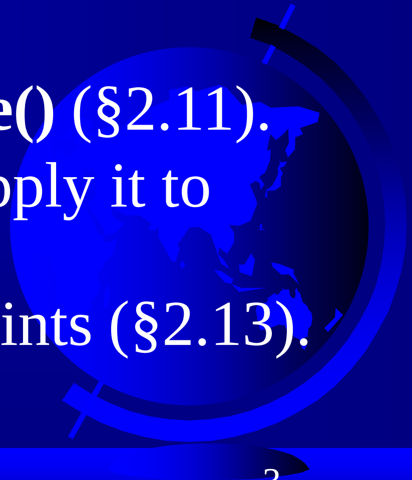
Chapter 2 Elementary Programming

Motivations

Suppose, for example, that you need to take out a student loan. Given the loan amount, loan term, and annual interest rate, can you write a program to compute the monthly payment and total payment? This chapter shows you how to write programs like this. Along the way, you learn the basic steps that go into analyzing a problem, designing a solution, and implementing the solution by creating a program.

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 variables (§2.4).
- To assign data to variables (§2.5).
- To define named constants (§2.6).
- To use the operators **+**, **-**, *****, **/**, **//**, **%**, and ****** (§2.7).
- To write and evaluate numeric expressions (§2.8).
- To use augmented assignment operators to simplify coding (§2.9).
- To perform numeric type conversion and rounding with the **int** and **round** functions (§2.10).
- To obtain the current system time by using **time.time()** (§2.11).
- To describe the software development process and apply it to develop the loan payment program (§2.12).
- To compute and display the distance between two points (§2.13).



Introducing Programming with an Example

Listing 2.1 Computing the Area of a Circle

This program computes the area of the circle.

ComputeArea

Run

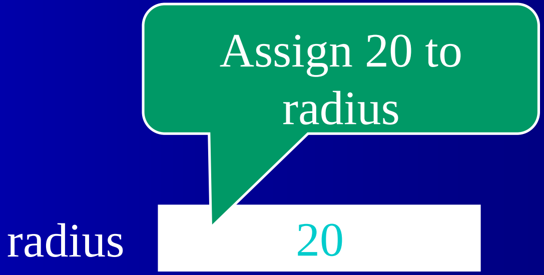
IMPORTANT NOTE:

(1) To enable the buttons, you must download the entire slide file slide.zip and unzip the files into a directory (e.g., c:\slide). (2) You must have installed Python and set python bin directory in the environment path. (3) If you are using Office 2010, check PowerPoint2010.doc located in the same folder with this ppt file.

animation

Trace a Program Execution

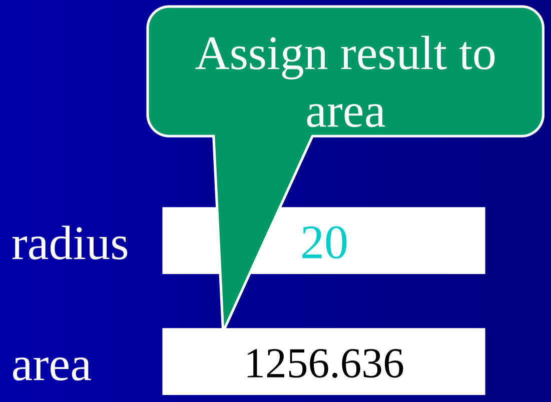
```
# Assign a radius
radius = 20 # radius is now 20
# Compute area
area = radius * radius * 3.14159
# Display results
print("The area for the circle of radius " +
    str(radius) + " is " + str(area))
```



animation

Trace a Program Execution

```
# Assign a radius
radius = 20 # radius is now 20
# Compute area
area = radius * radius * 3.14159
# Display results
print("The area for the circle of radius“,
   radius, " is "area)
```



Assign result to area

animation

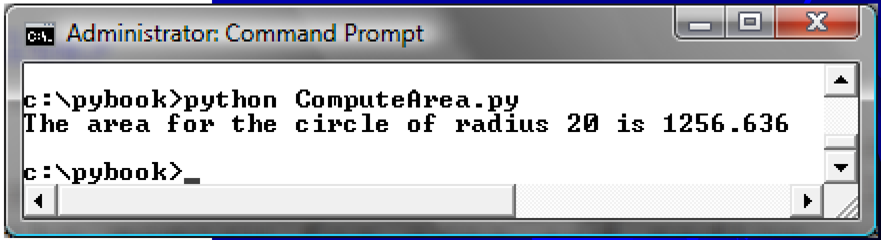
Trace a Program Execution

```
# Assign a radius
radius = 20 # radius is now 20
# Compute area
area = radius * radius * 3.14159
# Display results
print("The area for the circle of radius",
   radius, "is", area)
```

print a message to the console

radius 20

area 1256.636



```
c:\pybook>python ComputeArea.py
The area for the circle of radius 20 is 1256.636

c:\pybook>_
```

Reading Input from the Console

1. Use the input function

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

2. Use the eval function

```
var = eval(stringVariable)

eval("51 + (54 * (3 + 2))") returns 321.
```

ComputeAreaWithConsoleInput

Run

ComputeAverage

Run

Comments in Python

- Anything after a # is ignored by Python
- Why comment?
  - Describe what is going to happen in a sequence of code
  - Document who wrote the code or other ancillary information
  - Turn off a line of code - perhaps temporarily

Identifiers/Variable Names

- An identifier is a sequence of characters that consists of letters, digits, underscores (_), and asterisk (*).
- An identifier must start with a letter or an underscore. It cannot start with a digit.
- An identifier cannot be a reserved word. (See Appendix A, "Python Keywords," for a list of reserved words.) Reserved words have special meanings in Python, which we will later.
- An identifier can be of any length.

Python Variable Name Rules

- Must start with a letter or underscore _
- Must consist of letters and numbers and underscores
- Case Sensitive
- Good: spam eggs spam23 _speed
- Bad: 23spam #sign var.12
- Different: spam Spam SPAM

Reserved Words

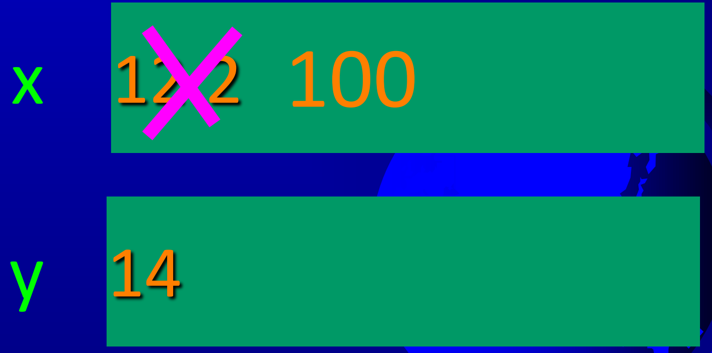
- You can not use reserved words as variable names / identifiers

and del for is raise
assert elif from lambda return
break else global not try
class except if or while
continue exec import pass yield
def finally in print

Variables

- A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable “name”
- Programmers get to choose the names of the variables
- You can change the contents of a variable in a later statement

```
x = 12.2
y = 14
x = 100
```



Variables

```
# Compute the first area
radius = 1.0
area = radius * radius * 3.14159
print("The area is ", area,
    " for  radius ", radius)
# Compute the second area
radius = 2.0
area = radius * radius * 3.14159
print("The area is ", area,
    " for radius ", radius)
```

