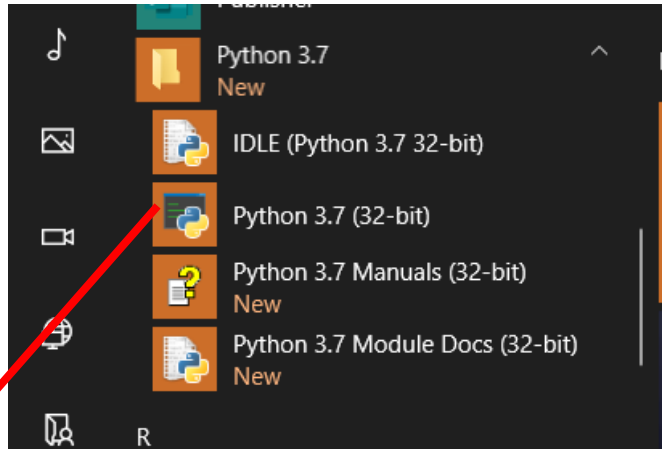
- In this chapter, you will learn many basic concepts in Python programming. So, You may need to try some codes in a quick way.
- You learned in the previous chapter that Python has two modes: interactive mode and script mode.
- In the interactive mode, you don't have to create a file to execute the code. Also, you don't have to use the print function to display results of expressions and values of variables.
- Python provides Python Shell for programming in the interactive mode.
- You can use Python Shell in form of command line using ("Python 3.7") or GUI (Graphical User Interface) using ("IDLE").

Python Shell (GUI)

Python Shell (Command Line)



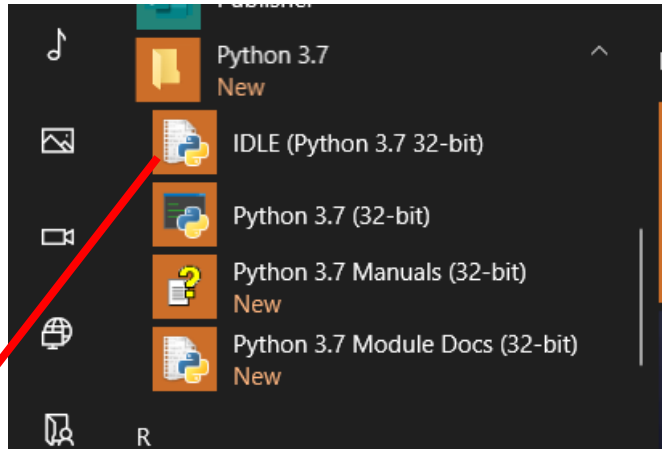
Python Shell (Command Line)



Python 3.7 (32-bit)

```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> x = "Hello World"
>>> x
'Hello World'
>>> 10 + 20
30
>>>
```

Python Shell (GUI)



Python 3.7.3 Shell

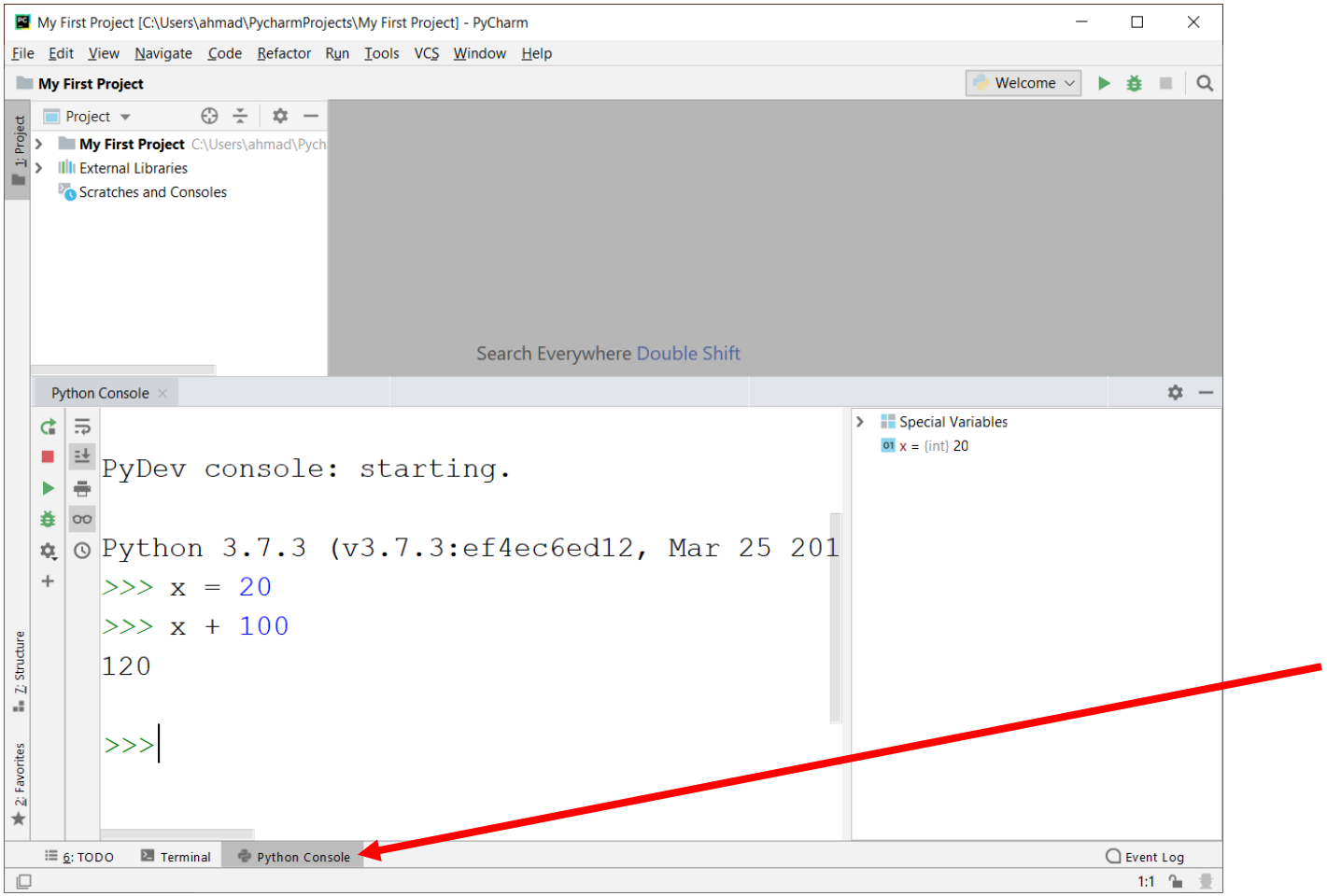
File Edit Shell Debug Options Window Help

```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> name = "Ahmad"
>>> print(name)
Ahmad
>>> 5 / 3
1.6666666666666667
>>>
```

Ln: 8 Col: 4

Python Shell in PyCharm

- Open or create a project in PyCharm.



2.1. Motivations

Motivations

- In the preceding chapter (Chapter 1), you learned how to create and run a Python program.
- Starting from this chapter, you will learn how to solve practical problems programmatically.
- Through these problems, you will learn Python basic data types and related subjects, such as variables, constants, data types, operators, expressions, and input and output.

2.2. Writing a Simple Program

- Program 1: Compute Area
- Data Types
- Python Data Types
- Check Point #1 - #2

Compute Area
Program 1

Write a program that will calculate the area of a circle.

area = radius x radius x π

Here is a sample run of the program (suppose that radius is **20**):

```
The area for the circle of radius 20 is 1256.636
```

- Remember:
  - Phase 1: Problem-solving
  - Phase 2: Implementation

Compute Area
Phase 1: Problem-solving

Write a program that will calculate the area of a circle.

- Phase 1: Design your algorithm
  1. Get the radius of the circle.
  2. Compute the area using the following formula:
     area = radius x radius x π
  3. Display the result

- Tip:
  It's always good practice to outline your program (or its underlying problem) in the form of an algorithm (Phase 1) before you begin coding (Phase 2).

Compute Area
Phase 2: Implementation

Write a program that will calculate the area of a circle.

- Phase 2: Implementation (code the algorithm)

ComputeArea.py

```
1  # Step 1: Assign a value to radius
2
3
4  # Step 2: Compute area
5
6
7  # Step 3: Display results
8
```

Compute Area
Phase 2: Implementation

Write a program that will calculate the area of a circle.

- Phase 2: Implementation (code the algorithm)
  - In this problem, the program needs to read the radius, which the program's user enters from the keyboard.
  - This raises two important issues:
    - Reading the radius from the user. → Solution: using the input function
    - Storing the radius in the program. → Solution: using variables
  - **Let's address the second issue first.**

Compute Area
Phase 2: Implementation

Write a program that will calculate the area of a circle.

- Phase 2: Implementation (code the algorithm)
  - In order to store the radius, the program must create a symbol called a variable.
  - A variable is a name that references a value stored in the computer's memory.
  - You should choose descriptive names for variables
    - Do not choose "x" or "y"… these have no meaning
    - Choose names with meaning …"area" or "radius"

Compute Area
Phase 2: Implementation

Write a program that will calculate the area of a circle.

- Phase 2: Implementation (code the algorithm)
  - The first step is to create and set a value for radius.
    - Later we will learn how to ask the user to input the value for radius!
    - For now, you can assign a fixed value to radius in the program as you write the code. For example, let radius be 20

ComputeArea.py

```
1  # Step 1: Assign a value to radius
2  radius = 20 # radius is now 20
3
4  # Step 2: Compute area
5
6
7  # Step 3: Display results
8
```

Compute Area
Phase 2: Implementation

Write a program that will calculate the area of a circle.

- Phase 2: Implementation (code the algorithm)
  - The second step is to compute area by assigning the result of the expression (radius * radius * 3.14159) to area.
    - Note that: **π = 3.14159**, so we can rewrite the equation (**area = radius x radius x π**) to be (**area = radius x radius x 3.14159**).

ComputeArea.py

```
1  # Step 1: Assign a value to radius
2  radius = 20 # radius is now 20
3
4  # Step 2: Compute area
5  area = radius * radius * 3.14159
6
7  # Step 3: Display results
8
```

Compute Area
Phase 2: Implementation

Write a program that will calculate the area of a circle.

- Phase 2: Implementation (code the algorithm)
  - The final step is to display the value of area on the console by using Python's print function.

ComputeArea.py

```
1  # Step 1: Assign a value to radius
2  radius = 20 # radius is now 20
3
4  # Step 2: Compute area
5  area = radius * radius * 3.14159
6
7  # Step 3: Display results
8  print("The area for the circle of radius ", radius, " is ", area)
```

Compute Area
Phase 2: Implementation

Write a program that will calculate the area of a circle.

- Phase 2: Implementation (code the algorithm)

LISTING 2.1 ComputeArea.py

```
1  # Assign a value to radius
2  radius = 20 # radius is now 20
3
4  # Compute area
5  area = radius * radius * 3.14159
6
7  # Display results
8  print("The area for the circle of radius ", radius, " is ", area)
```

Run

```
The area for the circle of radius  20  is  1256.636
```

Compute Area
Trace The Program Execution

It is a comment. Do Nothing.

LISTING 2.1 ComputeArea.py

```
1  # Assign a radius
2  radius = 20 # radius is now 20
3
4  # Compute area
5  area = radius * radius * 3.14159
6
7  # Display results
8  print("The area for the circle of radius ", radius, " is ", area)
```

Compute Area
Trace The Program Execution

Assign **20** to radius

LISTING 2.1 ComputeArea.py

```
1  # Assign a radius
2  radius = 20 # radius is now 20
3
4  # Compute area
5  area = radius * radius * 3.14159
6
7  # Display results
8  print("The area for the circle of radius ", radius, " is ", area)
```

radius | 20

Compute Area
Trace The Program Execution

It is a comment. Do Nothing.

LISTING 2.1 ComputeArea.py

```
1  # Assign a radius
2  radius = 20 # radius is now 20
3
4  # Compute area
5  area = radius * radius * 3.14159
6
7  # Display results
8  print("The area for the circle of radius ", radius, " is ", area)
```

radius 20

Compute Area
Trace The Program Execution

Assign the result (**1256.636**) to area

LISTING 2.1 ComputeArea.py

```
1  # Assign a radius
2  radius = 20 # radius is now 20
3
4  # Compute area
5  area = radius * radius * 3.14159
6
7  # Display results
8  print("The area for the circle of radius ", radius, " is ", area)
```

radius	20
area	**1256.636**

Compute Area
Trace The Program Execution

It is a comment. Do Nothing.

LISTING 2.1 ComputeArea.py

```
1  # Assign a radius
2  radius = 20 # radius is now 20
3
4  # Compute area
5  area = radius * radius * 3.14159
6
7  # Display results
8  print("The area for the circle of radius ", radius, " is ", area)
```

radius 20

area 1256.636

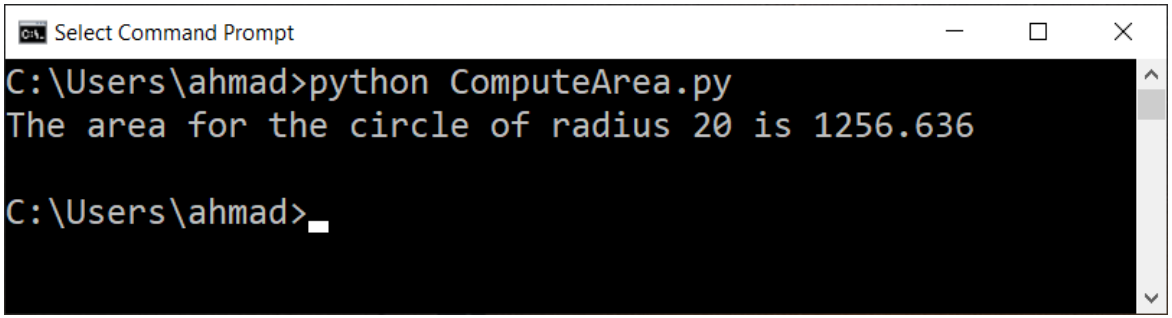
Compute Area
Trace The Program Execution

Print the message to the console.

LISTING 2.1 ComputeArea.py

```
1  # Assign a radius
2  radius = 20 # radius is now 20
3
4  # Compute area
5  area = radius * radius * 3.14159
6
7  # Display results
8  print("The area for the circle of radius ", radius, " is ", area)
```

radius	20
area	1256.636



Compute Area Discussion

LISTING 2.1 ComputeArea.py

```
1  # Assign a radius
2  radius = 20 # radius is now 20
3
4  # Compute area
5  area = radius * radius * 3.14159
6
7  # Display results
8  print("The area for the circle of radius ", radius, " is ", area)
```

- Variables such as radius and area reference values stored in memory.
- Every variable has a name that refers to a value.
- You can assign a value to a variable using the syntax as shown in line 2.

```
radius = 20
```

- This statement assigns 20 to the variable radius. So now radius references the value 20.

Compute Area Discussion

LISTING 2.1 ComputeArea.py

```
1  # Assign a radius
2  radius = 20 # radius is now 20
3
4  # Compute area
5  area = radius * radius * 3.14159
6
7  # Display results
8  print("The area for the circle of radius ", radius, " is ", area)
```

- The statement in line 5 uses the value in radius to compute the expression and assigns the result into the variable area.

```
area = radius * radius * 3.14159
```

Compute Area Discussion

LISTING 2.1 ComputeArea.py

```
1  # Assign a radius
2  radius = 20 # radius is now 20
3
4  # Compute area
5  area = radius * radius * 3.14159
6
7  # Display results
8  print("The area for the circle of radius ", radius, " is ", area)
```

- The following table shows the value in memory for the variables area and radius as the program is executed.

Line #	radius	area
2	20	It does not exist
5		1256.636

- This method of reviewing a program is called “tracing a program”.
- It helps you to understand how programs work.

