PROGRAMMING IN PYTHON

UNIT-I

CH-2: Input, Processing, Output

Ms. SALMA BEGUM

Assistant Professor, Dept. Of Computer Science

RBVRR Women's College, Narayanaguda, Hyderabad.

CONTENT

2.1 Designing a Program
- Program Development Life Cycle
2.2 Input, Processing, and Output
2.3 Displaying Output with the print Function
2.4 Comments
2.5 Variables
2.6 Reading Input from the Keyboard
2.7 Performing Calculations
2.7.1. Operators
2.7.2. Type conversions
2.7.3. Expressions
2.8 More About Data Output

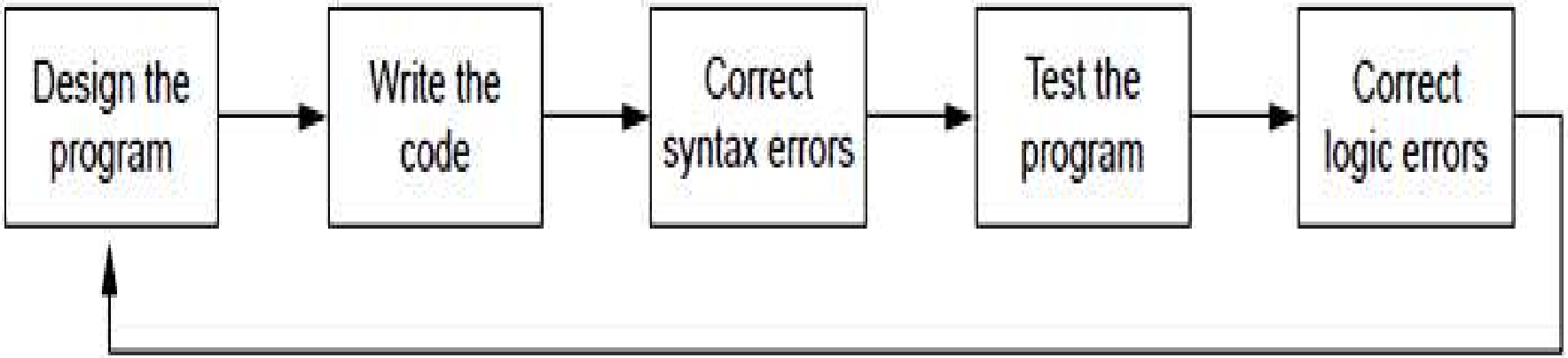
2.1 Designing a Program

Programs must be carefully designed before they are written. During the design process, programmers use tools such as pseudocode and flowcharts to create models of programs.

- **The Program Development Cycle**

  The process of creating a program that works correctly typically requires the five phases shown in the below Figure. The entire process is known as the *program development cycle.*

Figure The program development cycle



Cycle 1 - Design the Program:

- All professional programmers will tell you that a program should be carefully designed before the code is actually written. When programmers begin a new project, they never jump right in and start writing code as the first step. They start by creating a design of the program. There are several ways to design a program.
- The process of designing a program is the most important part of the cycle. The process of designing a program can be summarized in the following two steps:

1. Understand the task that the program is to perform.

- Understand what a program is supposed to do before we can determine the steps that the program will perform. To get a sense of what a program is supposed to do, the programmer usually interviews the customer.
- During the interview, the customer will describe the task that the program should perform, and the programmer will ask questions to uncover as many details as possible about the task.
- A follow-up interview is needed because customers rarely mention everything they want during the initial meeting, and programmers often think of additional questions.
- The programmer studies the information that was gathered from the customer during the interviews and creates a list of different software requirements.
- A *software requirement is simply a* single task that the program must perform in order to satisfy the customer. Once the customer agrees that the list of requirements is complete, the programmer can move to the next phase.

2. Determine the steps that must be taken to perform the task.

- A programmer breaks down the task that a program must perform. An algorithm is created, which lists all of the logical steps that must be taken. The steps in this list have to be translated into code. Programmers commonly use two tools to help them accomplish this: pseudocode and flowcharts.
- **Pseudocode : S**mall mistakes like misspelled words and forgotten punctuation characters can cause syntax errors, programmers must be careful, when writing code. For this reason, programmers find it helpful to write a program in pseudocode before they write it in the actual code of a programming language.