Expression

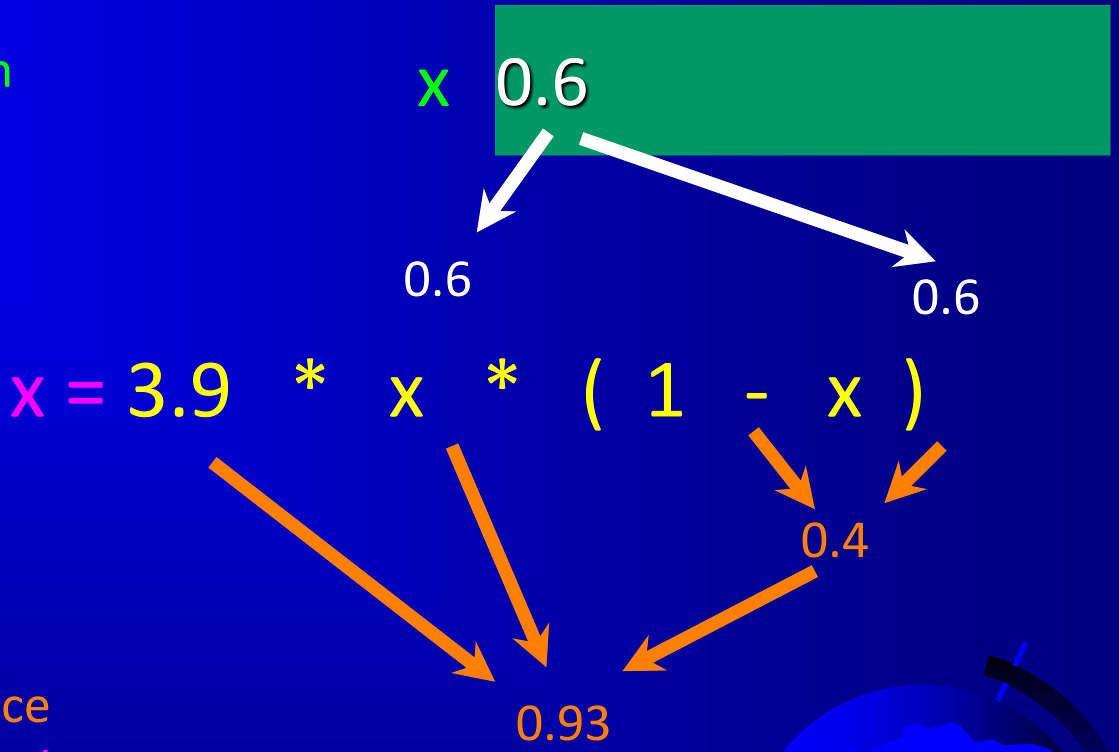
```
x = 1             # Assign 1 to variable x
radius = 1.0      # Assign 1.0 to variable radius

# Assign the value of the expression to x
x = 5 * (3 / 2) + 3 * 2

x = y + 1     # Assign the addition of y and 1 to x
area = radius * radius * 3.14159 # Compute area
```

Assignment Statements

- We assign a value to a variable using the assignment statement (=)
- An assignment statement consists of an expression on the right hand side and a variable to store the result

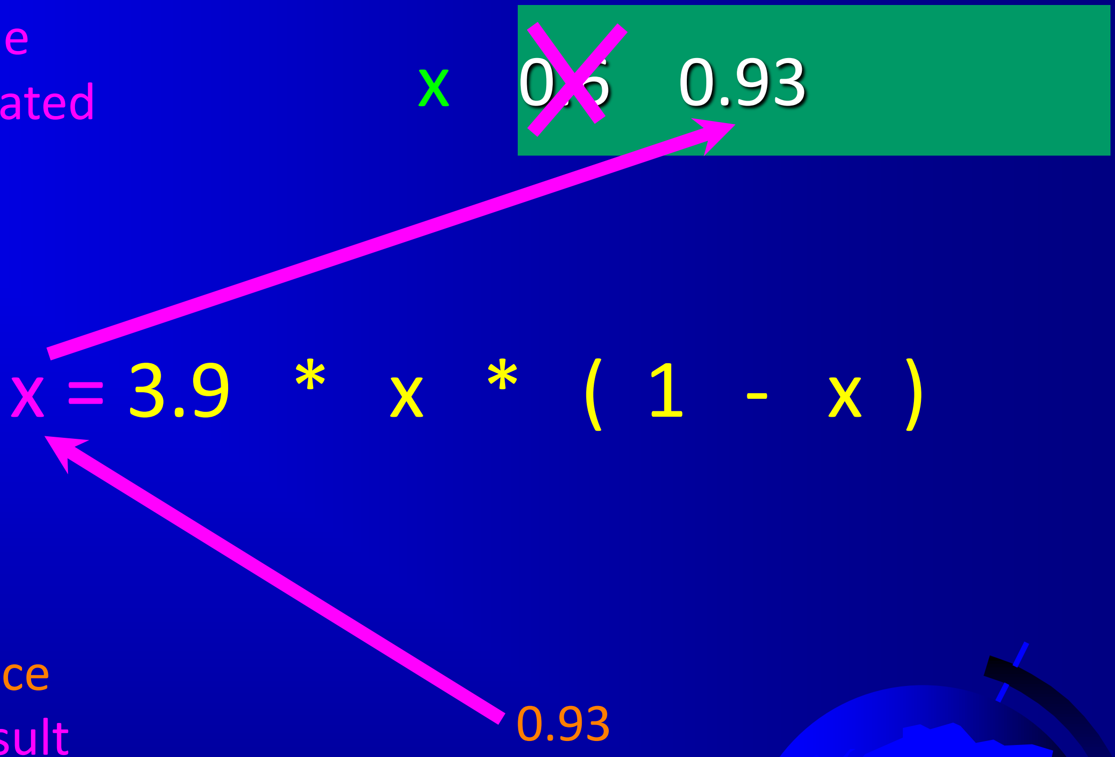
```
x = 3.9  *  x  *  ( 1  -  x )
```

A variable is a memory location used to store a value (0.6).



Right side is an expression. Once expression is evaluated, the result is placed in (assigned to) x.

A variable is a memory location used to store a value. The value stored in a variable can be updated by replacing the old value (0.6) with a new value (0.93).



x ~~0.6~~ 0.93

x = 3.9 * x * (1 - x)

0.93

Right side is an expression. Once expression is evaluated, the result is placed in (assigned to) the variable on the left side (i.e. x).

Assignment Statements

```
x = 1       # Assign 1 to x

x = x + 1

i = j = k = 1
```

Simultaneous Assignment

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

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

ComputeAverageWithSimultaneousAssignment

Run

Named Constants

The value of a variable may change during the execution of a program, but a *named constant* or simply *constant* represents 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.

Numerical Data Types

- integer: e.g., 3, 4
- float: e.g., 3.0, 4.0

Several Types of Numbers

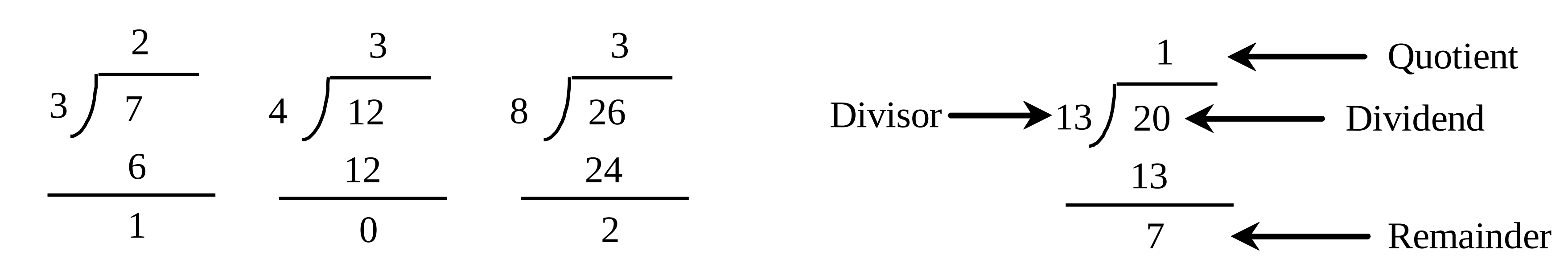
- Numbers have two main types
  - Integers are whole numbers: -14, -2, 0, 1, 100, 401233
  - Floating Point Numbers have decimal parts: -2.5 , 0.0, 98.6, 14.0
- There are other number types - they are variations on float and integer

```
>>> xx = 1
>>> type (xx)
<class 'int'>
>>> temp = 98.6
>>> type(temp)
< class 'float'>
>>> type(1)
< class 'int'>
>>> type(1.0)
< class 'float'>
>>>
```