Compute Area Discussion

LISTING 2.1 ComputeArea.py

```
1  # Assign a radius
2  radius = 20 # radius is now 20
3
4  # Compute area
5  area = radius * radius * 3.14159
6
7  # Display results
8  print("The area for the circle of radius ", radius, " is ", area)
```

- The statement in line 8 displays four items on the console.

```
print("The area for the circle of radius " , radius , " is " , area)
```

- You can display any number of items in a print statement using the following syntax:

```
print(item1, item2 , ..., itemk)
```

- If an item is a number, the number is automatically converted to a string for displaying.

Data Types

- A variable represents a value stored in the computer's memory.
- Every variable has a name and value.
- Every value has a data type, and the data type is to specify what type of the value are being used, such as integers or strings (text characters).
- In many programming languages such as Java, you have to define the type of the variable before you can use it. You don't do this in Python.
- However, Python automatically figures out the data type of a variable according to the value assigned to the variable.

Python Data Types

- Python provides basic (built-in) data types for integers, real numbers, string, and Boolean types.
- Here are some examples of different types of values stored in different variables:

```
var1 = 25          # Integer
var2 = 25.8        # Float
var3 = "Ahmad"     # String
var4 = 'Python'    # String
var5 = True        # Boolean
```

Check Point #1

Show the printout of the following code:

```
1 width = 5.5
2 height = 2
3 print("area is", width * height)
```

- Solution: area is 11.0

Check Point #2

Translate the following algorithm into Python code:

- Step 1: Use a variable named miles with initial value **100**.
- Step 2: Multiply miles by **1.609** and assign it to a variable named kilometers.
- Step 3: Display the value of kilometers.

What is kilometers after Step 3?

➢ Solution:

```
1 miles = 100
2 kilometers = miles * 1.609
3 print("kilometers = ", kilometers)
```

- kilometers after Step 3 is **160.9**

2.3. Reading Input from the Console

- input(…)
- eval(…)
- Program 2: Compute Area With Console Input
- Program 3: Compute Average
- Line Continuation Symbol (\)
- IPO
- Check Point #3

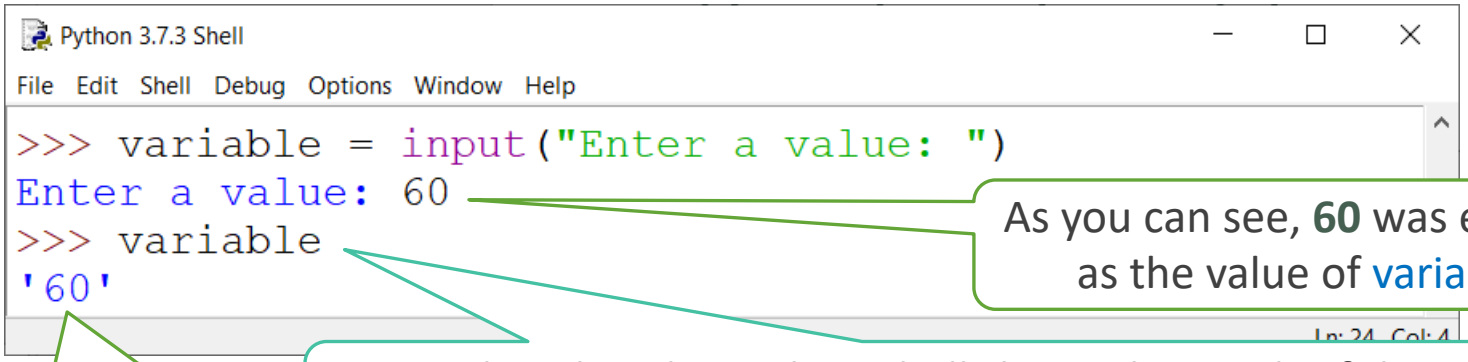
input(...)

- In the last example, the radius was fixed.
- To use a different radius, you have to modify the source code.
- Your program can be better, and more interactive, by letting the user enter the radius.
- Python uses the input function for console input.
- The input function is used to ask the user to input a value, and then return the value entered as a string (string data type).
- The following statement prompts (asks) the user to enter a value, and then it assigns the value to the variable:

```
variable = input("Enter a value: ")
```

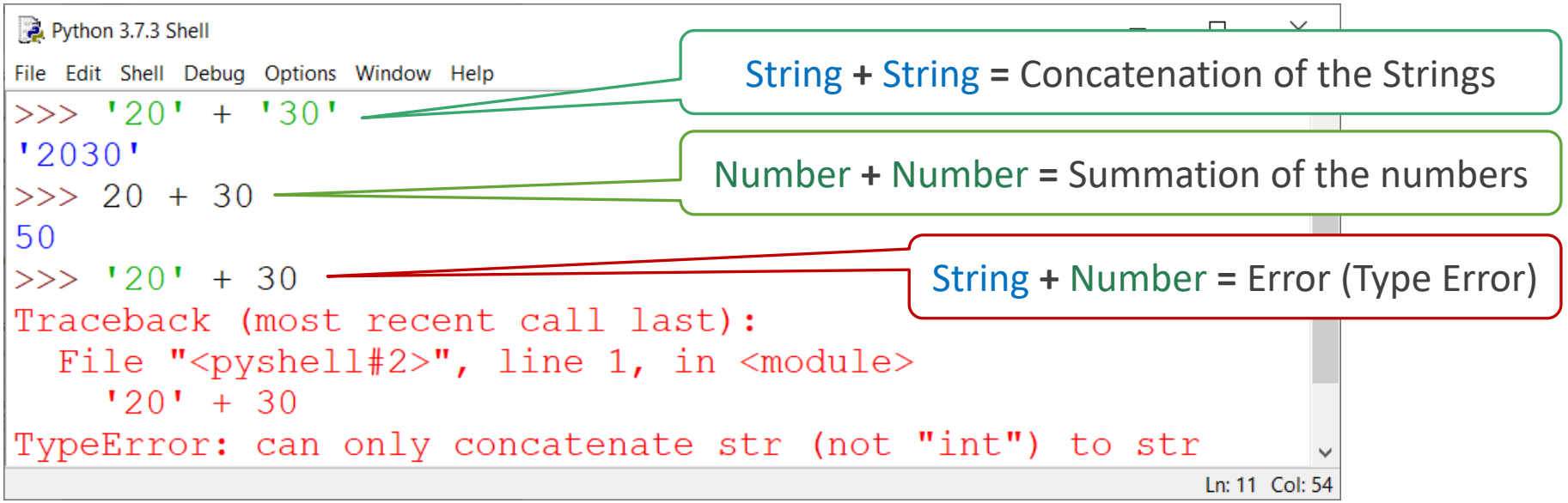
input(...)

- The example of the output of the previous statement is shown in the following figure (Python Shell – Interactive mode).



input(...)

- Does it matter if a numeric value is being stored as a string, not a number?
- **Yes, it does**. See the following examples:



eval(...)

- You can use eval function to evaluate and convert the passed value (string) to a numeric value.

```
1 eval("34.5")                    # returns 34.5 (float)
2 eval("345")                     # returns 345  (integer)
3 eval("3 + 4")                   # returns 7    (integer)
4 eval("51 + (54 * (3 + 2))")     # returns 321  (integer)
```

- So, to get an input (value) from the user as a number and store it into the variable x, you can write the following code:

```
x = eval(input("Enter the value of x: "))
```

- Also, you can separate the process into two lines if you would like:

```
x = input("Enter the value of x: ") # Read x as string
x = eval(x) # Convert value x to number and save it to x
```

Compute Area With Console Input
Program 2

- Now, let us revisit the last example (Program 1), and modify it in order to let the user enter the radius value.

LISTING 2.2 ComputeAreaWithConsoleInput.py

```
1  # Prompt the user to enter a radius
2  radius = eval(input("Enter a value for radius: "))
3
4  # Compute area
5  area = radius * radius * 3.14159
6
7  # Display results
8  print("The area for the circle of radius " , radius , " is " , area)
```

Run

```
Enter a value for radius: 2.5  <Enter>
The area for the circle of radius 2.5 is 19.6349375
```

```
Enter a value for radius: 23  <Enter>
The area for the circle of radius 23 is 1661.90111
```

Compute Area With Console Input Discussion

LISTING 2.2 ComputeAreaWithConsoleInput.py

```
1  # Prompt the user to enter a radius
2  radius = eval(input("Enter a value for radius: "))
3
4  # Compute area
5  area = radius * radius * 3.14159
6
7  # Display results
8  print("The area for the circle of radius " , radius , " is " , area)
```

- Line 2 prompts the user to enter a value (in the form of a string) and converts it to a number, which is equivalent to:

```
# Read input as a string
radius = input("Enter a value for radius: ")
# Convert the string to a number
radius = eval(radius)
```

- After the user enters a number and presses the <Enter> key, the number is read and assigned to radius.

Compute Average
Program 3

Write a program to get three values from the user and compute their average.

```
Enter the first number: 1   <Enter>
Enter the second number: 2  <Enter>
Enter the third number: 3   <Enter>
The average of 1 2 3 is 2.0
```

- Remember:
  - Phase 1: Problem-solving
  - Phase 2: Implementation

Compute Average
Phase 1: Problem-solving

Write a program to get three values from the user and compute their average.

- Phase 1: Design your algorithm
  1. Get three numbers from the user.
     - Use the input function.
  2. Compute the average of the three numbers: **average = (num1 + num2 + num3) / 3**
  3. Display the result

Compute Average
Phase 2: Implementation

Write a program to get three values from the user and compute their average.

- Phase 2: Implementation (code the algorithm)

```
# Prompt the user to enter three numbers

# Compute average

# Display result
```

Compute Average
Phase 2: Implementation

Write a program to get three values from the user and compute their average.

- Phase 2: Implementation (code the algorithm)

LISTING 2.3 ComputeAverage.py

```
1  # Prompt the user to enter three numbers
2  number1 = eval(input("Enter the first number: "))
3  number2 = eval(input("Enter the second number: "))
4  number3 = eval(input("Enter the third number: "))
5
6  # Compute average
7  average = (number1 + number2 + number3) / 3
8
9  # Display result
10 print("The average of", number1, number2, number3,
11 "is", average)
```

Compute Average
Example Runs of The Program

```
Enter the first number: 1  <Enter>
Enter the second number: 2  <Enter>
Enter the third number: 3  <Enter>
The average of 1 2 3 is 2.0
```

```
Enter the first number: 10.5  <Enter>
Enter the second number: 11  <Enter>
Enter the third number: 11.5  <Enter>
The average of 10.5 11 11.5 is 11.0
```

Compute Average Discussion

LISTING 2.3 ComputeAverage.py

```
1  # Prompt the user to enter three numbers
2  number1 = eval(input("Enter the first number: "))
3  number2 = eval(input("Enter the second number: "))
4  number3 = eval(input("Enter the third number: "))
5
6  # Compute average
7  average = (number1 + number2 + number3) / 3
8
9  # Display result
10 print("The average of", number1, number2, number3,
11 "is", average)
```

- The program prompts the user to enter three integers (lines 2–4), computes their average (line 7), and displays the result (lines 10–11).
- If the user enters something other than a number, the program will terminate with a runtime error.

Compute Average Discussion

LISTING 2.3 ComputeAverage.py

```
1  # Prompt the user to enter three numbers
2  number1 = eval(input("Enter the first number: "))
3  number2 = eval(input("Enter the second number: "))
4  number3 = eval(input("Enter the third number: "))
5
6  # Compute average
7  average = (number1 + number2 + number3) / 3
8
9  # Display result
10 print("The average of", number1, number2, number3,
11 "is", average)
```

- Normally a statement ends at the end of the line.
- In Line 10, the print statement is split into two lines (lines 10–11).
- This is okay, because Python scans the print statement in line 10 and knows it is not finished until it finds the closing parenthesis in line 11.
- We say that these two lines are joined implicitly.

Line Continuation Symbol (\)

- In some cases, the Python interpreter cannot determine the end of the statement written in multiple lines. You can place the line continuation symbol (\) at the end of a line to tell the interpreter that the statement is continued on the next line.
- For example, the following statement:

```
1  sum = 1 + 2 + 3 + 4 + \
2        5 + 6
```

is equivalent to

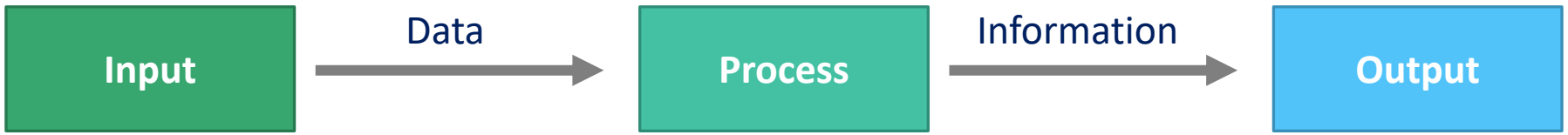
```
1  sum = 1 + 2 + 3 + 4 + 5 + 6
```

- Note that the following statement will cause a syntax error:

```
1  sum = 1 + 2 + 3 + 4 +
2        5 + 6
```

IPO

- Most of the programs in early chapters of this book perform three steps: **I**nput, **P**rocess, and **O**utput, called **IPO**.
- Input is to receive input from the user.
- Process is to produce results using the input.
- Output is to display the results.



Check Point #3

What happens if the user enters 5a when executing the following code?

```
1 radius = eval(input("Enter a radius: "))
```

- Answer: Runtime error

2.4. Identifiers

- Python Keywords
- Check Point #4

Identifiers

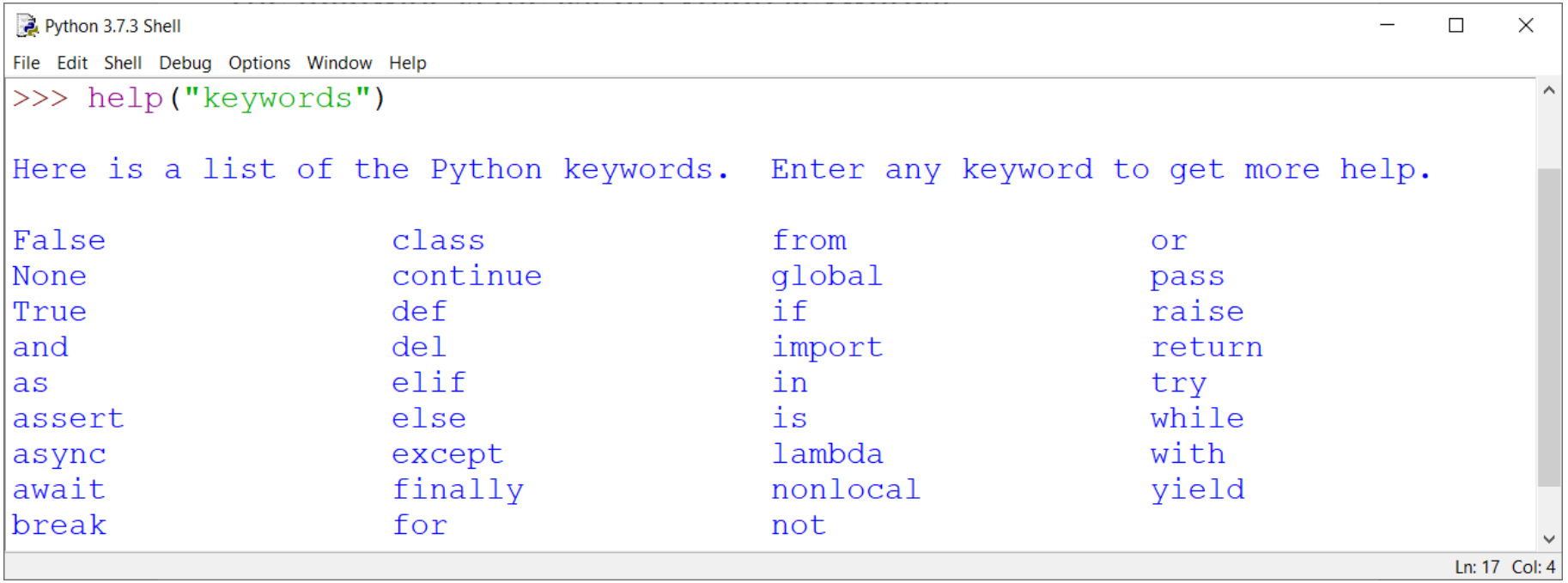
- Identifiers are the names that identify the elements such as variables and functions in a program.
- All identifiers must obey the following rules:
  - An identifier is a sequence of characters that consists of letters, digits, and underscores (_).
  - An identifier must start with a letter or an underscore. It cannot start with a digit.
  - An identifier cannot be a Keyword.
    - Keywords, also called reserved words, have special meanings in Python.
    - For example, import is a keyword, which tells the Python interpreter to import a module to the program.
  - An identifier can be of any length.