  The word "pseudo" means fake, so *pseudocode is fake code. It is an informal language* that has no syntax rules and is not meant to be compiled or executed. Instead, programmers use pseudocode to create models, or "mock-ups," of programs. Because programmers don't have to worry about syntax errors while writing pseudocode, they can focus all of their attention on the program's design. Once a satisfactory design has been created with pseudocode, the pseudocode can be translated directly to actual code.
- **Flowcharts :** Flowcharting is another tool that programmers use to design programs. A *flowchart is a* diagram that graphically depicts the steps that take place in a program.. there are three types of symbols in the flowchart: ovals, parallelograms, diamond and a rectangle. Each of these symbols represents a step in the program, described as:
  - ***ovals***, which appear at the top and bottom of the flowchart, are called *terminal symbols.* The *Start* terminal marks the program's starting point and the End terminal marks the program's ending point.
  - ***Parallelograms*** are used as *input symbols* and *output symbols.* They represent steps in which the program reads input or displays output.
  - ***Rectangles*** are used as *processing symbols.* They represent steps in which the program performs some process on data, such as a mathematical calculation.
  - ***Diamond/Rhombus*** are used as *decision making symbol*.

The symbols are connected by arrows that represent the "flow" of the program. To step through the symbols in the proper order, you begin at the *Start terminal and follow the* arrows until you reach the *End terminal.*

Cycle 2-Write the Code: After designing the program, the programmer begins writing code in a high-level language such as Python. Recall from Chapter 1 that each language has its own rules, known as syntax, that must be followed when writing a program. A language's syntax rules dictate things such as how key words, operators, and punctuation characters can be used. A syntax error occurs if the programmer violates any of these rules.

Cycle 3-Correct Syntax Errors: If the program contains a syntax error, or even a simple mistake such as a misspelled key word, the compiler or interpreter will display an error message indicating what the error is. Once all of the syntax errors and simple typing mistakes have been corrected, the program can be compiled and translated into a machine language program (or executed by an interpreter, depending on the language being used).

Cycle 4-Test the Program: Once the code is in an executable form, it is then tested to determine whether any logic errors exist or not. A *logic error is a mistake that does not* prevent the program from running, but causes it to produce incorrect results. (Mathematical mistakes are common causes of logic errors.)

Cycle 5-Correct Logic Errors: If the program produces incorrect results, the programmer *debugs the code.* This means that the programmer finds and corrects logic errors in the program. Sometimes during this process, the programmer discovers that the program's original design must be changed. In this event, the program development cycle starts over and continues until no errors can be found.

2.2 Input, Processing, and Output

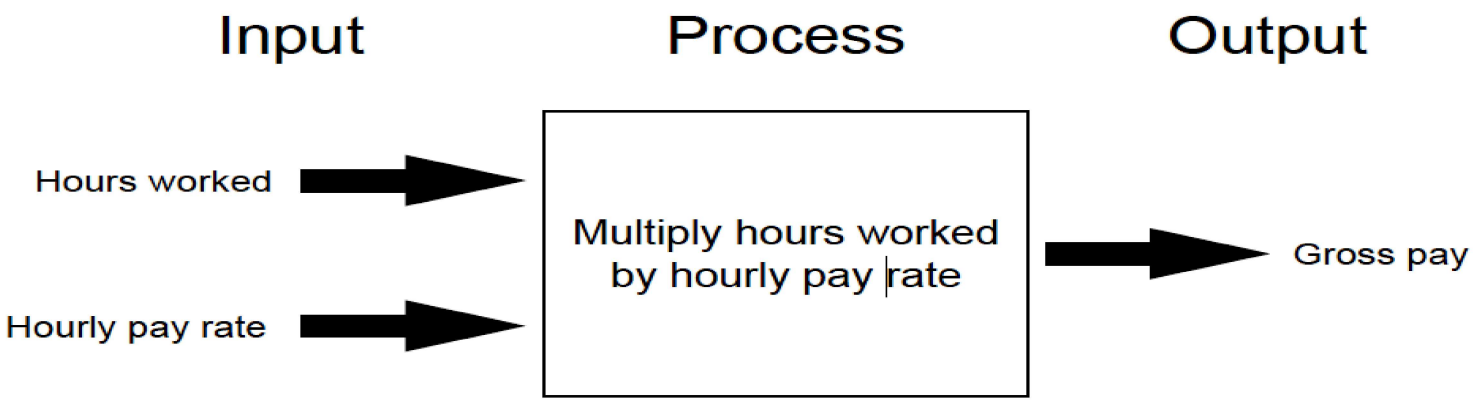
Input is data that the program receives. When a program receives data, it usually processes it by performing some operation with it. The result of the operation is sent out of the program as output.