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

The % Operator



2
3) 7
6
1

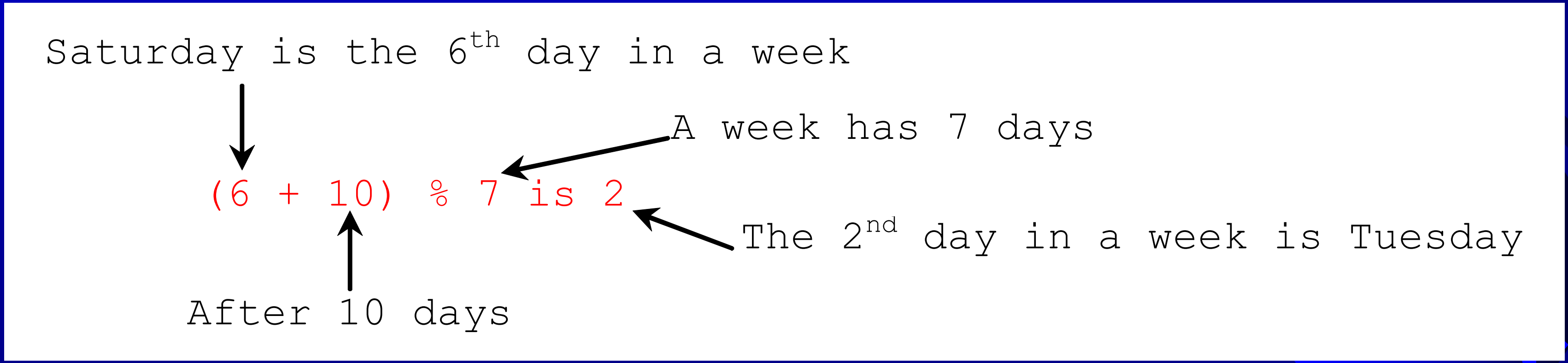
3
4) 12
12
0

3
8) 26
24
2

1 ← Quotient
Divisor → 13) 20 ← Dividend
13
7 ← Remainder

Remainder Operator

Remainder is very useful in programming. For example, an even number % 2 is always 0 and an odd number % 2 is always 1. So you can use this property to determine whether a number is even or odd. Suppose today is Saturday and you and your friends are going to meet in 10 days. What day is in 10 days? You can find that day is Tuesday using the following expression:



Saturday is the 6th day in a week

A week has 7 days

(6 + 10) % 7 is 2

The 2nd day in a week is Tuesday

After 10 days

Problem: Displaying Time

Write a program that obtains hours and minutes from seconds.

DisplayTime

Run

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*.

```
>>>245.0 ** 1000

OverflowError: 'Result too large'
```

Underflow

When a floating-point number is too small (i.e., 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, for 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.

Arithmetic Expressions

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

is translated to

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

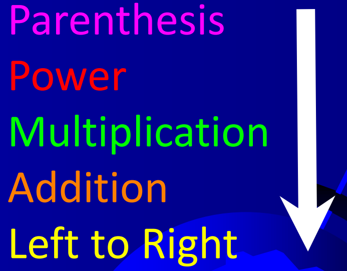
Order of Evaluation

- When we string operators together - Python must know which one to do first
- This is called “operator precedence”
- Which operator “takes precedence” over the others

```
x = 1 + 2 * 3 - 4 / 5 ** 6
```

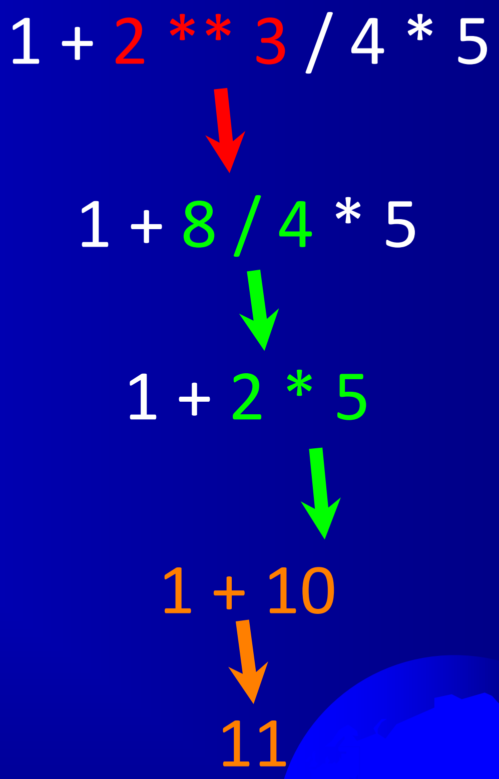
Operator Precedence Rules

- Highest precedence rule to lowest precedence rule
  - Parenthesis are always respected
  - Exponentiation (raise to a power)
  - Multiplication, Division, and Remainder
  - Addition and Subtraction
  - Left to right



Parenthesis
Power
Multiplication
Addition
Left to Right

```
>>> x = 1 + 2 ** 3 / 4 * 5
>>> print(x)
11
>>>
```



1 + 2 ** 3 / 4 * 5

1 + 8 / 4 * 5

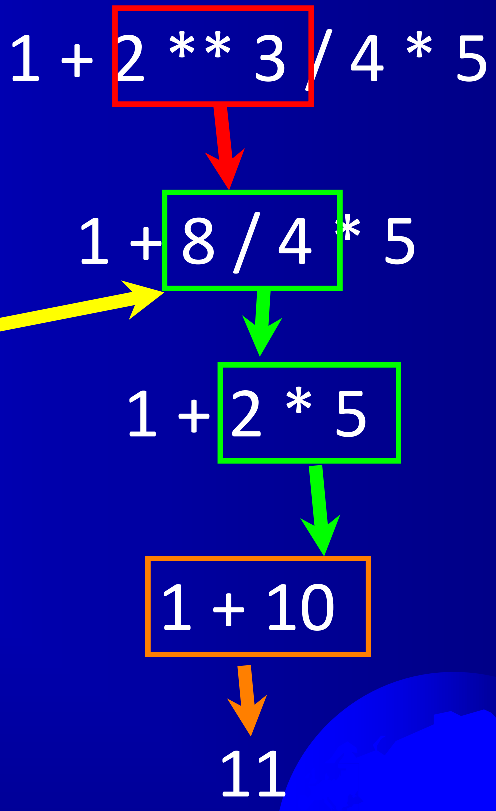
1 + 2 * 5

1 + 10

11

Parenthesis
Power
Multiplication
Addition
Left to Right

```
>>> x = 1 + 2 ** 3 / 4 * 5
>>> print x
11
>>>
```



1 + 2 ** 3 / 4 * 5

1 + 8 / 4 * 5

1 + 2 * 5

1 + 10

11

Note 8/4 goes before 4*5 because of the left-right rule.

Parenthesis
Power
Multiplication
Addition
Left to Right

Operator Precedence

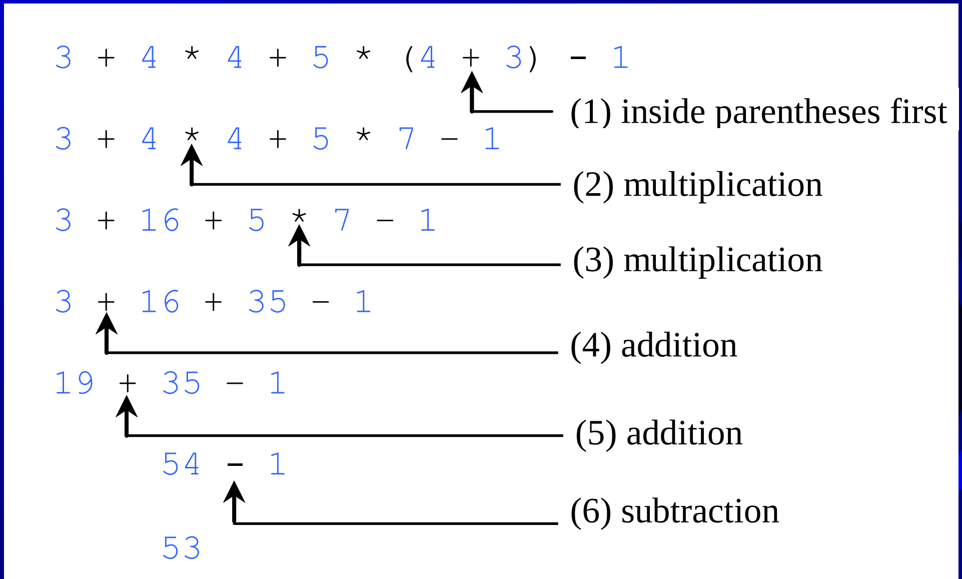
- Remember the rules top to bottom
- When writing code - use parenthesis
- When writing code - keep mathematical expressions simple enough that they are easy to understand
- Break long series of mathematical operations up to make them more clear