Identifiers

- Examples of legal identifiers:
  - area , radius, ComputeArea, _2, average, If, IN
- Examples of illegal identifiers:
  - 2A, d+4, a*, test#, @hmad, if, in
    - These do not follow the rules.
    - if and in are keywords in Python.
    - Python will report that you have a syntax error!
- Note: Python is case sensitive.
  - area, Area, and all are different identifiers.
    - AREA

Python Keywords

- Keywords are reserved words by programming language.
- Keywords can not be used as identifiers.
- The following is the list of Python keywords:



Check Point #4

Which of the following identifiers are valid? Which are Python keywords?

1. miles → ✓
2. Test → ✓
3. a+b → ✗
4. b–a → ✗
5. 4#R → ✗
6. $4 → ✗
7. #44 → ✗
8. apps → ✓
9. elif → ✗ (Keyword)
10. if → ✗ (Keyword)
11. y → ✓
12. iF → ✓

2.5. Variables, Assignment Statements, and Expressions

- Variables
- Assignment Statements
- Expression
- Assigning a Value To Multiple Variables
- Scope of Variables

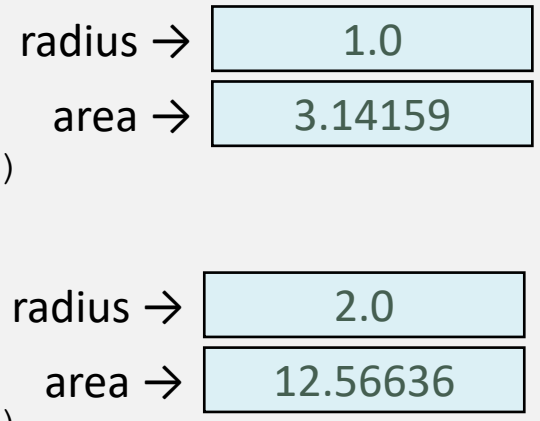
Variables

- Variables are used to reference (represent) values that may be changed in the program.
  - In the previous programs, we used variables to store values: area, radius, average.
- They are called variables because their values can be changed!

Variables

- For example, see the following code:

```
1  # Compute the first area
2  radius = 1.0
3  area = radius * radius * 3.14159
4  print("The area is", area, "for radius", radius)
5
6  # Compute the second area
7  radius = 2.0
8  area = radius * radius * 3.14159
9  print("The area is", area, "for radius", radius)
```



- Discussion:
  - radius is initially 1.0 (line 2)
    then changed to 2.0 (line 7)
  - area is set to 3.14159 (line 3)
    then reset to 12.56636 (line 8)

Assignment Statements

- The statement for assigning a value to a variable is called an assignment statement.
- In Python, the equal sign (=) is used as the assignment operator. The syntax for assignment statements is as follows:

```
variable = value
```

- or

```
variable = expression
```

Expression

- An expression represents a computation involving values, variables, and operators that, taken together, evaluate to a value.
- For example, consider the following code:

```
1 y = 1                      # Assign 1 to variable y
2 radius = 1.0               # Assign 1.0 to variable radius
3 x = 5 * (3 / 2) + 3 * 2 # Assign the value of the expression to x
4 x = y + 1                  # Assign the addition of y and 1 to x
5 area = radius * radius * 3.14159 # Compute area
```

- You can use a variable in an expression.

Expression

- A variable can also be used in both sides of the = operator.
- For example:

```
x = x + 1
```

- In this assignment statement, the result of x + 1 is assigned to x. If x is 1 before the statement is executed, then it becomes 2 after the statement is executed.
- If x is not created before, Python will report an error.

Note

- In mathematics, **x = 2 * x + 1** denotes an equation.
- However, in Python, **x = 2 * x + 1** is an assignment statement that evaluates the expression **2 * x + 1** and assigns the result to x.

Assigning a Value To Multiple Variables

- If a value is assigned to multiple variables, you can use a syntax like this:

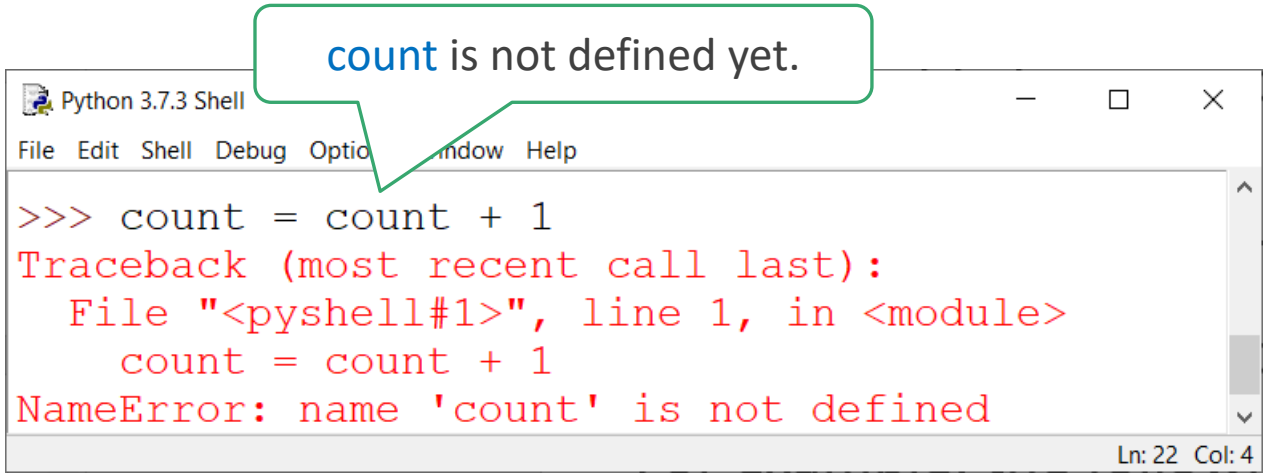
```
i = j = k = 1
```

- which is equivalent to

```
k = 1
j = k
i = j
```

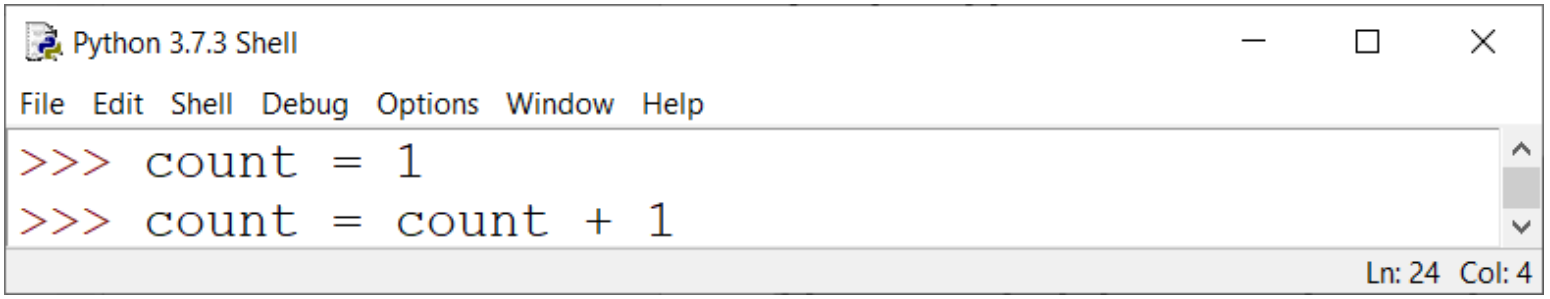
Scope of Variables

- Every variable has a scope.
- The scope of a variable is the part of the program where the variable can be referenced (used).
- A variable must be created before it can be used.
- For example, the following code is wrong:



Scope of Variables

- To fix the previous example, you may write the code like this:



Caution

- A variable must be assigned a value before it can be used in an expression.
- For example:

```
interestRate = 0.05
interest = interestrate * 45
```

- This code is wrong, because interestRate is assigned a value 0.05, but interestrate is not defined.
- Python is case-sensitive.
- interestRate and interestrate are two different variables.

2.6. Simultaneous Assignments

- Swapping Variable Values
- Obtaining Multiple Input In One Statement
- Program 4: Compute Average With Simultaneous Assignment
- Check Point #5

Simultaneous Assignment

- Python also supports simultaneous assignment in syntax like this:

```
var1, var2, ..., varn = exp1, exp2, ..., expn
```

- It tells Python to evaluate all the expressions on the right and assign them to the corresponding variable on the left simultaneously.
- Example:

```
x, y, name = 15, 20.5, "Ahmad"
```

Swapping Variable Values

- Swapping variable values is a common operation in programming and simultaneous assignment is very useful to perform this operation.
- Consider two variables: x and y. How do you write the code to swap their values? A common approach is to introduce a temporary variable as follows:

```
x = 1
y = 2
temp = x # Save x in a temp variable
x = y # Assign the value in y to x
y = temp # Assign the value in temp to y
```

- But you can simplify the task using the following statement to swap the values of x and y.

```
x, y = y, x # Swap x with y
```

Obtaining Multiple Input In One Statement

- Simultaneous assignment can also be used to obtain multiple input in one statement.
- Program 3 gives an example that prompts the user to enter three numbers and obtains their average.
- This program can be simplified using a simultaneous assignment statement, as shown in the following slide (Program 4).

Program 4: Compute Average With Simultaneous Assignment

LISTING 2.4 ComputeAverageWithSimultaneousAssignment.py

```
1  # Prompt the user to enter three numbers
2  number1, number2, number3 = eval(input(
3  "Enter three numbers separated by commas: "))
4
5  # Compute average
6  average = (number1 + number2 + number3) / 3
7
8  # Display result
9  print("The average of", number1, number2, number3,
10  "is", average)
```

Run

```
Enter three numbers separated by commas: 1, 2, 3  <Enter>
The average of 1 2 3 is 2.0
```

Check Point #5

Assume that a = **1** and b = **2**. What is a and b after the following statement?

```
a, b = b, a
```

- Answer:
  a = **2**
  b = **1**

2.7. Named Constants

- Program 5: Compute Area with a Constant
- Benefits of Using Constants
- Naming Conventions

Named Constants

- A named constant is an identifier that represents a permanent value.
- The value of a variable can change during execution of a program.
- However, a named constant, or simply constant, represents a permanent data that never changes.
- Python does not have a special syntax for naming constants.
- You can simply create a variable to denote a constant. To distinguish a constant from a variable, use all uppercase letters to name a constant.
- Example:

```
PI = 3.14159 # This is a constant
```

Compute Area with a Constant
Program 5

- In our Compute Area program (Program 1), **π** is a constant.
- If you use it frequently, you don't want to keep typing 3.14159; instead, you can use a descriptive name PI for the value.

```
1  # Assign a radius
2  radius = 20 # radius is now 20
3
4  # Compute area
5  PI = 3.14159
6  area = radius * radius * PI
7
8  # Display results
9  print("The area for the circle of radius", radius, "is", area)
```

Run

Benefits of Using Constants

1. You don’t have to repeatedly type the same value if it is used multiple times.
2. If you have to change the constant’s value (for example, from 3.14 to 3.14159 for PI), you need to change it only in a single location in the source code.
3. Descriptive names make the program easy to read.

Naming Conventions

- Choose meaningful and descriptive names.
  - Do not use abbreviations.
    - For example: use average instead of avg.

Naming Conventions

Variables and function names:

- Use lowercase.
  - For example: radius, area.
- If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name.
  - This naming style is known as the camelCase.
  - For example: computeArea, interestRate, yourFirstName.
- **Or** use lowercase for all words and concatenate them using underscore (_).
  - For example: compute_area, interest_rate, your_first_name.

Naming Conventions

- **Constants:**
  - Capitalize all letters in constants and use underscores to connect words.
    - For example: PI, MAX_VALUE.

- Do you have to follow these rules?
  - No. But it makes your program much easier to read!

2.8. Numeric Data Types and Operators

- Numeric Data Types
- Numeric Operators
- Unary Operator & Binary Operator
- Float Division (/) Operator
- Integer Division (//) Operator
- Exponentiation (**) Operator
- Remainder (%) Operator
- Program 6: Convert Time
- Check Point #6 - #7

Numeric Data Types

- The information stored in a computer is generally referred to as data.
- There are two types of numeric data: integers and real numbers.
- Integer types (int for short) are for representing whole numbers.
- Real types are for representing numbers with a fractional part.
- Inside the computer, these two types of data are stored differently.
- Real numbers are represented as floating-point (or float) values.

Numeric Data Types

- How do we tell Python whether a number is an integer or a float?
- A number that has a decimal point is a float even if its fractional part is 0.
- For example, 1.0 is a float, but 1 is an integer.
- In the programming terminology, numbers such as 1.0 and 1 are called literals.
- A literal is a constant value that appears directly in a program.

```
n1 = 5                  # Integer
n2 = 5.0                # Float
n3 = 10 + 20            # Integer -> 30
n4 = 10.0 + 20.0        # Float   -> 30.0
n5 = 10.0 + 20          # Float   -> 30.0
```

Numeric Operators

TABLE 2.1 Numeric Operators

Name	*Meaning*	*Example*	*Result*
`+`	Addition	`34 + 1`	`35`
`-`	Subtraction	`34.0 - 0.1`	`33.9`
`*`	Multiplication	`300 * 30`	`9000`
`/`	Float Division	`1 / 2`	`0.5`
`//`	Integer Division	`1 // 2`	`0`
`**`	Exponentiation	`4 ** 0.5`	`2.0`
`%`	Remainder	`20 % 3`	`2`

Unary Operator & Binary Operator

- The **+**, **-**, and ***** operators are straightforward, but note that the **+** and **-** operators can be both unary and binary.
- A unary operator has only one operand; a binary operator has two.
- For example, the **-** operator in -5 is a unary operator to negate the number 5, whereas the **–** operator in 4 - 5 is a binary operator for subtracting 5 from 4.

```
e1 = -10 + 50    # 40
e2 = -10 + -50   # -60
e3 = +10 + +20   # 30
e4 = +10++20     # 30
e5 = 10++20      # 30
e6 = -20--30     # 10
```

Float Division (/) Operator

- The **/** operator performs a float division that results in a floating number. For example:


```
>>> 4 / 2
2.0
>>> 2 / 4
0.5
>>>
```

Integer Division (//) Operator

- The **//** operator performs an integer division; the result is an integer, and any fractional part is truncated. For example:


```
>>> 5 // 2
2
>>> 2 // 4
0
>>>
```

Exponentiation (**) Operator

- To compute a^b (a with an exponent of b) for any numbers a and b, you can write a ** b in Python. For example:

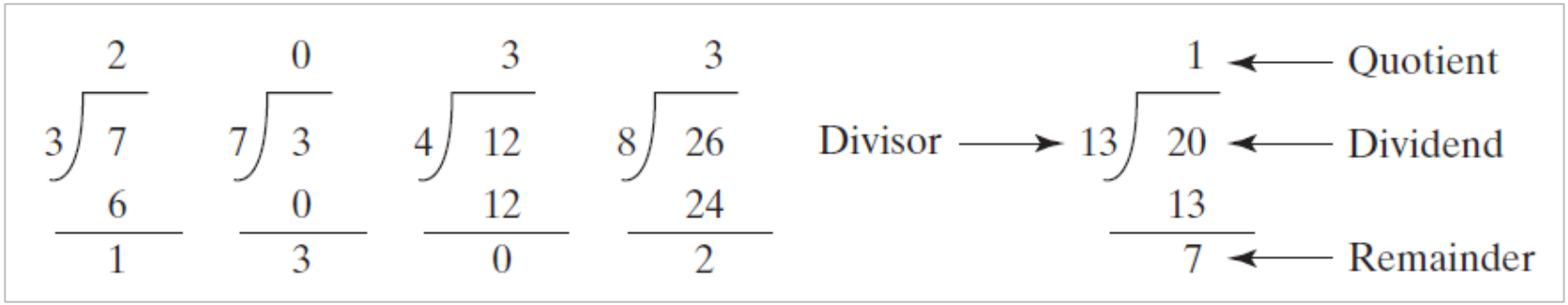

```
>>> 2.3 ** 3.5
18.45216910555504
>>> (-2.5) ** 2
6.25
>>>
```

Remainder (%) Operator

- The % operator, known as remainder or modulo operator, yields the remainder after division.
- The left-side operand is the dividend and the right-side operand is the divisor.
- Examples:

  7 % 3 = 1 　　3 % 7 = 3 　　12 % 4 = 0

  26 % 8 = 2 　　20 % 13 = 7



Remainder (%) Operator

- Remainder is very useful in programming.
- For example, an even number % 2 is always 0
- An odd number % 2 is always 1
- So you can use this property to determine whether a number is even or odd.
- You can also mod by other values to achieve valuable results.


```
>>> 100 % 2
0
>>> 99 % 2
1
>>>
```

Remainder (%) Operator Example

- If today is Friday, it will be Friday again in 7 days. Suppose you and your friends will meet in 10 days. What day is it in 10 days?
- Let us assume Sunday is the 1st day of the week.