Computer programs typically perform the following three-step process:

1. Input is received.
2. Some process is performed on the input.
3. Output is produced.

- Input is any data that the program receives while it is running. One common form of input is data that is typed on the keyboard. Once input is received, some process, such as a mathematical calculation, is usually performed on it. The results of the process are then sent out of the program as output.

The input, processing, and output of the pay calculating program



Displaying O/P with the Print Function

- A ***function*** is a piece of prewritten code that performs an operation.
- *Python has numerous* built-in functions that perform various operations. Perhaps the most fundamental built-in function is the print function, which displays output on the screen.

General format of ***print*** function is: *print("String")*

Eg:

```
print('Hello world')
```

- When programmers execute a function, means *calling the function.* When we call the print function, we type the word print, followed by a set of parentheses. Inside the parentheses, we type an argument, which is the data we want to displayed on the screen.
- Notice that the quote marks are not displayed when the statement executes. The quote marks specify the beginning and the end of the text that you wish to display.

Eg:

```
1. print('Kate Austen')
2. print('123 Full Circle Drive')
3. print('Asheville, NC 28899')
```

Program Output

Kate Austen

123 Full Circle Drive

Asheville, NC 28899

Cont.....

- A sequence of characters that is used as data is called a ***string***.
- A string which appears in the code of a program is called a ***string literal***.
- In Python code, string literals must be enclosed in quote marks, which mark where the string data begins and ends.
- ***String literals*** are enclosed in a set of ***single-quote marks*** (') or a set of ***double quote marks*** (") or ***triple quote marks(***either """ or ''').

Eg:

1. print('Kate Austen')
2. print("Kate Austen")
3. print("'Kate Austen'")

- Triple quotes can also be used to surround multiline strings, where single and double quotes cannot be used.

Eg:

```
print("""One
Two
Three""")
```

Cont.....

- **Single quote or apostrophe as String literals**

Eg: print("Don't fear!")

- **Single-quote marks that contains double quotes as string literal**

Eg: print('Your assignment is to read "Hamlet" by tomorrow.')

- **Displaying Multiple Items with the print Function**

The following two statements :

1. print('I am staying in room number')
2. print(room)

We called the print function twice because we needed to display two pieces of data. Line-1 displays the string literal *'I am staying in room number*', and line-2 displays the value referenced by the *room* variable.

This program can be simplified, because Python allows us to display multiple items with one call to the print function by using commas to separate items.

Program:

1. # This program demonstrates a variable.
2. room = 503
3. print('I am staying in room number', room)

Output:

I am staying in room number 503

Comments

- Comments are notes of explanation that document lines or sections of a program.
- They are part of the program, but the Python interpreter ignores them.
- They are intended for people who may be reading the source code.
- In Python you begin a comment with the # character. When the Python interpreter sees a # character, it ignores everything from that character to the end of the line.

Eg-1:

```
# This program displays
# a person's name and address.
print('Kate Austen')
print('123 Full Circle Drive')
```

Eg-2:

```
print('Kate Austen') # Display the name.
print('123 Full Circle Drive') # Display the address.
print('Asheville, NC 28899') # Display the city, state, and ZIP.
```

Variables

- A ***variable*** is a name that represents a value stored in the computer's memory.
- Programs use variables to access and manipulate data that is stored in memory.
- An assignment statement is used to create a variable and make it reference a piece of data

  Syntax: *variable = expression*
- The equal sign (=) is known as the *assignment operator.* In the general format, *variable* is the name of a variable and *expression* is a value, or any piece of code that results in a value.
- After an assignment statement executes, the variable listed on the left side of the = operator will reference the value given on the right side of the = operator.

  Example:

  name="Samuel"

  Age=90

Cont……

Variable Naming Rules

The following rules must be followed for creating a variable:

- We cannot use key words as a variable name.
- A variable name cannot contain spaces.
- The first character must be one of the letters a through z, A through Z, or an underscore character (_).
- After the first character you may use the letters a through z or A through Z, the digits 0 through 9, or underscores.
- Uppercase and lowercase characters are distinct. This means the variable name ItemsOrdered is not the same as itemsordered.

Variable Name Legal or Illegal?

- units_per_day → Legal
- dayOfWeek →Legal
- 3dGraph → Illegal. Variable names cannot begin with a digit.
- June1997 → Legal
- Mixture#3 → Illegal. Variable names may only use letters, digits, or underscores.