Parenthesis
Power
Multiplication
Addition
Left to Right

Exam Question: x = 1 + 2 * 3 - 4 / 5

How to Evaluate an Expression

Though Python has its own way to evaluate an expression behind the scene, the result of a Python expression and its corresponding arithmetic expression are the same. Therefore, you can safely apply the arithmetic rule for evaluating a Python expression.



```
3 + 4 * 4 + 5 * (4 + 3) - 1
                                   (1) inside parentheses first
3 + 4 * 4 + 5 * 7 - 1
                                   (2) multiplication
3 + 16 + 5 * 7 - 1
                                   (3) multiplication
3 + 16 + 35 - 1
                                   (4) addition
19 + 35 - 1
                                   (5) addition
     54 - 1
                                   (6) subtraction
     53
```

Mixing Integer and Floating

- When you perform an operation where one operand is an integer and the other operand is a floating point the result is a floating point
- The integer is converted to a floating point before the operation

```
>>> print (1 + 2 * 3 / 4.0 – 5)
-2.5
>>>
```

Augmented Assignment Operators

Operator	*Example*	*Equivalent*
`+=`	`i += 8`	`i = i + 8`
`-=`	`f -= 8.0`	`f = f - 8.0`
`*=`	`i *= 8`	`i = i * 8`
`/=`	`i /= 8`	`i = i / 8`
`%=`	`i %= 8`	`i = i % 8`

Type Conversion and Rounding

datatype(value)

i.e., int(4.5) => 4
float(4) => 4.0
str(4) => “4”

round(4.6) => 5
round(4.5) => 4

Problem: Keeping Two Digits After Decimal Points

Write a program that displays the sales tax with two digits after the decimal point.

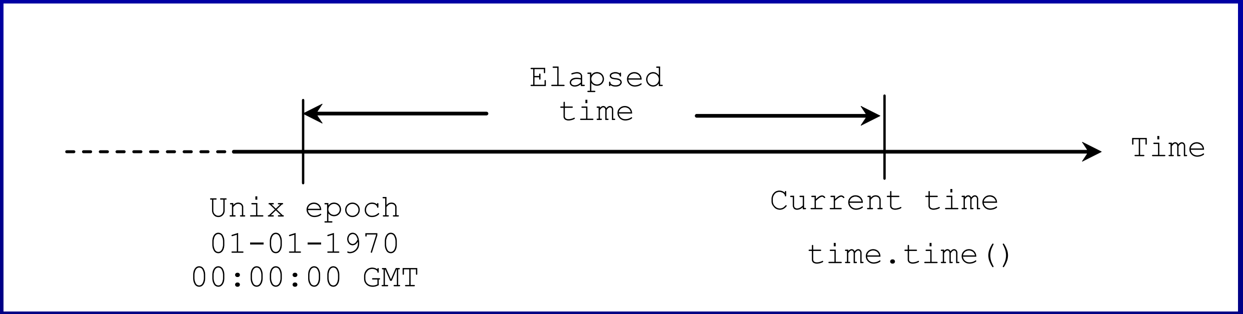
SalesTax

Run

Problem: Displaying Current Time

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

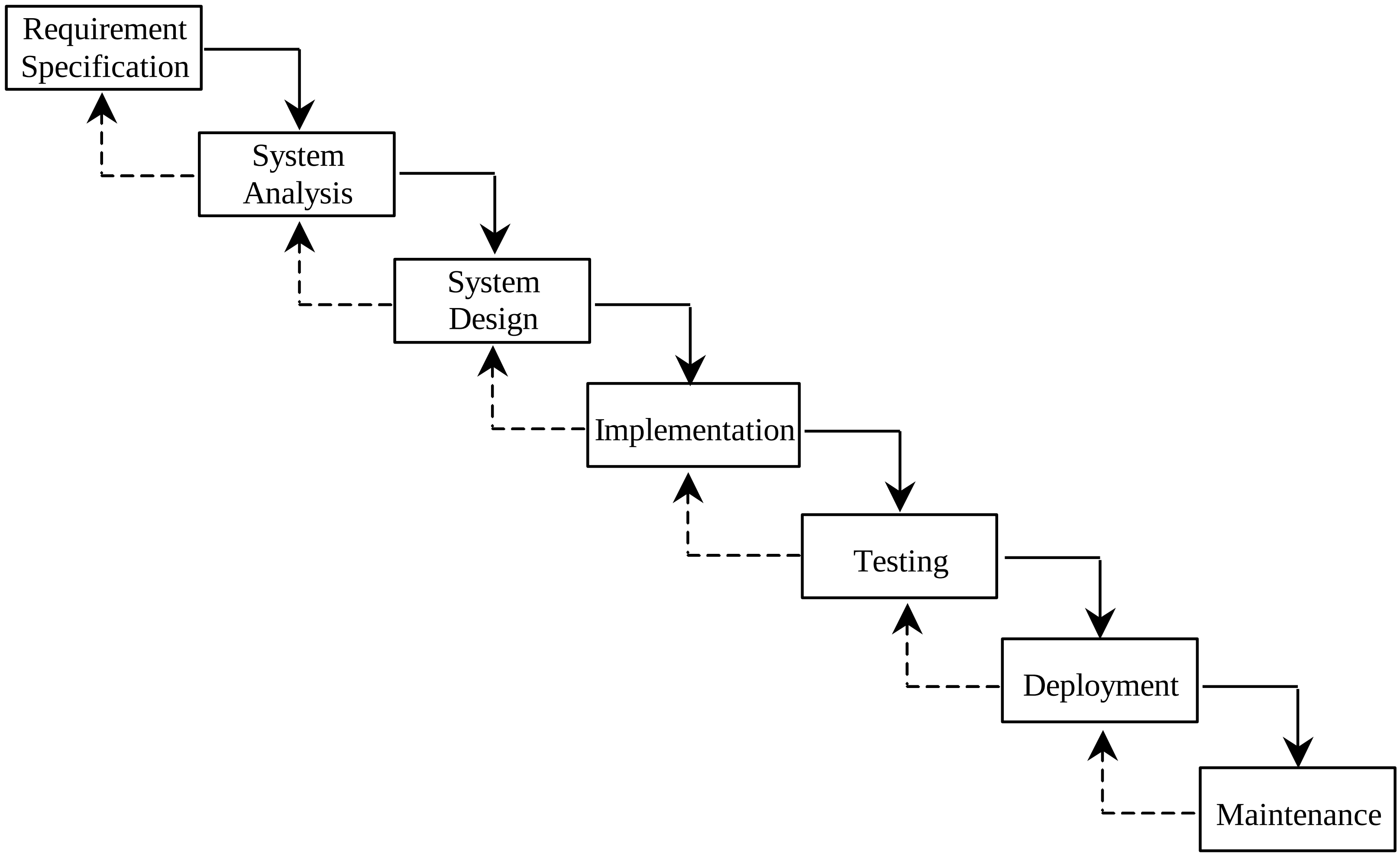
The time.time() function returns the current time in seconds with millisecond precision since the midnight, January 1, 1970 GMT. (1970 was the year when the Unix operating system was formally introduced.) You can use this function to obtain the current time, and then compute the current second, minute, and hour as follows.