1	2	3	4	5	6	7 (0)
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday

7 % 7 = 0

- We can find that in 10 days, the day will be Monday by using the following equation:

Friday is the 6th day in a week

A week has 7 days

$$(6 + 10) \ \% \ 7 \text{ is } 2$$

After 10 days

The 2nd day in a week is Monday

Convert Time
Program 6

Write a program to get an amount of time from the user in seconds. Then your program should convert this time into minutes and the remaining seconds.

```
Enter an integer for seconds: 60  <Enter>
60 seconds is 1 minutes and 0 seconds
```

```
Enter an integer for seconds: 500  <Enter>
500 seconds is 8 minutes and 20 seconds
```

- Remember:
  - Phase 1: Problem-solving
  - Phase 2: Implementation

Convert Time
Phase 1: Problem-solving

- If you are given seconds, how do you then calculate the minutes and remaining seconds?
- Example:
  - Given 624 seconds, how do we calculate the minutes?
  - We divide by 60!
    - We see how many complete 60s are in 624.
    - Answer: 10 of them. 10 x 60 = 600.
  - So in 624 seconds, there are a full 10 minutes.
  - After we remove those 10 minutes, how many seconds are remaining?
    - 624 - (10 x 60) = 24 seconds remaining
    - We can use mod! 624 % 60 = 24 seconds remaining

Convert Time
Phase 1: Problem-solving

- Design your algorithm:
  1. Get amount of seconds from the user.
     - Use input function
  2. Compute the minutes and seconds remaining:
     - From these seconds, determine the number of minutes
     - Example:
       - 150 seconds => 2 minutes and 30 seconds
         - **150 // 60 = 2** and **150 % 60 = 30**
       - 315 seconds => 5 minutes and 15 seconds
         - **315 // 60 = 5** and **315 % 60 = 15**
  3. Display the result

Convert Time
Phase 2: Implementation

LISTING 2.5 DisplayTime.py

```
1  # Prompt the user for input
2  seconds = eval(input("Enter an integer for seconds: "))
3
4  # Get minutes and remaining seconds
5  minutes = seconds // 60 # Find minutes in seconds
6  remainingSeconds = seconds % 60 # Seconds remaining
7  print(seconds, "seconds is", minutes,
8  "minutes and", remainingSeconds, "seconds")
```

Run

```
Enter an integer for seconds: 150  <Enter>
150 seconds is 2 minutes and 30 seconds
```

```
Enter an integer for seconds: 315  <Enter>
315 seconds is 5 minutes and 15 seconds
```

Convert Time
Trace The Program Execution

```
Enter an integer for seconds: 500  <Enter>
500 seconds is 8 minutes and 20 seconds
```

line#	seconds	minutes	remainingSeconds
2	500		
5		8	
6			20

LISTING 2.5 DisplayTime.py

```
1  # Prompt the user for input
2  seconds = eval(input("Enter an integer for seconds: "))
3
4  # Get minutes and remaining seconds
5  minutes = seconds // 60 # Find minutes in seconds
6  remainingSeconds = seconds % 60 # Seconds remaining
7  print(seconds, "seconds is", minutes,
8  "minutes and", remainingSeconds, "seconds")
```

Note

- Calculations involving floating-point numbers are approximated because these numbers are not stored with complete accuracy.
- For example:

```
print(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1)
```

  - displays **0.5000000000000001**, not **0.5**, and:

```
print(1.0 - 0.9)
```

  - displays **0.09999999999999998**, not **0.1**.
- Integers are stored precisely.
- Therefore, calculations with integers yield a precise integer result.

Overflow

- When a variable is assigned a value that is too large (in size) to be stored, it causes overflow.
- For example, executing the following statement causes overflow.

Python

```
>>> 245.0 ** 1000
OverflowError: 'Result too large'
>>>
```

Underflow

- When a floating-point number is too small (for example, too close to zero) to be stored, it causes underflow.
- Python approximates it to zero. So normally you should not be concerned with underflow.

Scientific Notation

- Floating-point literals can also be specified in scientific notation.
- Example:
  - 1.23456e+2, same as 1.23456e2, is equivalent to 123.456
  - and 1.23456e-2 is equivalent to 0.0123456
  - E (or e) represents an exponent and it can be either in lowercase or uppercase.


```
>>> 20e2
2000.0
>>> 123.456e3
123456.0
>>> 20e3
20000.0
```


```
>>> 123.456e2
12345.6
>>> 123.456e-2
1.23456
>>>
```

Check Point #6

What are the results of the following expressions?

Expression	Result
42 / 5	8.4
42 // 5	8
42 % 5	2
40 % 5	0
1 % 2	1
2 % 1	0
45 + 4 * 4 - 2	59
45 + 43 % 5 * (23 * 3 % 2)	48
5 ** 2	25
5.1 ** 2	26.009999999999998

Check Point
#7

If today is Tuesday, what day of the week will it be in 100 days?
Suppose that Saturday is 1st day in a week.

- Answer: Thursday

1	2	3	4	5	6	7 (0)
Saturday	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday

Tuesday is the 4th day in a week

A week has 7 days

$(4 + 100) \ \% \ 7$ is 6

After 100 days

The 6th day in a week is Thursday

2.9. Evaluating Expressions and Operator Precedence

- Arithmetic Expressions
- How to Evaluate an Expression
- Check Point #8 - #9

Arithmetic Expressions

- Python expressions are written the same way as normal arithmetic expressions.
- Example:

$$\frac{3+4x}{5} - \frac{10(y-5)(a+b+c)}{x} + 9\left(\frac{4}{x} + \frac{9+x}{y}\right)$$

- It is translated into:

```
((3 + (4 * x)) / 5) - ((10 * (y - 5) * (a + b + c)) / x) + (9 * ((4 / x) + ((9 + x) / y)))
```

Zoom In

```
((3 + (4 * x)) / 5) - ((10 * (y - 5) * (a + b +
    c)) / x) + (9 * ((4 / x) + ((9 + x) / y))
```

Arithmetic Expressions
Explanation

3 + (4 * x)

10 * (y – 5) * (a + b + c)

$$\frac{3+4x}{5} - \frac{10(y-5)(a+b+c)}{x} + 9\left(\frac{4}{x} + \frac{9+x}{y}\right)$$

Expression

$$\frac{3 + (4 * x)}{5} - \frac{10 * (y - 5) * (a + b + c)}{x} + 9\left(\frac{4}{x} + \frac{9+x}{y}\right)$$

Arithmetic Expressions
Explanation

((3 + (4 * x)) / 5)

((10 * (y – 5) * (a + b + c)) / x)

$$\frac{3 + (4 * x)}{5} - \frac{10 * (y - 5) * (a + b + c)}{x} + 9\left(\frac{4}{x} + \frac{9 + x}{y}\right)$$

Expression

$$((3 + (4 * x)) / 5) - ((10 * (y - 5) * (a + b + c)) / x) + 9\left(\frac{4}{x} + \frac{9 + x}{y}\right)$$

Arithmetic Expressions
Explanation

$$(((3 + (4 * x)) / 5) - (((10 * (y - 5) * (a + b + c)) / x) + 9\left(\frac{4}{x} + \frac{9 + x}{y}\right)$$

((9 + x) / y)

(4 / x)

Expression

$$(((3 + (4 * x)) / 5) - (((10 * (y - 5) * (a + b + c)) / x) + 9((4 / x) + (((9 + x) / y))$$

Arithmetic Expressions
Explanation

(9 * ((4 / x) +((9 + x) / y)))

$$((3 + (4 * x)) / 5) - (((10 * (y - 5) * (a + b + c)) / x) + 9((4 / x) + ((9 + x) / y))$$

Expression

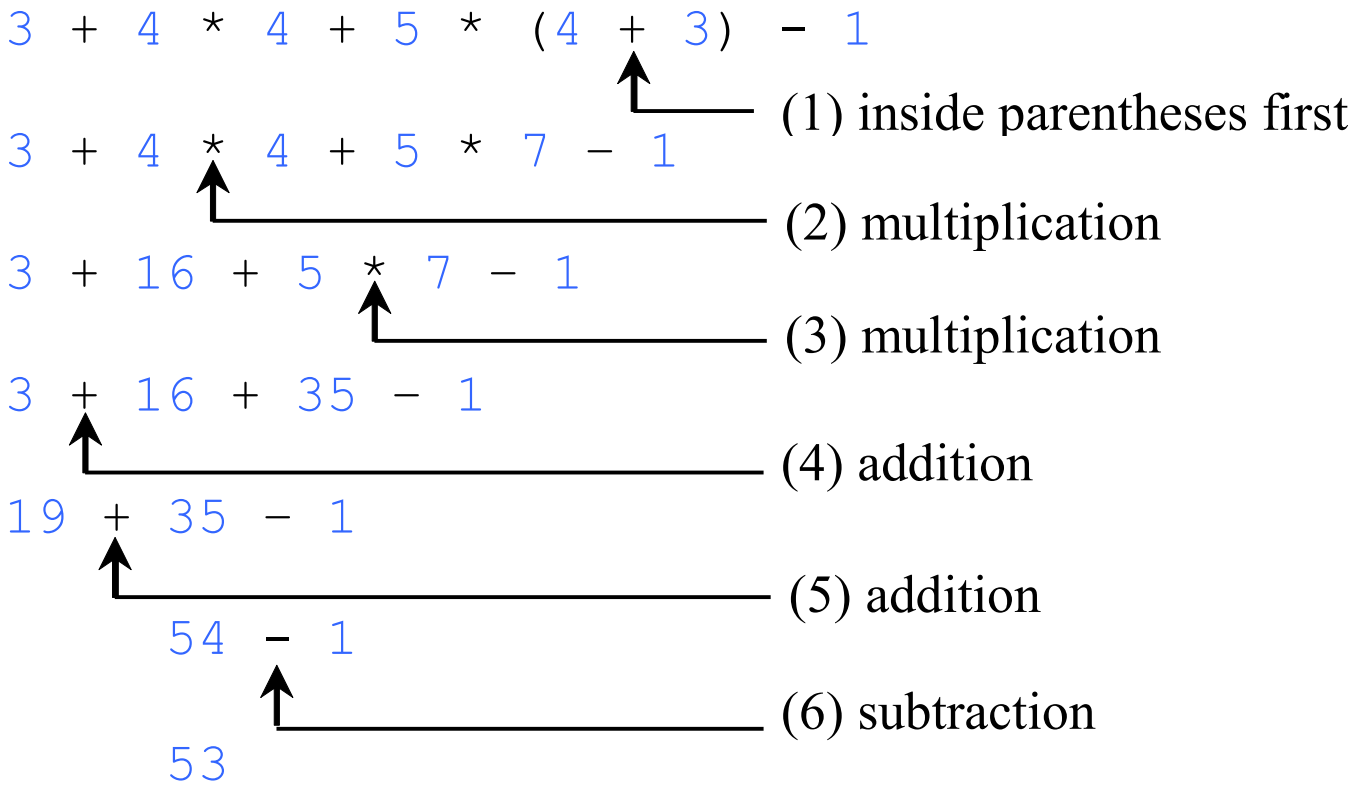
$$((3 + (4 * x)) / 5) - (((10 * (y - 5) * (a + b + c)) / x) + (9 * ((4 / x) + ((9 + x) / y))))$$

How to Evaluate an Expression

- You can safely apply the arithmetic rule for evaluating a Python expression.
  1. Operators inside parenthesis are evaluated first.
     - Parenthesis can be nested
     - Expression in inner parenthesis is evaluated first
  2. Use operator precedence rule.
     - Exponentiation (**) is applied first.
     - Multiplication (*), float division (/), integer division (//) , and remainder operators (%) are applied next.
       - If an expression contains several multiplication, division, and remainder operators, they are applied from left to right.
     - Addition (+) and subtraction (-) operators are applied last.
       - If an expression contains several addition and subtraction operators, they are applied from left to right.

How to Evaluate an Expression

- Example of how an expression is evaluated:



Check Point
#8

How would you write the following arithmetic expression in Python?

$$\frac{4}{3(r+34)} - 9(a+bc) + \frac{3+d(2+a)}{a+bd}$$

- Solution:

```
(4 / (3 * (r + 34))) - (9 * (a + (b * c))) + ((3 + (d * (2 + a))) / (a + (b * d)))
```

Check Point #9

Suppose m and r are integers. Write a Python expression for mr^2.