Cont……

Variable Reassignment

- Variables are called "variable" because they can reference different values while a program is running. When you assign a value to a variable, the variable will reference that value until you assign it a different value.
- For example, in the below Program. The statement in line 3 creates a variable named ***dollars*** and assigns a value 2.75. Then, the statement in line 6 assigns a different value, 99.95, to the ***dollars*** variable.
- The old value, 2.75, is still in the computer's memory, but it can no longer be used because it isn't referenced by a variable.
- When a value in memory is no longer referenced by a variable, the Python interpreter automatically removes it from memory through a process known as *garbage collection.*

PROGRAM

```
1.  # This program demonstrates variable reassignment.
2.  # Assign a value to the dollars variable.
3.  dollars = 2.75
4.  print('I have', dollars, 'in my account.')
5.  # Reassign dollars so it references a different value.
6.  dollars = 99.95
7.  print('But now I have', dollars, 'in my  account!')
```

Cont……

Numeric Data Types and Literals

- Because different types of numbers are stored and manipulated in different ways, Python uses *data types* to categorize values in memory.
- When an integer is stored in memory, it is classified as an *int*, and when a real number is stored in memory, it is classified as a *float*.
- A number that is written into a program's code is called a *numeric literal. When the Python interpreter reads a numeric literal* in a program's code, it determines its data type according to the following rules:
  - A numeric literal that is written as a whole number with no decimal point is considered an int. Examples are 7, 124, and −9.
  - A numeric literal that is written with a decimal point is considered a float. Examples are 1.5, 3.14159, and 5.0.
- When you store an item in memory, it is important for you to be aware of the item's data type. Some operations behave differently depending on the type of data involved, and some operations can only be performed on values of a specific data type.
- As an experiment, we can use the built-in type function in interactive mode to determine the data type of a value. For example:

```
>>> type(1)
<class 'int'>
```

- Here the value 1 is passed as an argument to the type function. The message that is displayed on the next line, <class 'int'>, indicates that the value is an int.

Cont……

Storing Strings with the str Data Type:

In addition to the *int* and *float* data types, Python also has a data type named *str*, which is used for storing strings in memory. The code in Program below shows how strings can be assigned to variables.

```
1. # Create variables to reference two strings.
2. first_name = 'Kathryn'
3. last_name = 'Marino‘
4. Display the values referenced by the variables.
5. print(first_name, last_name)
```

Program Output

Kathryn Marino

Reassigning a Variable to a Different Type:

- A Variable is a mechanism that makes it easy for us to store and retrieve data.
- Internally, the Python interpreter keeps track of the variable names that we create and the pieces of data to which those variable names refer.
- Any time when we need to retrieve one of those pieces of data, we simply use the variable name that refers to it.
- A variable in Python can refer to items of any type. After a variable has been assigned an item of one type, it can be reassigned an item of a different type.

Example:

```
>>> x = 99
>>> print(x)
 99
 >>> x = 'Take me to your leader'
 >>> print(x)
 Take me to your leader.
```

Warnings

- We cannot use currency symbols, spaces, or commas in numeric literals.

Eg:

value = $4,567.99 # Error!

value = 4567.99 # Correct

Reading Input from the Keyboard

- Programs commonly need to read input typed by the user on the keyboard**.**
- A basic input operation: reading data that has been typed on the keyboard.
- When a program reads data from the keyboard, it stores that data in a variable so it can be used later by the program.
- We use Python's built-in *input* function to read input from the keyboard.
- The *input* function reads a piece of data that has been entered at the keyboard and returns it as a string, back to the program.
- We use the input function in an assignment statement that follows this general format:

variable = input(prompt)

Here *prompt* is a string that is displayed on the screen. The string's purpose is to instruct the user to enter a value; *variable* is the name of a variable that references the data that was entered on the keyboard.

Example : name = input('What is your name? ')

Cont.....

When this statement executes, the following things happen:

- The string 'What is your name? ' is displayed on the screen.
- The program pauses and waits for the user to type something on the keyboard and then to press the Enter key.
- When the Enter key is pressed, the data that was typed is returned as a string and assigned to the name variable.