ShowCurrentTime

Run

Software Development Process


Requirement Specification

System Analysis

System Design

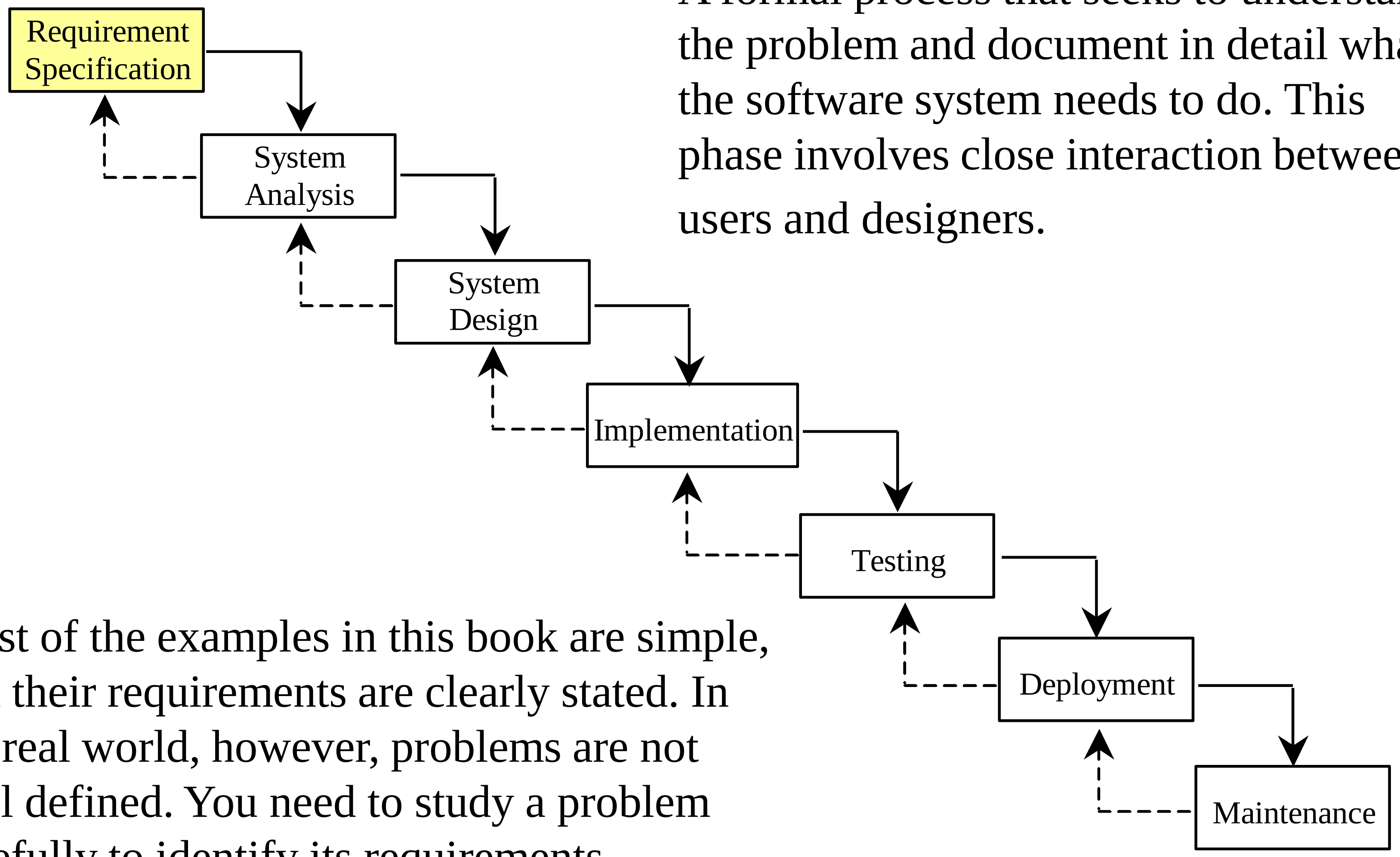
Implementation

Testing

Deployment

Maintenance

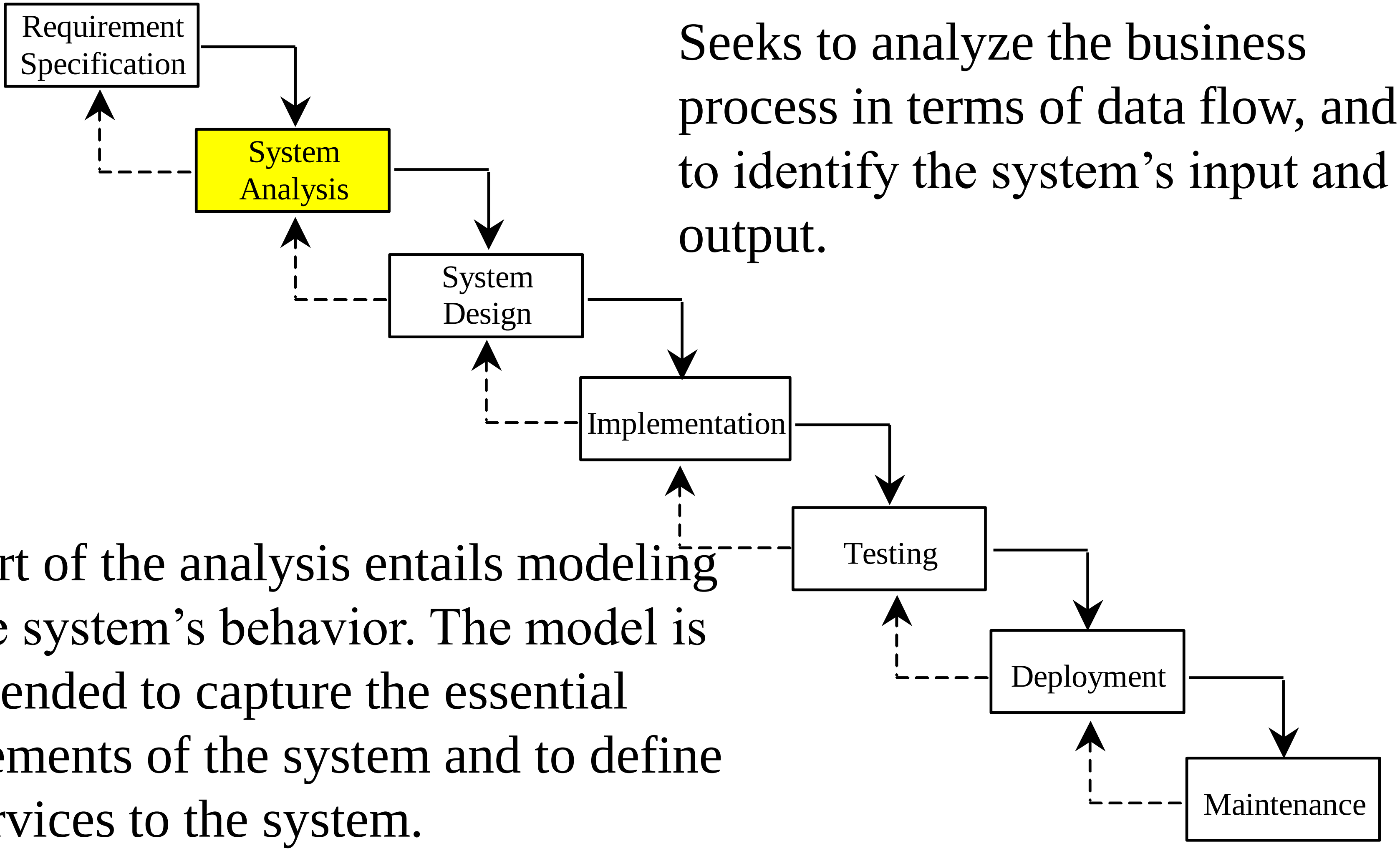
Requirement Specification



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.