- Solution:

```
m * (r ** 2)
```

2.10. Augmented Assignment Operators

- Check Point #10

Augmented Assignment Operators

- Very often the current value of a variable is used, modified, and then reassigned back to the same variable.
- For example, the following statement increases the variable count by 1:

```
count = count + 1
```

- Python allows you to combine assignment and addition operators using an augmented (or compound) assignment operator.
- For example:

```
count += 1
```

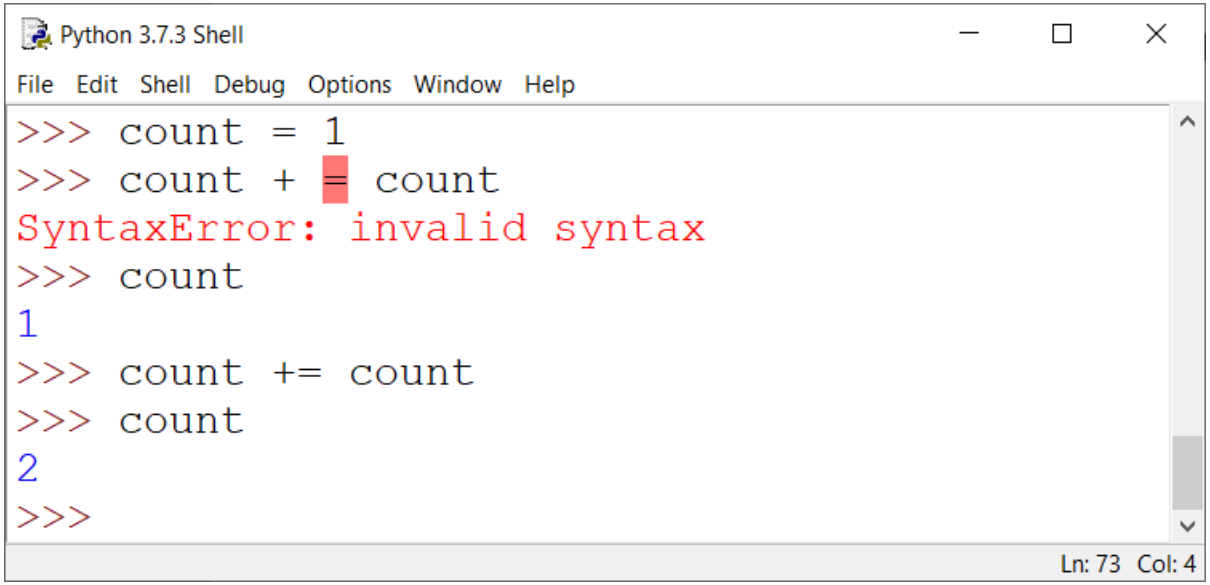
Augmented Assignment Operators

TABLE 2.2 Augmented Assignment Operators

Operator	*Name*	*Example*	*Equivalent*
`+=`	Addition assignment	`i += 8`	`i = i + 8`
`-=`	Subtraction assignment	`i -= 8`	`i = i - 8`
`*=`	Multiplication assignment	`i *= 8`	`i = i * 8`
`/=`	Float division assignment	`i /= 8`	`i = i / 8`
`//=`	Integer division assignment	`i //= 8`	`i = i // 8`
`%=`	Remainder assignment	`i %= 8`	`i = i % 8`
`**=`	Exponent assignment	`i **= 8`	`i = i ** 8`

Caution

- There are no spaces in the augmented assignment operators.
- For example, + = should be +=



Note

- The augmented assignment operator is performed last after all the other operators in the expression are evaluated.
- Example:

```
x /= 4 + 5.5 * 1.5
```

is same as

```
x = x / (4 + 5.5 * 1.5)
```

Check Point #10

Assume that a = **1**, and that each expression is independent. What are the results of the following expressions?

- a += 4 → 5
- a -= 4 → -3
- a *= 4 → 4
- a /= 4 → 0.25
- a //= 4 → 0
- a %= 4 → 1
- a = 56 * a + 6 → 62

2.11. Type Conversions and Rounding

- Type Conversions
- int(…)
- round(…)
- Int(…) vs. eval(…)
- str(…)
- Problem 7: Keeping Two Digits After Decimal Points
- Check Point #11 - #12

Type Conversions

- Can you perform binary operations with operands of different numeric types?
  - Meaning, can we add an integer literal with a floating-point literal?
- **Yes**. If an integer and a float are involved in a binary operation, Python automatically converts the integer to a float value.
- Example: `3 * 4.5` is the same as `3.0 * 4.5`
- This is called type conversion.

int(...)

- Sometimes, it is desirable to obtain the integer part of a fractional number.
- You can use the int(value) function to return the integer part of a float value. For example:


```
>>> value = 5.6
>>> int(value)
5
>>>
```

- Note that the fractional part of the number is truncated, not rounded up.

round(...)

- You can also use the round function to round a number to the nearest whole value. For example:

Python

```
>>> value = 5.6 #Odd
>>> round(value)
6
>>> round(5.5)
6
>>> round(5.3)
5
>>> round(-3.5)
-4
>>> round(-3.4)
-3
```

Python

```
>>> value = 6.6 #Even
>>> round(value)
7
>>> round(6.5)
6
>>> round(6.3)
6
>>> round(-6.5)
-6
>>> round(-6.6)
-7
```

- If the number you are rounding is **odd** and followed by **decimal >= 5**, Python rounds the number up.
- If the number you are rounding is **even** and followed by **decimal > 5**, Python rounds the number up.
- Otherwise, Python rounds the number down.

Note

- The functions int and round do not change the variable being converted.
- For example, value is not changed after invoking the function in the following code:


```
>>> value = 5.6
>>> round(value)
6
>>> value
5.6
>>>
```

Note

- If you would like to change the variable being converted by the functions int and round, reset the variable after invoking the function.
- For example, value is changed after invoking the function in the following code:


```
>>> value = 5.6
>>> value = int(value)
>>> value
5
>>>
```

Int(...) vs. eval(...)

- The int function can also be used to convert an integer string into an integer.
  - For example, int("34") returns **34**.
- So you can use the eval or int function to convert a string into an integer. Which one is better?
- The int function performs a simple conversion. It does not work for a non-integer string.
  - For example, int("3.4") **will cause an error**.
- The eval function does more than a simple conversion. It can be used to evaluate an expression.
  - For example, eval("3 + 4") returns **7**.

Int(...) vs. eval(...)

- However, there is a subtle "gotcha" for using the eval function.
  - The definition of gotcha is "a misfeature of a system, especially a programming language or environment, that tends to breed bugs or mistakes because it is both enticingly easy to invoke and completely unexpected and/or unreasonable in its outcome." (Hyper Dictionary)
- The eval function will produce an error for a numeric string that contains leading zeros. In contrast, the int function works fine for this case.
  - For example, eval("003") causes an **error**, but int("003") returns **3**.

str(...)

- You can use the str(value) function to convert the numeric value to a string. For example:


```
>>> value = 5.6
>>> str(value)
'5.6'
>>> value
5.6
>>> value = str(value)
>>> value
'5.6'
```

- Note: The functions str does not change the variable being converted.

Keeping Two Digits After Decimal Points
Program 7

Write a program to get a purchase amount from the user. Then your program should calculate and display the sales tax (6%) with two digits after the decimal point.

```
Enter purchase amount: 197.55   <Enter>
Sales tax is 11.85
```

- Remember:
  - Phase 1: Problem-solving
  - Phase 2: Implementation

Keeping Two Digits After Decimal Points
Phase 1: Problem-solving

- **Step 1:** If you are given a purchase amount, how do you then calculate the sales tax (6%)?
- Example:
  - Given 120, how do we calculate the sales tax (6%) = 7.2?
  - First, we know that 6% = 6 / 100 = 0.06
  - So, we can use the following formula:
    - sales tax = purchase amount x 0.06
  - If we applied the formula, we can get the result (6% of 120 = 7.2).
    - sales tax = 120 x 0.06 = **7.2**

Keeping Two Digits After Decimal Points
Phase 1: Problem-solving

- **Step 2:** If you are given a number, how do you then get the number with two digits after the decimal point?
- Example:
  - Given 123.456, how do we get **123.45** ?
  - Simply, we can multiply the number by 100, then convert the result to integer for removing the decimal points, and finally, divide the integer result by 100 to get the decimal point back with two digits.
    - 123.456 × 100 = **12345.6**
    - Convert 12345.6 to integer → **12345**
    - 12345 / 100 = **123.45**

Keeping Two Digits After Decimal Points
Phase 1: Problem-solving

- Design your algorithm:
  1. Get a purchase amount from the user.
     - Use input function
  2. Compute the sales tax.
     - sales tax = purchase amount x 0.06
  3. Display the result.
     - result = sales tax x 100
     - result = convert result to integer
     - result = result / 100

Keeping Two Digits After Decimal Points

Phase 2: Implementation

LISTING 2.6 SalesTax.py

```
1  # Prompt the user for input
2  purchaseAmount = eval(input("Enter purchase amount: "))
3
4  # Compute sales tax
5  tax = purchaseAmount * 0.06
6
7  # Display tax amount with two digits after decimal point
8  print("Sales tax is", int(tax * 100) / 100.0)
```

Run

```
Enter purchase amount: 197.55  <Enter>
Sales tax is 11.85
```

```
Enter purchase amount: 120  <Enter>
Sales tax is 7.19
```

Keeping Two Digits After Decimal Points
Trace The Program Execution

```
Enter purchase amount: 197.55  <Enter>
Sales tax is 11.85
```

line#	purchaseAmount	tax	output
2	197.55		
5		11.853	
8			11.85

LISTING 2.6 SalesTax.py

```
1  # Prompt the user for input
2  purchaseAmount = eval(input("Enter purchase amount: "))
3
4  # Compute sales tax
5  tax = purchaseAmount * 0.06
6
7  # Display tax amount with two digits after decimal point
8  print("Sales tax is", int(tax * 100) / 100.0)
```

Keeping Two Digits After Decimal Points Discussion

LISTING 2.6 SalesTax.py

```
1  # Prompt the user for input
2  purchaseAmount = eval(input("Enter purchase amount: "))
3
4  # Compute sales tax
5  tax = purchaseAmount * 0.06
6
7  # Display tax amount with two digits after decimal point
8  print("Sales tax is", int(tax * 100) / 100.0)
```

- The value of the variable purchaseAmount is 197.55 (line 2).
- The sales tax is 6% of the purchase, so the tax is evaluated as 11.853 (line 5).
- Note that:
  - tax * 100 is 1185.3
  - int(tax * 100) is 1185
  - int(tax * 100) / 100.0 is 11.85

Check Point #11

Does the int(value) function change the variable value?

- Answer: No, it does not. It return a new value.

Check Point #12

Are the following statements correct? If so, show their printout.

```
1  value = 4.6
2  print(int(value))
3  print(round(value))
4  print(eval("4 * 5 + 2"))
5  print(int("04"))
6  print(int("4.5"))
7  print(eval("04"))
```

1 → ✓
2 → ✓ (4)
3 → ✓ (5)
4 → ✓ (22)
5 → ✓ (4)
6 → ✗
7 → ✗

2.12. Case Study: Displaying the Current Time

- Problem 8: Displaying Current Time

Displaying Current Time
Program 8

Write a program that displays current time in Greenwich Mean Time (GMT) in the format hour:minute:second such as 1:45:19.

```
Current time is 17:31:8 GMT
```

Remember:

- Phase 1: Problem-solving
- Phase 2: Implementation

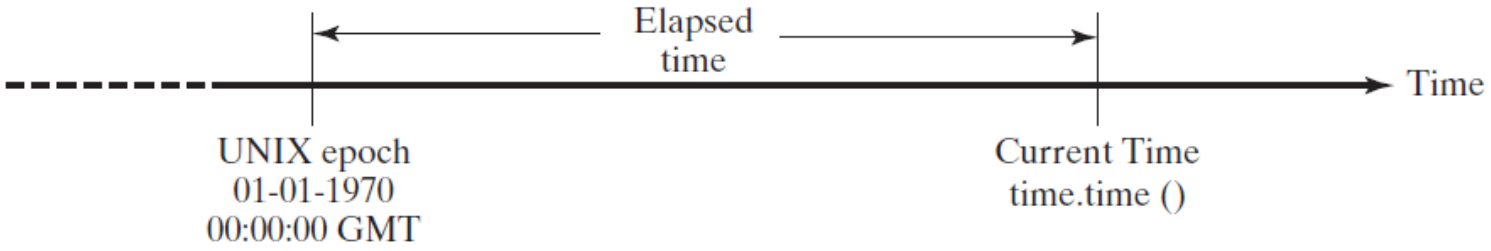
Displaying Current Time
Phase 1: Problem-solving

- Remember how you print to the screen?
  - You use the print function.
  - The Python interpreter has a number of functions and types built into it that are always available such as the print function.
  - There are other functions that Python provide, but you have to import their module (code library) first to can use it.
  - This is done by using import keyword.
- Python provides time() function in the time module to obtain the current system time.
  - This function returns the current time, in milliseconds, in milliseconds since midnight, January 1, 1970 GMT


```
>>> import time
>>> time.time()
1561794760.816502
```

Displaying Current Time
Phase 1: Problem-solving

- The time() function in the time module returns the current time in seconds with millisecond precision elapsed since the time 00:00:00 on January 1, 1970 GMT.
- Why this specific date? This time is known as the UNIX epoch. The epoch is the point when time starts. 1970 was the year when the UNIX operating system was formally introduced.
  - Important? Not really. Just a neat fact!
- For example, time.time() returns 1285543663.205, which means 1285543663 seconds and 205 milliseconds.



Displaying Current Time
Phase 1: Problem-solving

- So this function time.time() returns the number of milliseconds since 1970.
- That is a lot of milliseconds.
- It is 2020 … so 50 years since 1970.
- $50\ years \times \frac{365\ days}{1\ year} \times \frac{24\ hours}{1\ day} \times \frac{3600\ seconds}{1\ hour} \times \frac{1000\ ms}{1\ second}$
- Now take a calculator …
  - That comes to 1,576,800,000,000 milliseconds
- The point: this function returns a huge number.
  - So how can we calculate the time from this number?
- You can use this function to obtain the current time, and then compute the current second, minute, and hour as follows.

Displaying Current Time
Phase 1: Problem-solving

1. Obtain the current time (since midnight, January 1, 1970) by invoking time.time() .
   - For example: **1203183068.328**.
2. Obtain the total seconds (totalSeconds) using the int function.
   - int(**1203183068.328**) = **1203183068**.
3. Compute the current second from totalSeconds % 60.
   - **1203183068** seconds % **60** = **8**, which is the current second.
4. Obtain the total minutes (totalMinutes) from totalSeconds // 60.
   - 1203183068 seconds // 60 = 20053051 minutes.

Displaying Current Time
Phase 1: Problem-solving

5. Compute the current minute from totalMinutes % 60.
   - 20053051 minutes % 60 = 31, which is the current minute.
6. Obtain the total hours (totalHours) from totalMinutes // 60.
   - 20053051 minutes // 60 = 334217 hours.
7. Compute the current hour from totalHours % 24.
   - 334217 hours % 24 = 17, which is the current hour.

- **The final time:**
  17:31:8 GMT or 5:31 PM and 8 seconds

Displaying Current Time
Phase 2: Implementation

LISTING 2.7 ShowCurrentTime.py

```
1  import time
2
3  currentTime = time.time() # Get current time
4
5  # Obtain the total seconds since midnight, Jan 1, 1970
6  totalSeconds = int(currentTime)
7
8  # Get the current second
9  currentSecond = totalSeconds % 60
10
11 # Obtain the total minutes
12 totalMinutes = totalSeconds // 60
13
14 # Compute the current minute in the hour
15 currentMinute = totalMinutes % 60
16
17 # Obtain the total hours
18 totalHours = totalMinutes // 60
19
20 # Compute the current hour
21 currentHour = totalHours % 24
22
23 # Display results
24 print("Current time is " + str(currentHour) + ":"
25     + str(currentMinute) + ":" + str(currentSecond) + " GMT")
```