Example 1: Reads single input:

```
>>> name = input('What is your name? ')
What is your name? Holly
>>> print(name)
Holly
```

Example 2: Reads two inputs

```
1 first_name = input('Enter your first name: ') # Get the user's first name.
2 last_name = input('Enter your last name: ') # Get the user's last name.
3 print('Hello', first_name, last_name) # Print a greeting to the user.
```

Output with input:

```
Enter your first name: Vinny
Enter your last name: Brown
Hello Vinny Brown
```

Cont……

Reading Numbers with the input Function

- The input function always returns the user's input as a string, even if the user enters numeric data. For example, suppose you call the input function, type the number 72, and press the Enter key. The value that is returned from the input function is the string '72'.
- This can be a problem if you want to use the value in a math operation. Math operations can be performed only on numeric values, not strings.
- Python has built-in functions that can use to convert a string to a numeric type.

Data Conversion Functions

- int(*item) :* We pass an argument to the int() function and it returns the argument's value converted to an int.
- float(*item) :* We pass an argument to the float() function and it returns the argument's value converted to a float.

For example, suppose we are writing a payroll program and want to get the number of hours that the user has worked.

```
1. string_value = input('How many hours did you work? ')
2. hours = int(string_value)
```

- The first statement gets the number of hours from the user and assigns that value as a string to the string_value variable.
- The second statement calls the int() function, passing string_value as an argument. The value referenced by string_value is converted to an int and assigned to the hours variable.

The following code shows a better approach. This one statement does all the work that the above shown two statements do, and it creates only one variable:

- `hours = int(input('How many hours did you work? '))`

Cont……

Example: Program that uses the input function to read a string, an int, and a float, as input from the keyboard

```
1. # Get the user's name, age, and income.
2. name = input('What is your name? ')
3. age = int(input('What is your age? '))
4. income = float(input('What is your income? '))
5. print('Here is the data you entered:') # Display the data.
6. print('Name:', name)
7. print('Age:', age)
8. print('Income:', income)
```

Output with input :

What is your name? **Chris**
What is your age? **25**
What is your income? 75000.0
Here is the data you entered:
Name: Chris
Age: 25
Income: 75000.0

- The int() and float() functions work only if the item that is being converted contains a valid numeric value. If the argument cannot be converted to the specified data type, an error known as an exception occurs.
- An *exception* is an unexpected error that occurs while a program is running, causing the program to halt if the error is not properly dealt with.

Assignment

- Program to enter a customer's last name. Write a statement that prompts the user to enter this data and assigns the input to a variable.
- Program to enter the amount of sales for the week. Write a statement that prompts the user to enter this data and assigns the input to a variable.

Performing Calculations

Introduction

- Most real-world algorithms require calculations to be performed. A programmer's tools for performing calculations are *math operators.* The below lists the math operators that are provided by the Python language.

  1. '+' : Addition Adds two numbers
  2. '−' : Subtraction Subtracts one number from another
  3. '*' : Multiplication Multiplies one number by another
  4. '/' : Division Divides one number by another and gives the result as a floating-point number
  5. '//' : Integer division Divides one number by another and gives the result as an integer
  6. '**%**' : Remainder Divides one number by another and gives the remainder
  7. '**': Exponent Raises a number to a power

- Programmers use the math operators to create math expressions. A *math expression* performs a calculation and gives a value. The following is an example of a simple math expression:

  12 + 2

- Python has numerous operators that can be used to perform mathematical calculations.

Cont.....

- Variables may also be used as operands in a math expression.

For example, suppose we have two variables named *hours* and *pay_rate*. The following math expression uses the * operator to multiply the value referenced by the *hours* variable by the value referenced by the *pay_rate* variable:

*hours * pay_rate*

- When we use a math expression to calculate a value, we want to save that value in memory so that it can be use it in the program. We do this with an assignment statement.

Example: Program (simple_math.py)

```
1. salary = 2500.0 # Assign a value to the salary variable.
2. bonus = 1200.0 # Assign a value to the bonus variable.
3. # Calculate the total pay by adding salary and bonus.
4. # Assign the result to pay.
5. pay = salary + bonus
6. print('Your pay is', pay) # Display the pay.
```

Program Output

```
Your pay is 3700.0
```

Example

Suppose a retail business is planning to have a storewide sale where the prices of all items will be 20 percent off. We have been asked to write a program to calculate the sale price of an item after the discount is subtracted. Here is the algorithm:

1. *Get the original price of the item.*
2. *Calculate 20 percent of the original price. This is the amount of the discount.*
3. *Subtract the discount from the original price. This is the sale price.*
4. *Display the sale price.*

Program (sale_price.py)

```
1. # This program gets an item's original price and calculates its sale price, with a 20% discount.
2. original_price = float(input("Enter the item's original price: ")) # Get the item's original price.
3. discount = original_price * 0.2  # Calculate the amount of the discount.
4. sale_price = original_price − discount # Calculate the sale price.
5. print('The sale price is', sale_price) # Display the sale price.
```

Output with input:

Enter the item's original price: **100.00**
The sale price is 80.0

Floating-Point and Integer Division

- Python has two different division operators. The / operator performs floating-point division, and the // operator performs integer division. Both operators divide one number by another. The difference between them is that the / operator gives the result as a floating-point value, and the // operator gives the result as an integer.