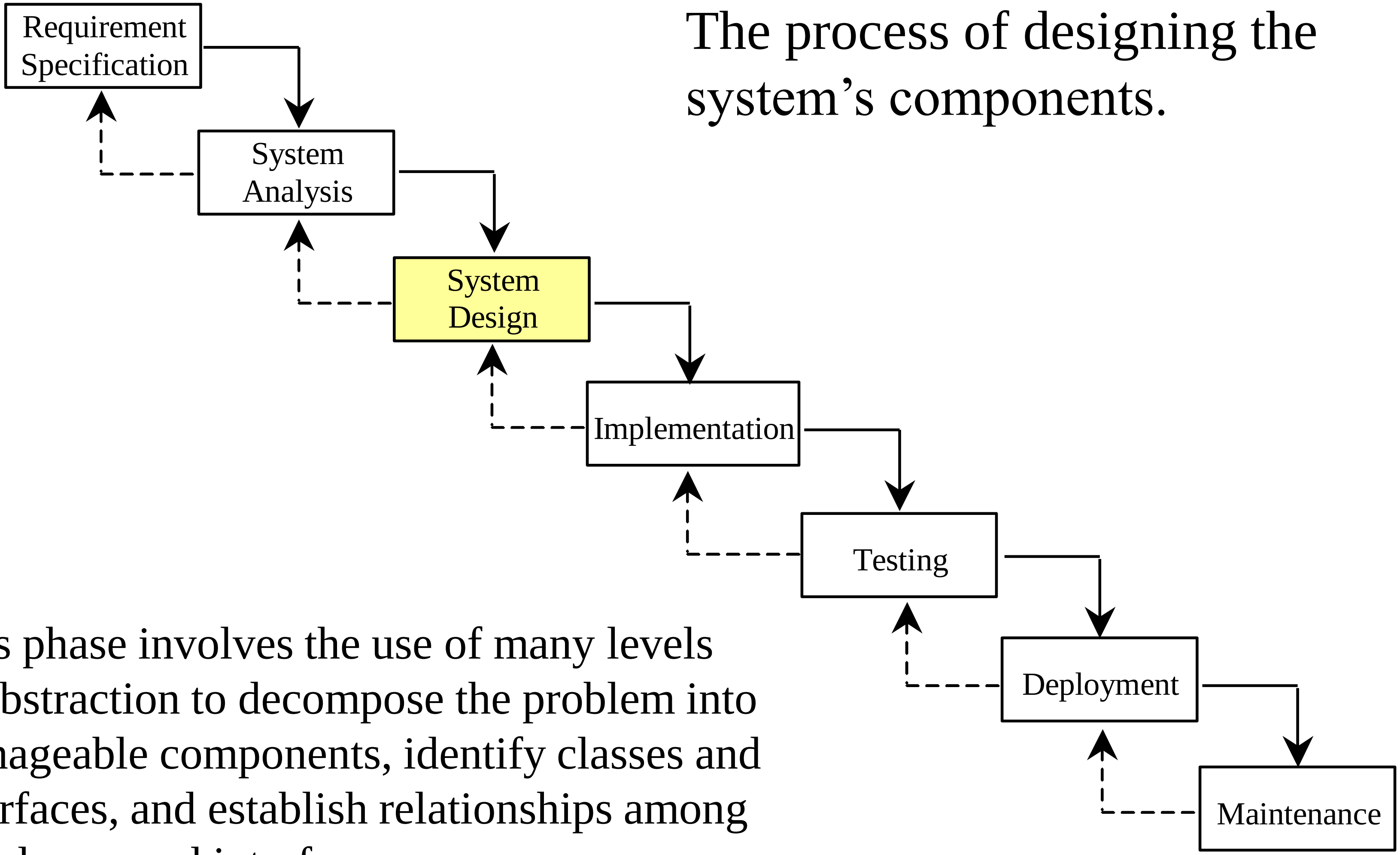
System Analysis



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.

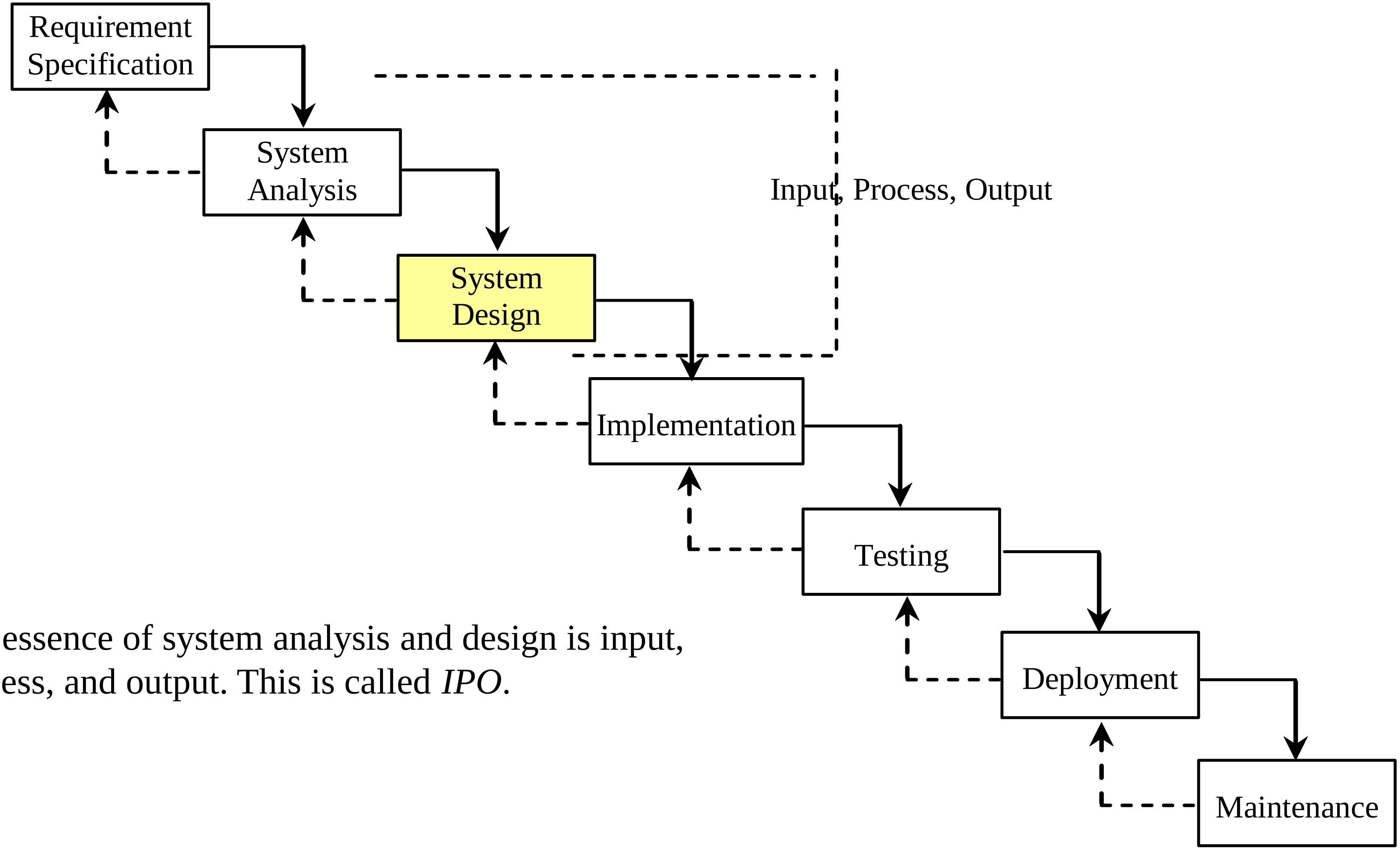
System Design



The process of designing the system’s components.