Displaying Current Time
Trace The Program Execution

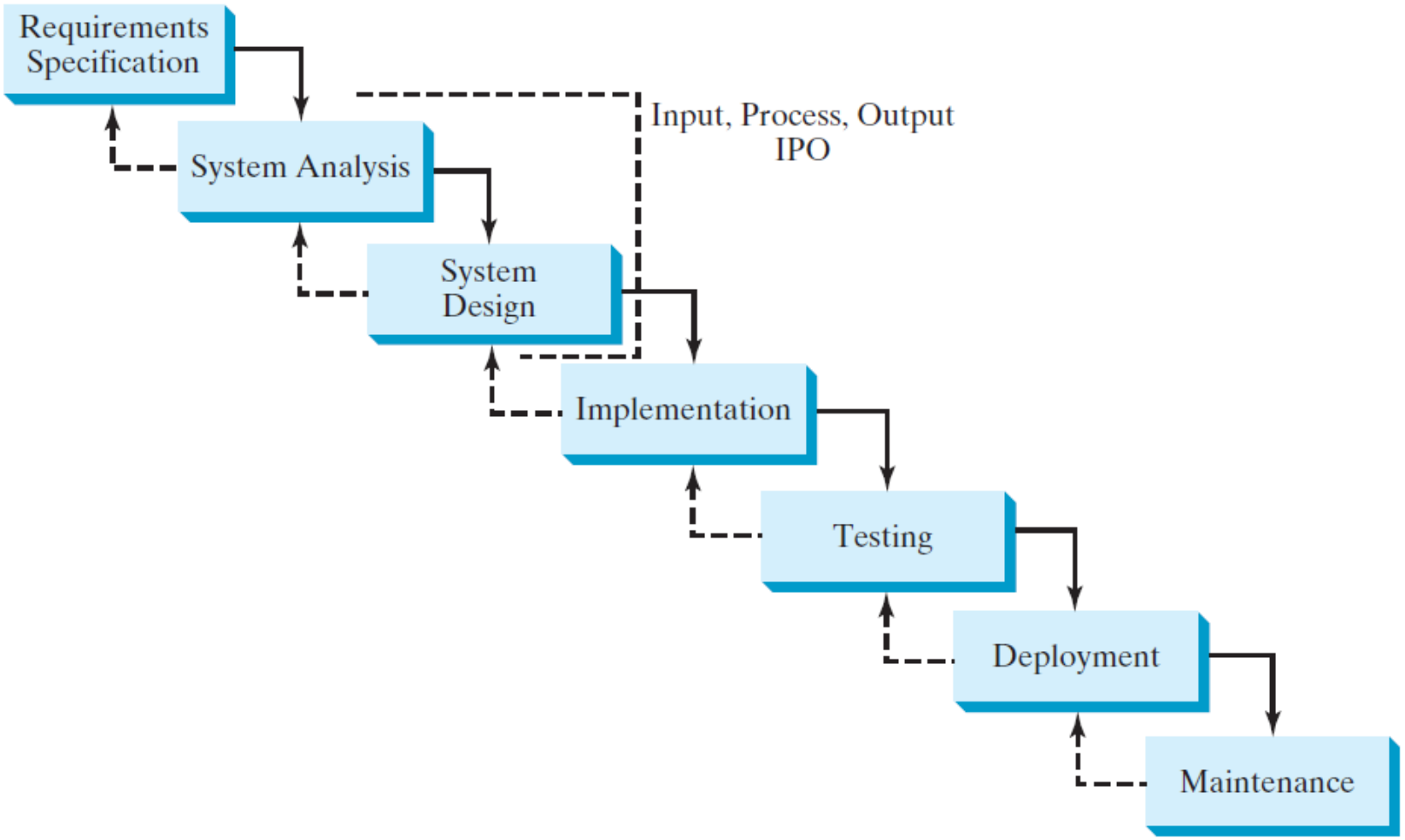
```
Current time is 17:31:8 GMT
```

variables \ line#	3	6	9	12	15	18	21
currentTime	1203183068.328						
totalSeconds		1203183068					
currentSecond			8				
totalMinutes				20053051			
currentMinute					31		
totalHours						334217	
currentHour							17

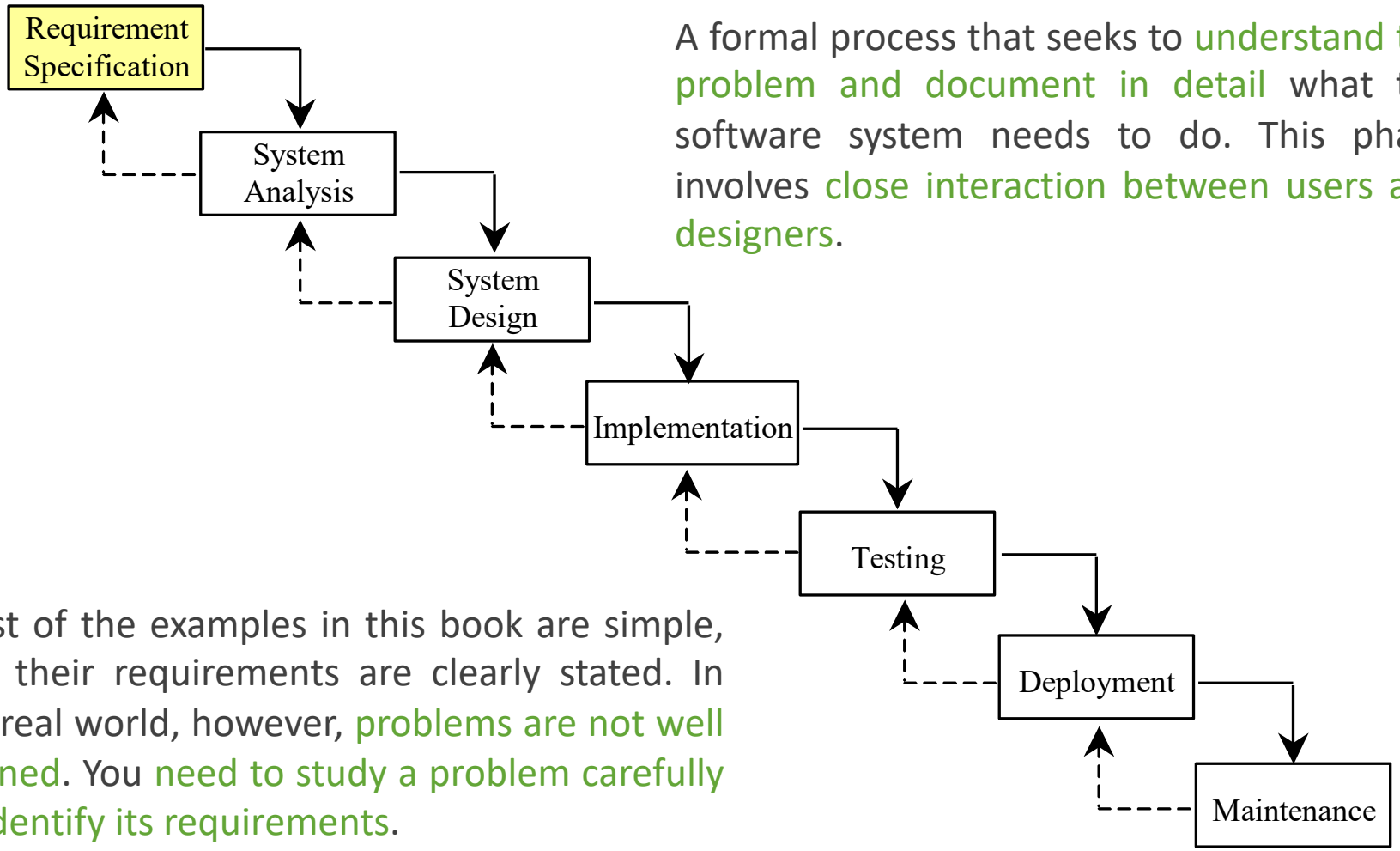
2.13. Software Development Process

- Problem 9: Computing Loan Payments

Software Development Process



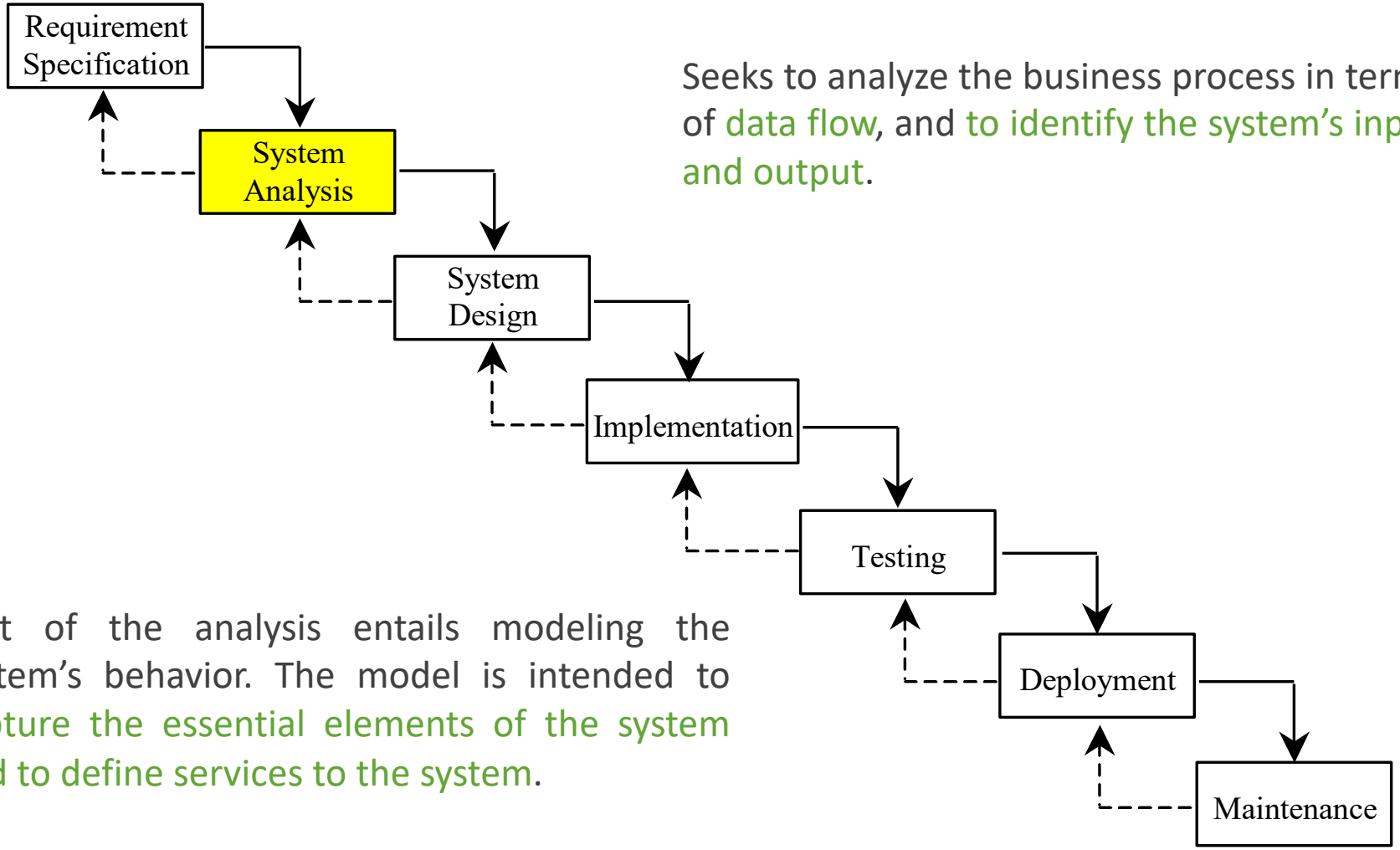
Requirement Specification



A formal process that seeks to understand the problem and document in detail what the software system needs to do. This phase involves close interaction between users and designers.

Most of the examples in this book are simple, and their requirements are clearly stated. In the real world, however, problems are not well defined. You need to study a problem carefully to identify its requirements.

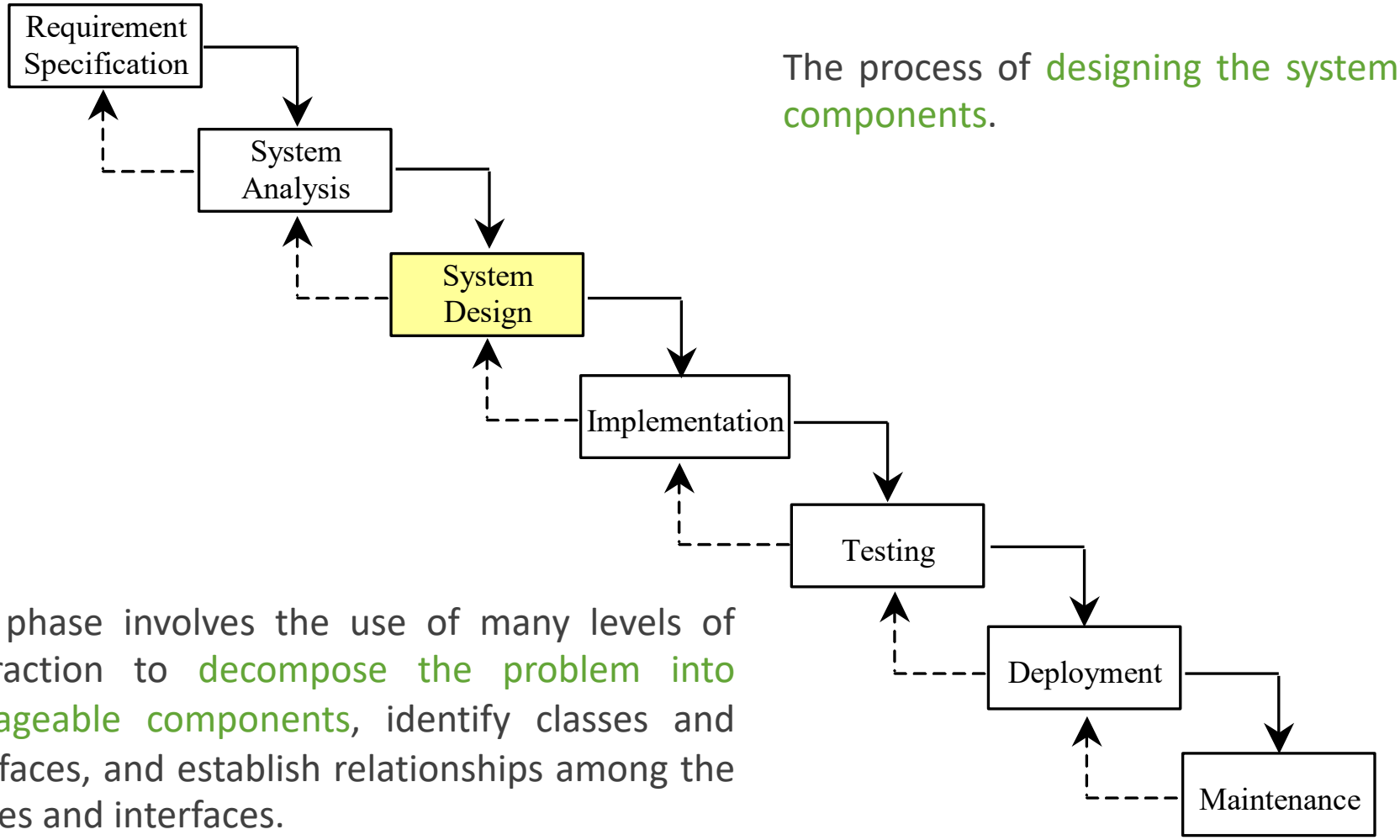
System Analysis



Seeks to analyze the business process in terms of data flow, and to identify the system's input and output.

Part of the analysis entails modeling the system's behavior. The model is intended to capture the essential elements of the system and to define services to the system.