Eg:
```
>>> 5 / 2
2.5
```

- We used the / operator to divide 5 by 2. And the result is 2.5.

Eg: The // operator to perform integer division:

```
>>> 5 // 2
2
```

The result is 2. The // operator works like this:

- When the result is positive, it is truncated, which means that its fractional part is thrown away.

- When the result is negative, it is rounded away from zero to the nearest integer.

For Example

```
>>> −5 // 2
−3
```

Cont.....

Operator Precedence

- First, operations that are enclosed in parentheses are performed. Then, when two operators share an operand, the operator with the higher precedence is applied first. The precedence of the math operators, from highest to lowest, are:
  1. Exponentiation: (**)
  2. Multiplication, division, and remainder: (*, /, //, %)
  3. Addition and subtraction: (+, −)
- When two operators with the same precedence share an operand, the operators execute from *left to right*.

For Example:

outcome = 12.0 + 6.0 / 3.0

- The value that will be assigned to *outcome* is 14.0 because the division operator has a higher precedence than the addition operator. As a result, the division takes place before the addition.

Grouping with Parentheses

- Parts of a mathematical expression may be grouped with parentheses to force some operations to be performed before others.
- In the following statement, the variables a and b are added together, and their sum is divided by 4: **result = (a + b) / 4**

Cont.....

Suppose you have taken three tests in your computer science class, and you want to write a program that will display the average of the test scores.

Algorithm:

1. *Get the first test score.*
2. *Get the second test score.*
3. *Get the third test score.*
4. *Calculate the average by adding the three test scores and dividing the sum by 3.*
5. *Display the average.*

Program

```
# Get three test scores and assign them to the test1, test2, and test3 variables.
test1 = float(input('Enter the first test score: '))
test2 = float(input('Enter the second test score: '))
test3 = float(input('Enter the third test score: '))
# Calculate the average of the three scores and assign the result to the average variable.
average = (test1 + test2 + test3) / 3.0
print('The average score is', average) # Display the average.
```

Program Output with input:

Enter the first test score: **90**
Enter the second test score: **80**
Enter the third test score: **100**
The average score is 90.0

The Exponent Operator

- In addition to the basic math operators for addition, subtraction, multiplication, and division, Python also provides an exponent operator (**). Its purpose is to raise a number to a power.
- For example, the following statement raises the *length* variable to the power of 2 and assigns the result to the *area* variable:

  *area = length**2*

The Remainder Operator

- In Python, the % symbol is the remainder operator. (This is also known as the *modulus operator.)* The remainder operator performs division, but instead of returning the quotient, it returns the ***remainder***.
- **For Example**:

  leftover = 17 % 3
- This statement assigns 2 to *leftover* because 17 divided by 3 is 5 with a remainder of 2.
- The remainder operator is useful in calculations that convert times or distances, detect odd or even numbers, and perform other specialized operations.

Cont.....

For example, Program gets a number of seconds from the user, and it converts that number of seconds to hours, minutes, and seconds.

Program (time_converter.py)

```
1. # Get a number of seconds from the user.
2. total_seconds = float(input('Enter a number of seconds: '))
3. hours = total_seconds // 3600  # Get the number of hours.
4. minutes = (total_seconds // 60) % 60
5. # Get the number of remaining seconds.
6. seconds = total_seconds % 60
7. # Display the results.
8. print('Here is the time in hours, minutes, and seconds:')
9. print('Hours:', hours)
10. print('Minutes:', minutes)
11. print('Seconds:', seconds)
```

Program Output with input:

Enter a number of seconds: **11730**

Here is the time in hours, minutes, and seconds:

- Hours: 3.0
- Minutes: 15.0
- Seconds: 30.0

Converting Math Formulas to Programming Statements

- *Python,* as well as other programming languages, requires an operator for any mathematical operation.
- When converting some algebraic expressions to programming expressions, we may have to insert parentheses that do not appear in the algebraic expression.
- The below tables shows some algebraic expressions that perform multiplication and the equivalent programming expressions.