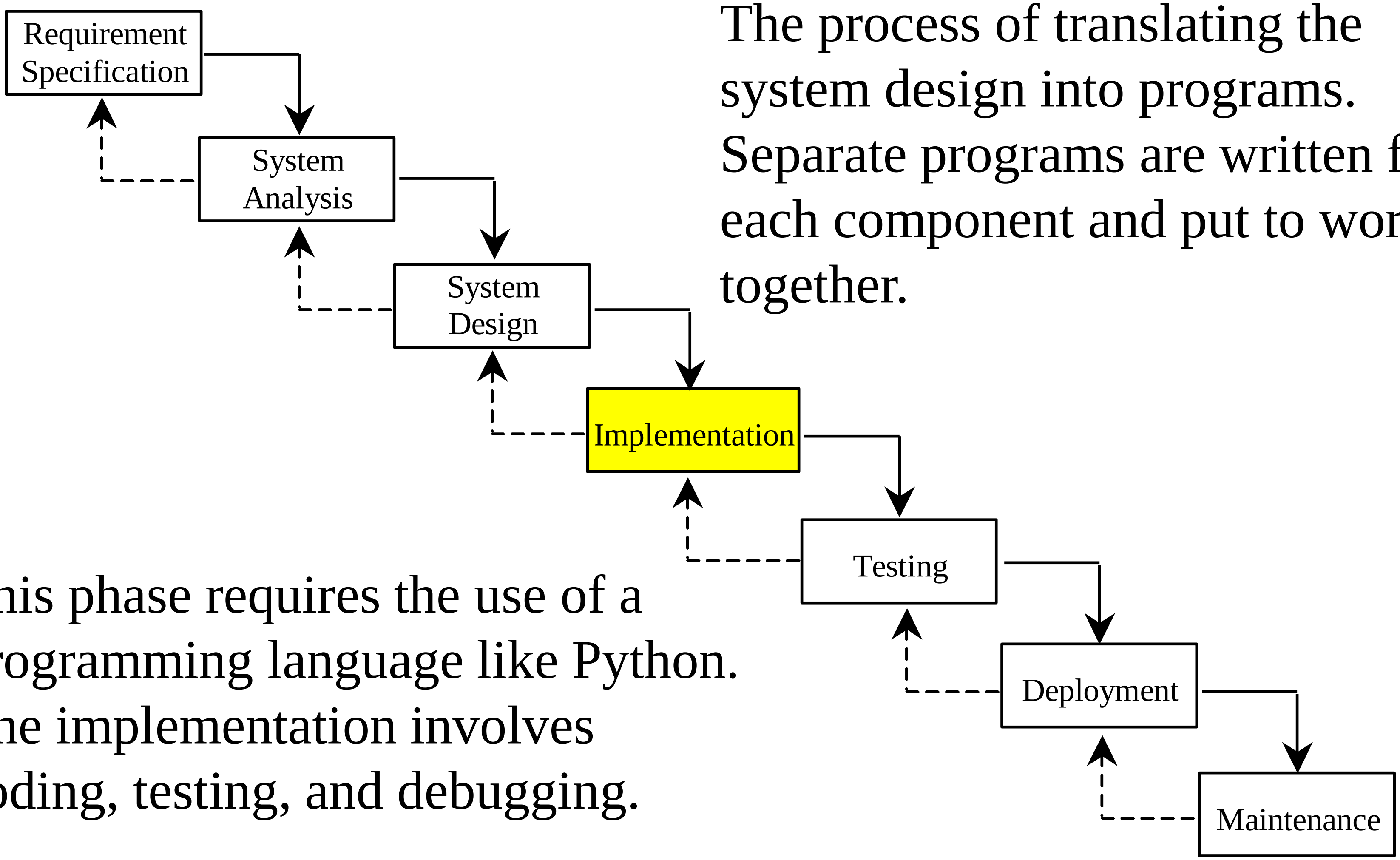
This phase involves the use of many levels of abstraction to decompose the problem into manageable components, identify classes and interfaces, and establish relationships among the classes and interfaces.

IPO



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

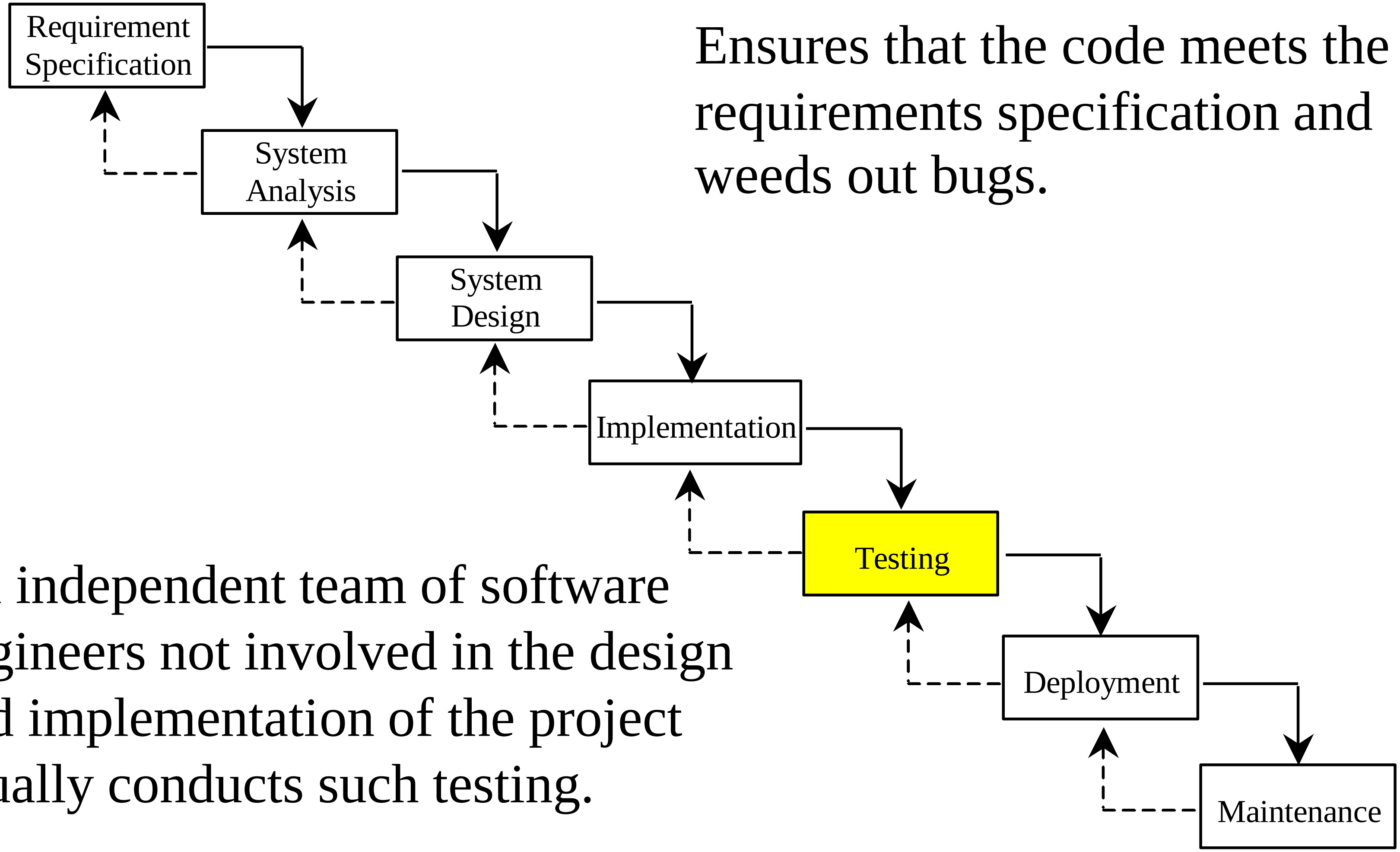
Implementation



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.