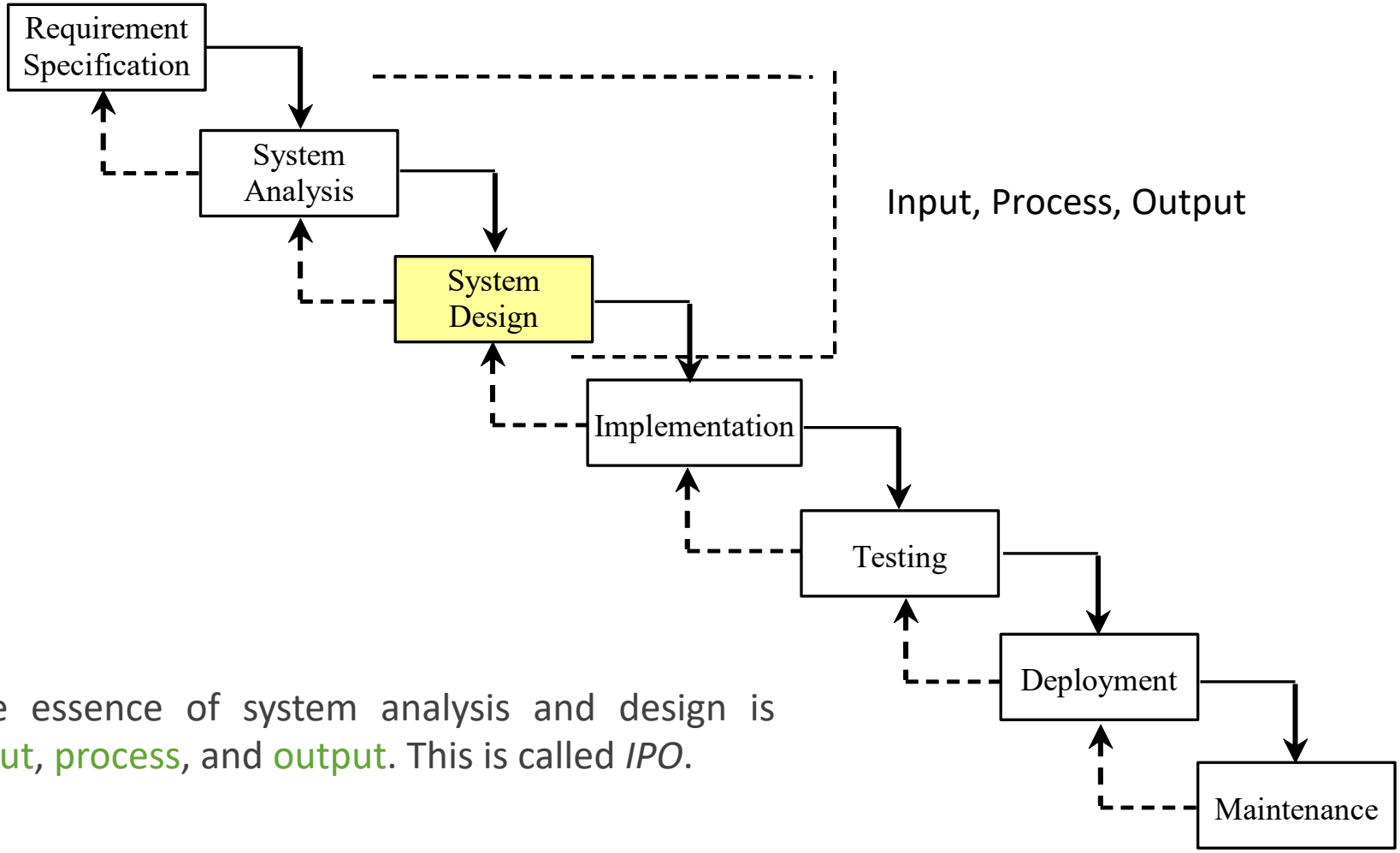
System Design



The process of designing the system's components.

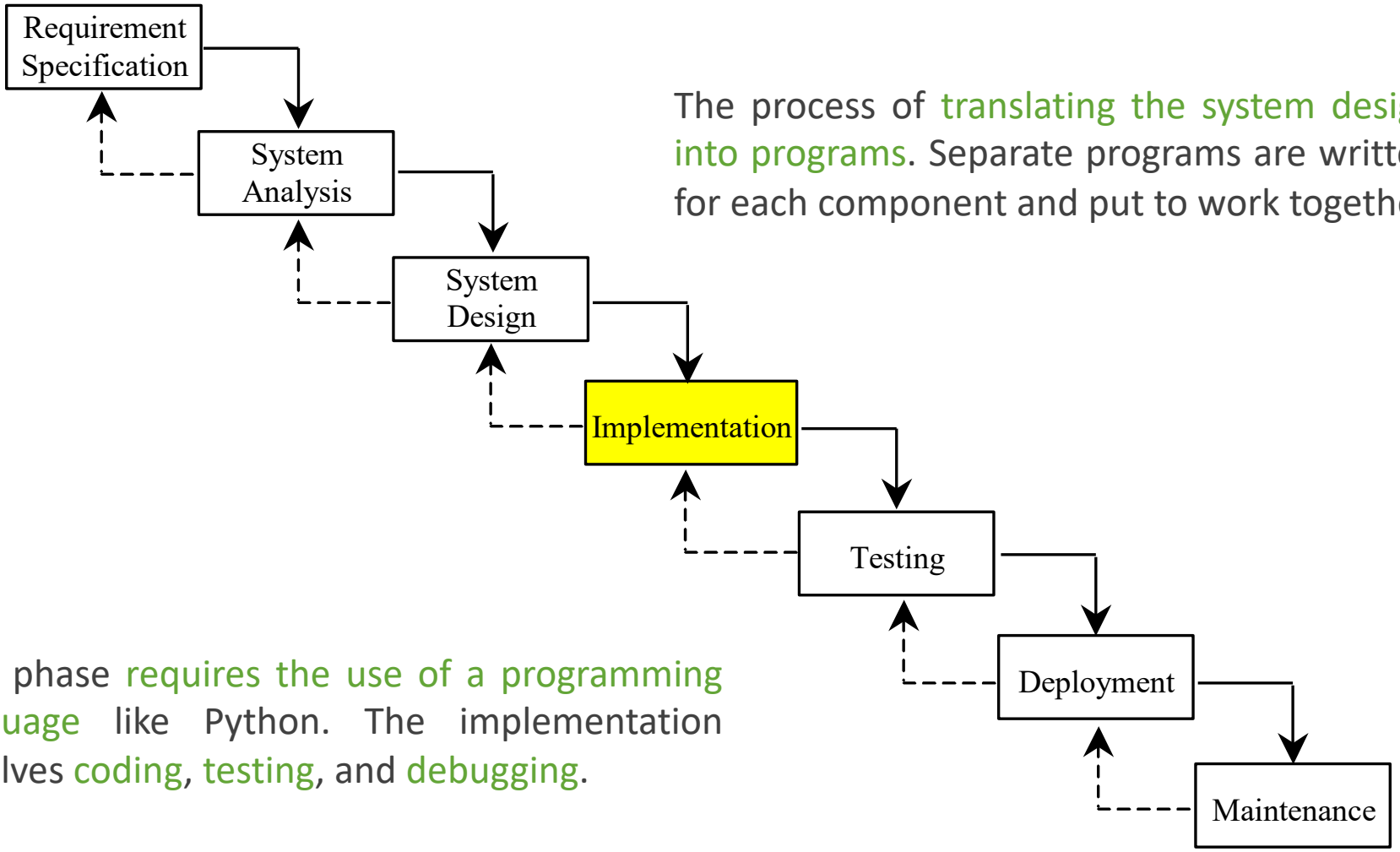
This phase involves the use of many levels of abstraction to decompose the problem into manageable components, identify classes and interfaces, and establish relationships among the classes and interfaces.

IPO



The essence of system analysis and design is input, process, and output. This is called *IPO*.

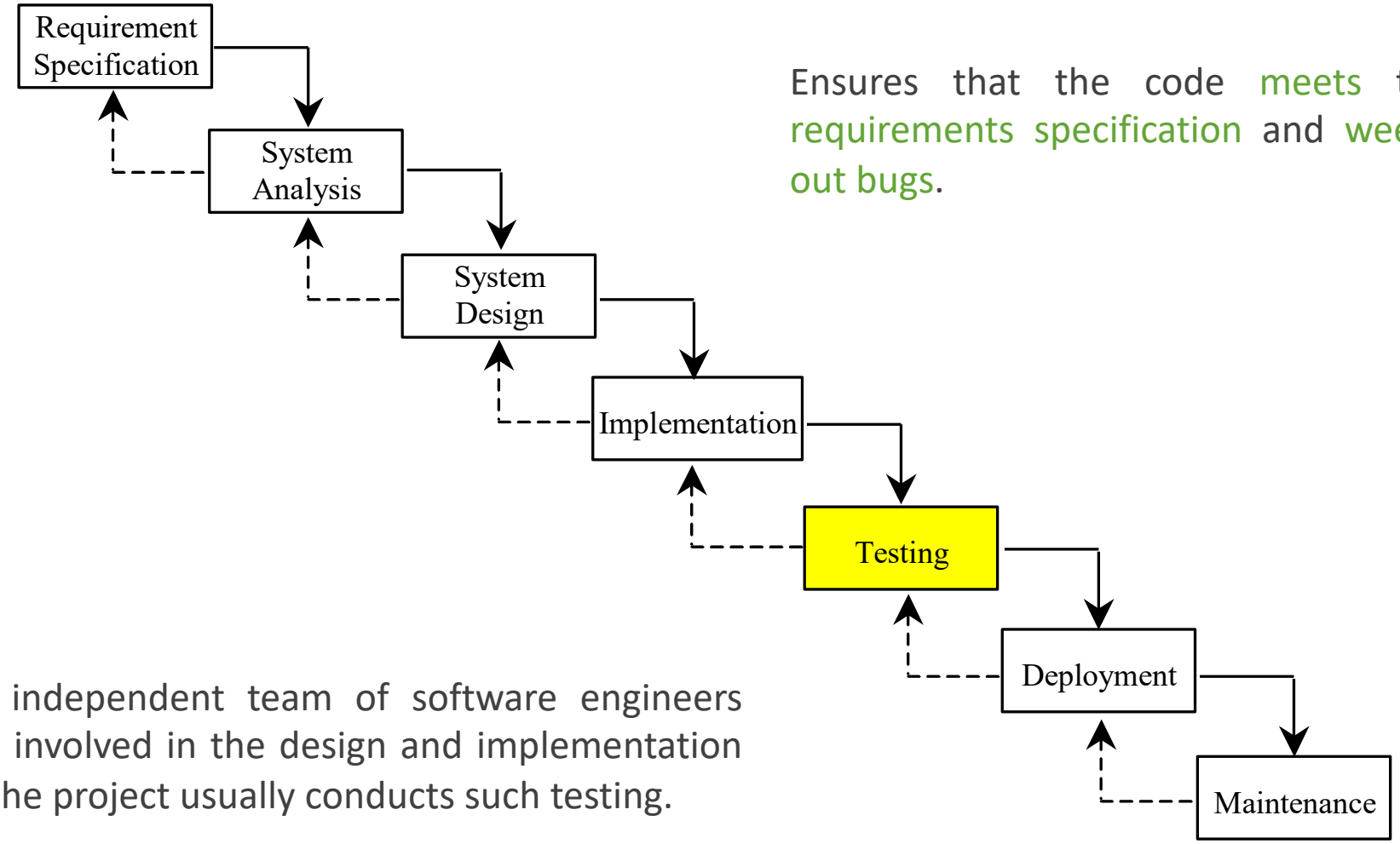
Implementation



The process of translating the system design into programs. Separate programs are written for each component and put to work together.

This phase requires the use of a programming language like Python. The implementation involves coding, testing, and debugging.

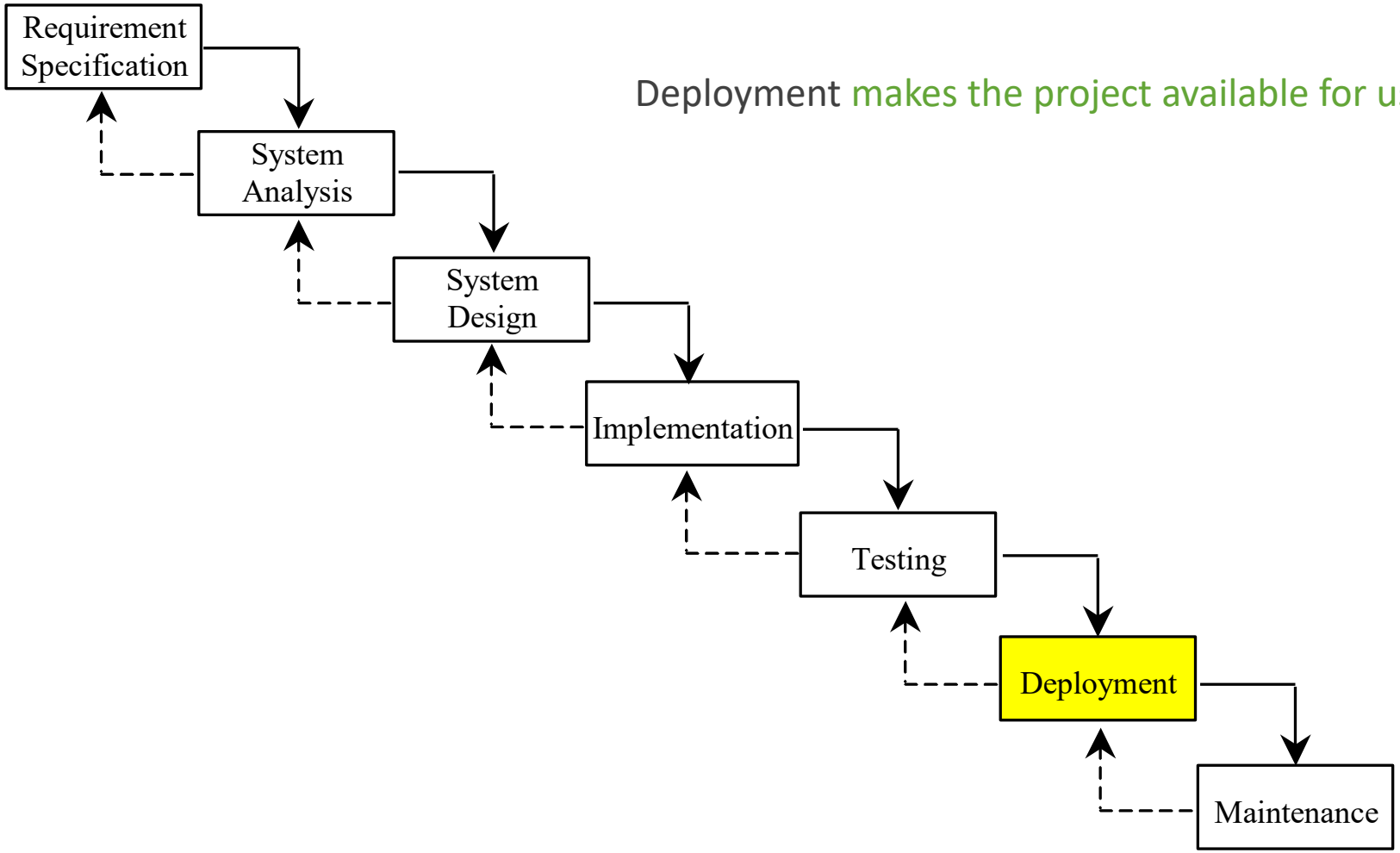
Testing



Ensures that the code meets the requirements specification and weeds out bugs.

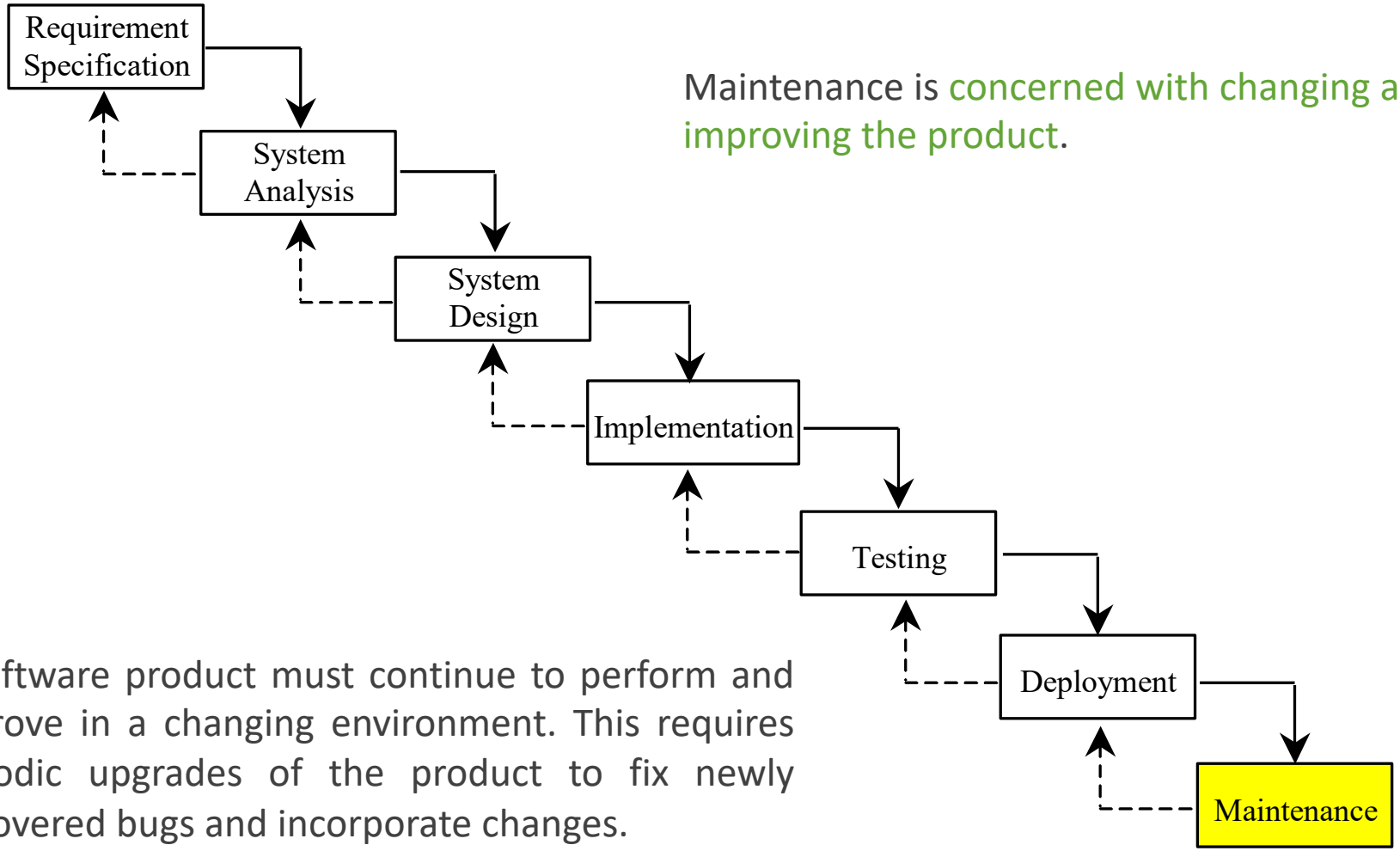
An independent team of software engineers not involved in the design and implementation of the project usually conducts such testing.