Algebraic expressions

Algebraic Expression	Operation Being Performed	Programming Expression
$6B$	6 times B	`6 * B`
$(3)(12)$	3 times 12	`3 * 12`
$4xy$	4 times x times y	`4 * x * y`

Algebraic and programming expressions

Algebraic Expression	Python Statement
$y = 3\frac{x}{2}$	`y = 3 * x / 2`
$z = 3bc + 4$	`z = 3 * b * c + 4`
$a = \frac{x + 2}{b - 1}$	`a = (x + 2) / (b - 1)`

Mixed-Type Expressions and Data Type Conversion

- When you perform a math operation on two operands, the data type of the result will depend on the type of the operands.
- Python follows these rules when evaluating mathematical expressions:
  - When an operation is performed on two int values, the result will be an int.
  - When an operation is performed on two float values, the result will be a float.
  - When an operation is performed on an int and a float, the int value will be temporarily converted to a float and the result of the operation will be a float.
- An expression that uses operands of different data types is called a ***mixed-type expression***.

Example for mixed-type expressions:

*my_number = 5 * 2.0*

- When this statement executes, the value 5 will be converted to a float (5.0) and then multiplied by 2.0, leading to the result, 10.0, assigned to *my_number*.

Cont.....

Breaking Long Statements into Multiple Lines

- Most programming statements are written in one line. If a programming statement is too long, will not be able to view it in editor window without scrolling horizontally.
- In addition, if you print your program code on paper and one of the statements is too long to fit on one line, it will wrap around to the next line and make the code difficult to read.
- Python allows you to break a statement into multiple lines by using the *line continuation character 'backslash (\)'.*
- *We type the backslash character at the point* you want to break the statement, and then press the Enter key. Here is a print function call that is broken into two lines with the line continuation character:

```
print('We sold', units_sold, \
                'for a total of', sales_amount)
```

- The line continuation character that appears at the end of the first line tells the interpreter that the statement is continued on the next line.

Example: statement that performs a mathematical calculation and has been broken up to fit on two lines:

```
result = var1 * 2 + var2 * 3 + \
 var3 * 4 + var4 * 5
```

Assignment

- Complete the following table by writing the value of each expression in the Value column.

Expression		Value
6 + 3 * 5	=	
12 / 2 − 4	=	
9 + 14 * 2 − 6	=	
(6 + 2) * 3	=	
14 / (11 − 4)	=	
9 + 12 * (8 − 3)	=	

- What value will be assigned to result after the following statement executes? result = 9 // 2
- What value will be assigned to result after the following statement executes? result = 9 % 2

2.8. More About Data Output

Suppressing the print Statement's Newline

The *print* statement displays a string and then prints an unseen newline character.

For Example:

```
print('One')
print ('Two')
print ('Three')
```

Output:

```
One
Two
Three
```

If you do not want to start a new line of output, you can use 'end' argument in the print function

For Example:

```
print ('One', end= `  `)
print ('Two', , end= `  `)
print ('Three' )
```

Output:

```
One  Two  Three
```

The argument end=' ' is passed to the *print* function. This specifies that the *print* function should print a space instead of a newline character at the end of its output.

Specifying an Item Separator

- When multiple arguments are passed to the print function, they are automatically separated by a space when they are displayed on the screen. Here is an example, demonstrated

Example:

```
>>> print('One', 'Two', 'Three')
One Two Three
```

- If you do not want a space printed between the items, you can pass the argument sep=' ‘ to the print function, as shown here:

```
>>> print('One', 'Two', 'Three', sep='')
OneTwoThree
```

- We can also use this special argument to specify a character other than the space to separate multiple items. Here is an example:

```
>>> print('One', 'Two', 'Three', sep='*')
One*Two*Three
```

Cont.....

Escape Characters

- An *escape character is a special character that is preceded with a backslash (\), appearing* inside a string literal.
- The escape characters are treated as special commands that are embedded in the string.
- For example, \n is the newline escape character, which print data in the next line.

For example:

```
print('One\nTwo\nThree')
```

Output:

```
One
Two
Three
```

Some of Python's escape characters

Escape Character	Effect
\n	Causes output to be advanced to the next line.
\t	Causes output to skip over to the next horizontal tab position.
\'	Causes a single quote mark to be printed.
\"	Causes a double quote mark to be printed.
\\	Causes a backslash character to be printed.