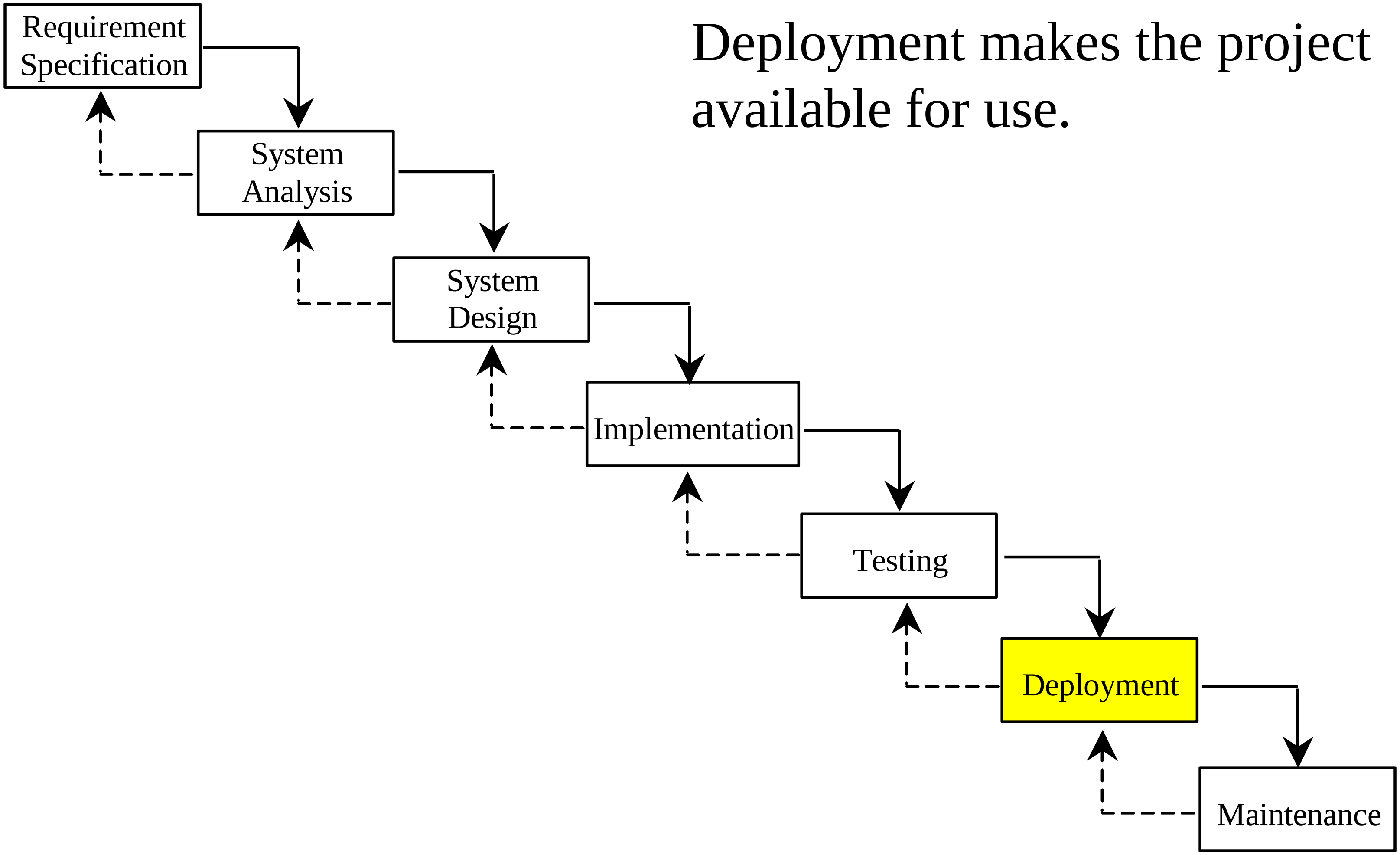
Testing



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

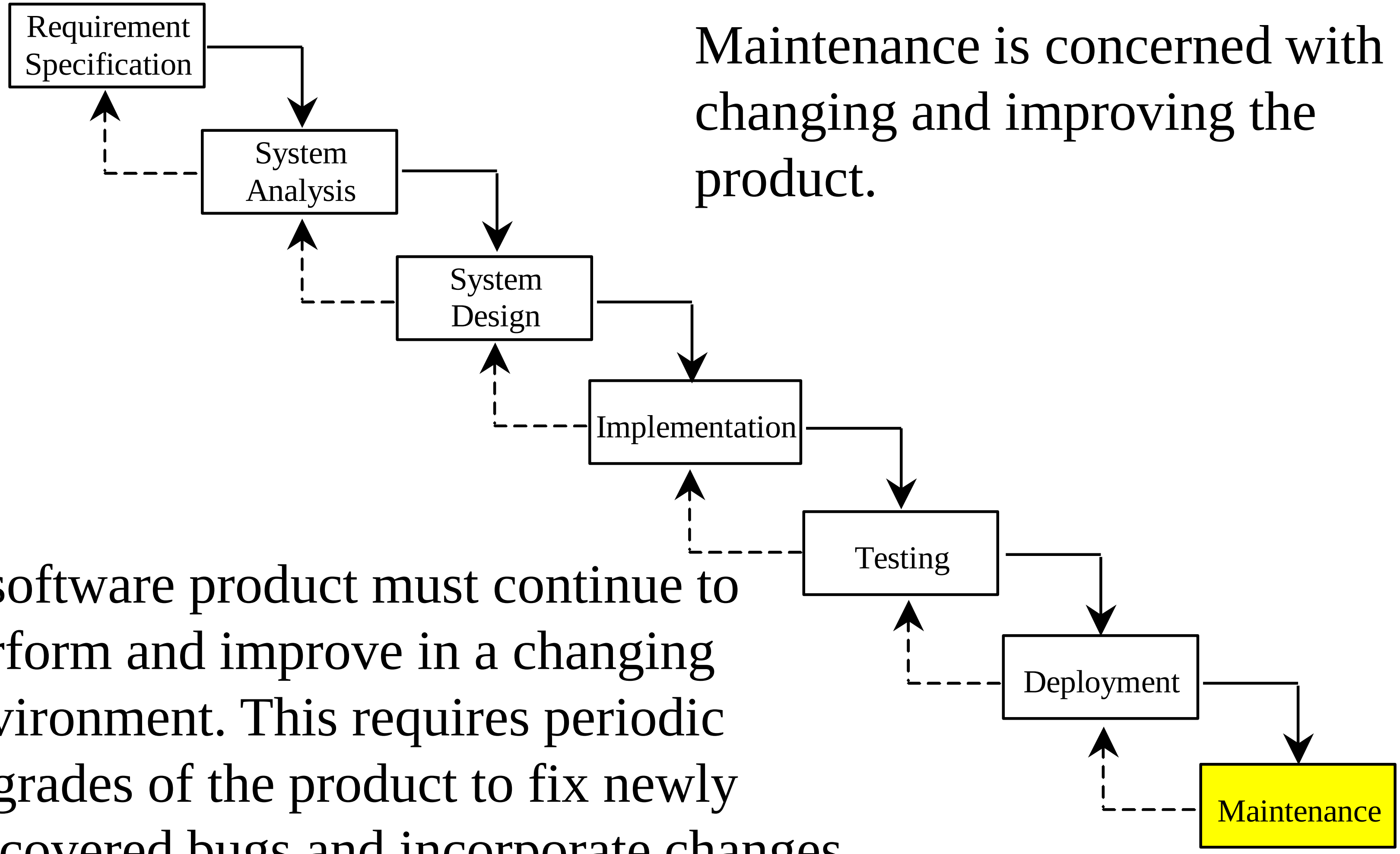
An independent team of software engineers not involved in the design and implementation of the project usually conducts such testing.

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.

Problem:
Computing Loan Payments

This program lets the user enter the interest rate, number of years, and loan amount, and computes monthly payment and total payment.

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

ComputeLoan

Run

Case Study: Computing Distances

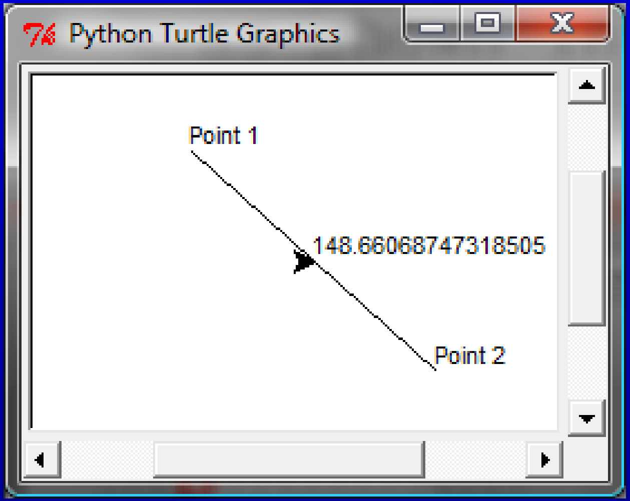
This program prompts the user to enter two points, computes their distance, and displays the points.

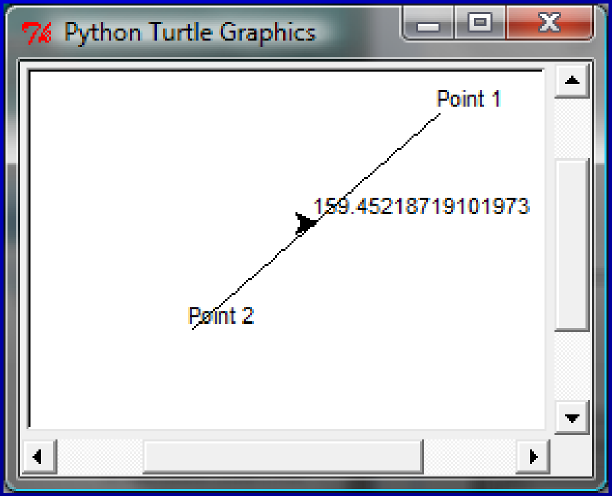
ComputeDistance

Run

Case Study: Computing Distances

This program prompts the user to enter two points, computes their distance, and displays the points and their distances in graphics.





ComputeDistanceGraphics

Run