Deployment



Deployment makes the project available for use.

Maintenance



Maintenance is concerned with changing and improving the product.

A software product must continue to perform and improve in a changing environment. This requires periodic upgrades of the product to fix newly discovered bugs and incorporate changes.

Example

- To see the software development process in action, we will now create a program that computes loan payments. The loan can be a car loan, a student loan, or a home mortgage loan.
- For an introductory programming course, we focus on requirements specification, analysis, design, implementation, and testing.

Computing Loan Payments
Program 9

Write a program that lets the user enter the annual interest rate, number of years, and loan amount, and computes monthly payment and total payment.

$$monthlyPayment = \frac{loanAmount \times monthlyInterestRate}{1 - \frac{1}{(1 + monthlyInterestRate)^{numberOfYears \times 12}}}$$

$$totalPayment = monthlyPayment \times numberOfYears \times 12$$

```
Enter annual interest rate, e.g., 8.25: 5.75   <Enter>
Enter number of years as an integer, e.g., 5: 15   <Enter>
Enter loan amount, e.g., 120000.95: 250000   <Enter>
The monthly payment is  2076.02
The total payment is  373684.53
```

Computing Loan Payments
Stage 1: Requirements Specification

- The program must satisfy the following requirements:
  - It must let the user enter the annual interest rate, the loan amount, and the number of years for which payments will be made.
  - It must compute and display the monthly payment and total payment amounts.

Computing Loan Payments
Stage 2: System Analysis

- The output is the monthly payment and total payment, which can be obtained using the following formula:

$$monthlyPayment = \frac{loanAmount \times monthlyInterestRate}{1 - \frac{1}{(1 + monthlyInterestRate)^{numberOfYears \times 12}}}$$

$$totalPayment = monthlyPayment \times numberOfYears \times 12$$

- So, the input needed for the program is the annual interest rate, the length of the loan in years, and the loan amount.

Computing Loan Payments
Stage 2: System Analysis

- Note 1:
  - The requirements specification says that the user must enter the interest rate, the loan amount, and the number of years for which payments will be made.
  - During analysis, however, it is possible that you may discover that input is not sufficient or that some values are unnecessary for the output.
  - If this happens, you can go back to modify the requirements specification.

Computing Loan Payments
Stage 2: System Analysis

- Note 2:
  - In the real world, you will work with customers from all walks of life.
  - You may develop software for chemists, physicists, engineers, economists, and psychologists and of course, you will not have (or need) the complete knowledge of all these fields.
  - Therefore, you don't have to know how the mathematical formulas are derived.
  - Nonetheless, given the annual interest rate, number of years, and loan amount, you can use the given formula to compute the monthly payment.
  - You will, however, need to communicate with the customers and understand how the mathematical model works for the system.

Computing Loan Payments
Stage 3: System Design

- During system design, you identify the steps in the program:
  1. Prompt the user to enter the annual interest rate, number of years, and loan amount.
  2. Compute monthly interest rate:
     - The input for the annual interest rate is a number in percent format, such as 4.5%. The program needs to convert it into a decimal by dividing it by 100.
     - To obtain the monthly interest rate from the annual interest rate, divide it by 12, since a year has 12 months.
     - So to obtain the monthly interest rate in decimal format, you need to divide the annual interest rate in percentage by 1200.
     - For example, if the annual interest rate is 4.5%, then the monthly interest rate is 4.5/1200 = 0.00375. It is equivalent to $(\frac{4.5}{100} \times \frac{1}{12} = \frac{4.5}{1200} = 0.00375)$

Computing Loan Payments
Stage 3: System Design

- During system design, you identify the steps in the program:

3. Compute the monthly payment using the formula given in Stage 2.

$$monthlyPayment = \frac{loanAmount \times monthlyInterestRate}{1 - \frac{1}{(1 + monthlyInterestRate)^{numberOfYears \times 12}}}$$

4. Compute the total payment, which is the monthly payment multiplied by 12 and multiplied by the number of years.

$$totalPayment = monthlyPayment \times numberOfYears \times 12$$

5. Display the monthly payment and total payment.

Computing Loan Payments
Stage 4: Implementation

- Implementation is also known as coding (writing the code).
- In the formula, you have to compute

$$(1 + monthlyInterestRate)^{numberOfYears \times 12}$$

- You can use the exponentiation operator to write it as:

```
(1 + monthlyInterestRate) ** (numberOfYears * 12)
```

Computing Loan Payments
Stage 4: Implementation

LISTING 2.8 ComputeLoan.py

Run

```
1  # Enter yearly interest rate
2  annualInterestRate = eval(input(
3      "Enter annual interest rate, e.g., 8.25: "))
4  monthlyInterestRate = annualInterestRate / 1200
5
6  # Enter number of years
7  numberOfYears = eval(input(
8      "Enter number of years as an integer, e.g., 5: "))
9
10 # Enter loan amount
11 loanAmount = eval(input("Enter loan amount, e.g., 120000.95: "))
12
13 # Calculate payment
14 monthlyPayment = loanAmount * monthlyInterestRate / (1
15    - 1 / (1 + monthlyInterestRate) ** (numberOfYears * 12))
16 totalPayment = monthlyPayment * numberOfYears * 12
17
18 # Display results
19 print("The monthly payment is ", int(monthlyPayment * 100) / 100)
20 print("The total payment is ", int(totalPayment * 100) / 100)
```

Computing Loan Payments
Trace The Program Execution

```
Enter annual interest rate, e.g., 8.25: 5.75   <Enter>
Enter number of years as an integer, e.g., 5: 15   <Enter>
Enter loan amount, e.g., 120000.95: 250000   <Enter>
The monthly payment is  2076.02
The total payment is  373684.53
```

variables \ line#	2	4	7	11	14	16
annualInterestRate	5.75					
monthlyInterestRate		0.0047916666666				
numberOfYears			15			
loanAmount				250000		
monthlyPayment					2076.0252175	
totalPayment						373684.539

Computing Loan Payments
Discussion

- Line 2 reads the annual interest rate, which is converted into the monthly interest rate in line 4.
- The formula for computing the monthly payment is translated into Python code in lines 14–15.
- The variable monthlyPayment is 2076.0252175 (line 14).
- Note that:
  - int(monthlyPayment * 100) is **207602**
  - int(monthlyPayment * 100) / 100.0 is **2076.02**
- So, the statement in line 19 displays the monthly payment 2076.02 with two digits after the decimal point.

Computing Loan Payments
Stage 5: Testing

- After the program is implemented, test it with some sample input data and verify whether the output is correct.
- Some of the problems may involve many cases as you will see in later chapters.
- For this type of problems, you need to design test data that cover all cases.

Computing Loan Payments
Stage 5: Testing

- Tip (Incremental Development and Testing)
  - The system design phase in this example identified several steps.
  - It is a good approach to develop and test these steps incrementally by adding them one at a time.
  - This process makes it much easier to pinpoint problems and debug the program.

2.14. Case Study: Computing Distances

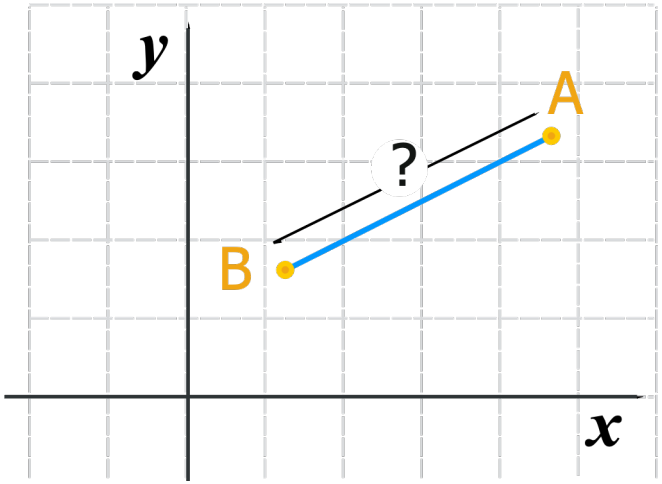
- Problem 10: Computing Distances

Computing Distances
Program 10

Write a program that prompts the user to enter two points, computes their distance, and displays the result.

$$distance((x1, y1), (x2, y2)) = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

```
Enter x1 and y1 for Point 1: 1.5, -3.4  <Enter>
Enter x2 and y2 for Point 2: 4, 5  <Enter>
The distance between the two points is 8.764131445842194
```



Remember:

- Phase 1: Problem-solving
- Phase 2: Implementation

Computing Distances
Phase 1: Problem-solving

- How you can calculate $\sqrt{a}$?
  - You can use a ** 0.5 to compute $\sqrt{a}$

$$a^{0.5} = \sqrt{a}$$

- Also, note that:

$$a^2 = a \times a$$

Computing Distances
Phase 1: Problem-solving

- Design your algorithm:
  1. Get x1, y1 from the user for the first point.
     - Use input function
  2. Get x2, y2 from the user for the second point.
     - Use input function
  3. Compute the distance.
     - distance = ((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2)) ** 0.5
  4. Display the distance.

Computing Distances
Phase 2: Implementation

LISTING 2.9 ComputeDistance.py

```
1  # Enter the first point with two float values
2  x1, y1 = eval(input("Enter x1 and y1 for Point 1: "))
3
4  # Enter the second point with two float values
5  x2, y2 = eval(input("Enter x2 and y2 for Point 2: "))
6
7  # Compute the distance
8  distance = ((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2)) ** 0.5
9
10 print("The distance between the two points is", distance)
```

Run

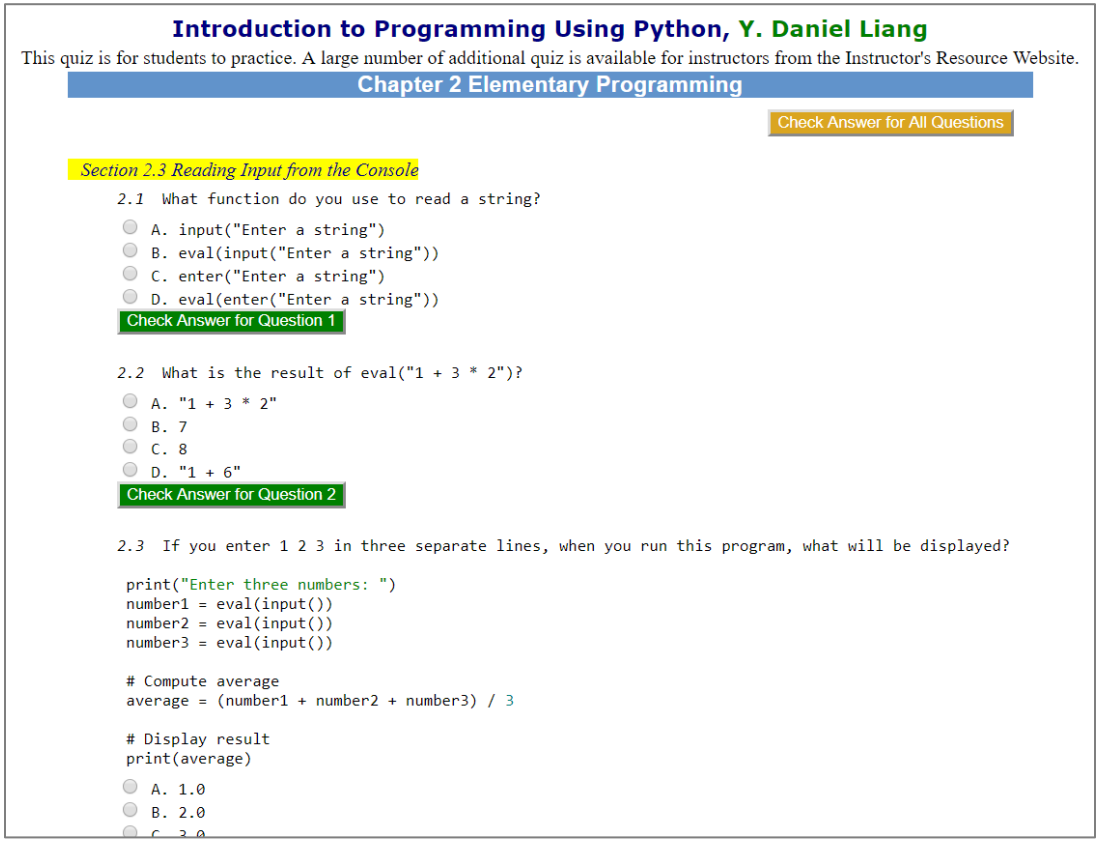
```
Enter x1 and y1 for Point 1: 1.5, -3.4  <Enter>
Enter x2 and y2 for Point 2: 4, 5  <Enter>
The distance between the two points is 8.764131445842194
```

End

- Test Questions
- Programming Exercises

Test Questions

- Do the test questions for this chapter online at https://liveexample-ppe.pearsoncmg.com/selftest/selftestpy?chapter=2



Introduction to Programming Using Python, Y. Daniel Liang

This quiz is for students to practice. A large number of additional quiz is available for instructors from the Instructor's Resource Website.

Chapter 2 Elementary Programming

Check Answer for All Questions

Section 2.3 Reading Input from the Console

2.1 What function do you use to read a string?

- A. input("Enter a string")
- B. eval(input("Enter a string"))
- C. enter("Enter a string")
- D. eval(enter("Enter a string"))

Check Answer for Question 1

2.2 What is the result of eval("1 + 3 * 2")?

- A. "1 + 3 * 2"
- B. 7
- C. 8
- D. "1 + 6"

Check Answer for Question 2

2.3 If you enter 1 2 3 in three separate lines, when you run this program, what will be displayed?

```
print("Enter three numbers: ")
number1 = eval(input())
number2 = eval(input())
number3 = eval(input())

# Compute average
average = (number1 + number2 + number3) / 3

# Display result
print(average)
```

- A. 1.0
- B. 2.0
- C. 3.0

Programming Exercises

- Page 55 – 60:
  - 2.1 – 2.8
  - 2.10
  - 2.12 - 14
  - 2.16 - 2.17
- Lab #3
- Lab #4