Cont.....

- The \t escape character advances the output to the next horizontal tab position. (A tab position normally appears after every eighth character.)

Example:

```
print('Mon\tTues\tWed')
print('Thur\tFri\tSat')
```

Output:

Mon Tues Wed

Thur Fri Sat

- We can use the \' and \" escape characters to display quotation marks.

Example:

```
print("Your assignment is to read \"Hamlet\" by tomorrow.")
print('I\'m ready to begin.')
```

Output:

Your assignment is to read "Hamlet" by tomorrow.
I'm ready to begin.

- We can use the \\ escape character to display a backslash, as shown in the following:

  ```
  print('The path is C:\\temp\\data.')
  ```

Output: The path is C:\temp\data.

Displaying Multiple Items with the + Operator

- The + operator is used with two strings to performs *string concatenation*.

Example: print('This is ' + 'one string.') **Output:** This is one string.

- String concatenation can be useful for breaking up a string literal. Here is an example:

Example: print('Enter the amount of ' + \
'sales for each day and ' + \
'press Enter.')

Formatting Numbers

- When a floating-point number is displayed by the print function, it can appear with up to 12 significant digits.

Program

```
1. # This program demonstrates how a floating-point number is displayed with no formatting.
2. amount_due = 5000.0
3. monthly_payment = amount_due / 12.0
4. print('The monthly payment is', monthly_payment)
```

Program Output

The monthly payment is 416.666666667

- Python provide the built-in *format* function, when we call the built-in format function, we pass two arguments to the function: a ***numeric value*** and a ***format specifie***r.
- The ***format specifier*** is a string that contains special characters specify how the numeric value should be formatted.

Example:

format(12345.6789, '.2f') , Here the first argument represents floating-point number 12345.6789 and the second argument, '.2f', is the format specifier, which specifies .2 precision. The ‘f ‘ specifies floating point data type . The number is rounded to two decimal places.

Cont.....

Formatting in Scientific Notation:

If you prefer to display floating-point numbers in scientific notation, you can use the letter ‘e ‘or ‘ E’ instead of ‘f.’

Example: print(format(12345.6789, 'e')) **Output:** 1.234568e+04

Inserting Comma Separators:

If you want the number to be formatted with comma separators, you can insert a comma into the format specifier, as shown here:

```
>>> print(format(12345.6789, ',.2f'))
12,345.68
```

- ***Program to demonstrates how the comma separator and a precision of two decimal places can be used to format larger numbers as currency amounts.***

```
1. # This program demonstrates how a floating-point  number can be displayed as
   currency.
2. monthly_pay = 5000.0
3. annual_pay = monthly_pay * 12
4. print('Your annual pay is $', format(annual_pay, ',.2f'), sep='')
```

Program Output

Your annual pay is $60,000.00

Cont.....

Specifying a Minimum Field Width:

The format specifier can also include a minimum field width, which is the minimum number of spaces that should be used to display the value. The following example prints a number in a field that is 12 spaces wide:

```
>>> print('The number is', format(12345.6789, '12,.2f')) s
    The number is 12,345.68
```

- If a value is too large to fit in the specified field width, the field is automatically enlarged to accommodate it.
- Field widths can help to print numbers aligned in columns.

Formatting a Floating-Point Number as a Percentage:

We can use the % symbol to format a floating point number as a percentage. It causes the number to be multiplied by 100 and displayed with a % sign following it.

Example 1:

```
>>> print(format(0.5, '%'))
50.000000%
```

Example that specifies 0 as the precision:

```
>>> print(format(0.5, '.0%'))
50%
```

Formatting Integers

- We can also use the format function to format integers. There are two differences to keep in mind when writing a format specifier that will be used to format an integer: a) We use ‘d’ as the type designator. b) we cannot specify precision.
- In the following session, the number 123456 is printed with no special formatting:

```
>>> print(format(123456, 'd'))
        123456
```

- In the following session, the number 123456 is printed with a comma separator:

```
>>> print(format(123456, ',d'))
        123,456
```

- In the following session, the number 123456 is printed in a field that is 10 spaces wide:

```
>>> print(format(123456, '10d'))
        123456
```

- In the following session, the number 123456 is printed with a comma separator in a field that is 10 spaces wide:

```
>>> print(format(123456, '10,d'))
        123,456
```

Exercises/Assignment

Chapter-2 Exercises.docx