



الجامعة التقنية الجنوبية المعهد التقني القرنة قسم تقنيات أنظمة الحاسوب



البرمجة بلغة بايثون
Python Programming

محاضرة رقم (1)

Introduction

- ▶ Computers can perform such a wide variety of tasks because they can be programmed. This means that computers are not designed to do just one job, but to do any job that their programs tell them to do.
- ▶ A program is a set of instructions that a computer follows to perform a task.
- ▶ Programs are commonly referred to as software. Software is essential to a computer because it controls everything the computer does.
- ▶ This course introduces you to the fundamental concepts of computer programming using the Python language. The Python language is a good choice for beginners because it is easy to learn, and programs can be written quickly using it.
- ▶ Python is a general-purpose language created in the early 1990s. It has become popular in business and academic applications.
- ▶ Python is also a powerful language, popular with professional software developers.

Compilers and Interpreters

- ▶ Because the CPU understands only machine language instructions, programs that are written in a high-level language must be translated into machine language.
- ▶ Depending on the language in which a program has been written, the programmer will use either a **compiler** or an **interpreter** to make the translation.
- ▶ A compiler is a program that translates a high-level language program into a separate machine language program. The machine language program can then be executed any time it is needed.
- ▶ The Python language uses an interpreter, which is a program that both translates and executes the instructions in a high-level language program.
- ▶ As the interpreter reads each individual instruction in the program, it converts it to machine language instructions then immediately executes them. Because interpreters combine translation and execution, they typically do not create separate machine language programs.

Using Python

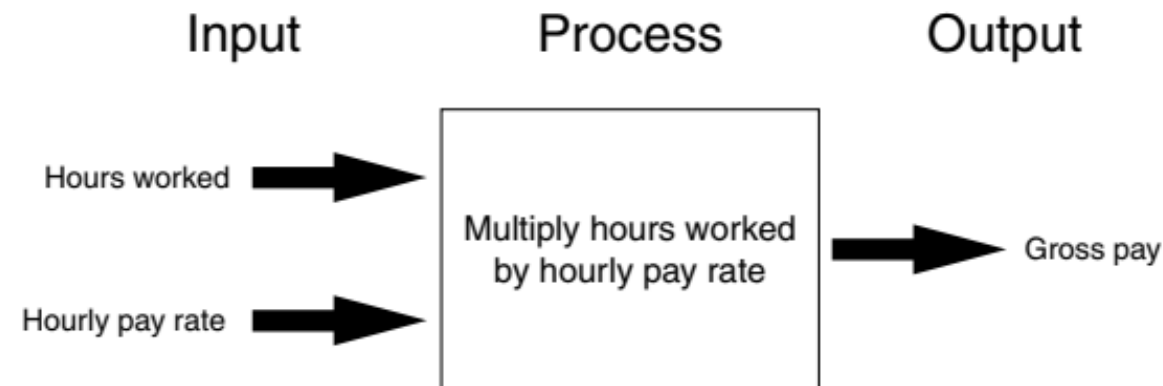
- ▶ Before write any programs of your own, you need to make sure that Python is installed on your computer and properly configured.
- ▶ When you install the Python language on your computer, one of the items that is installed is the Python interpreter.
- ▶ The Python interpreter is a program that can read Python programming statements and execute them.
- ▶ You can use the interpreter in two modes: **interactive mode** and **script mode**. In interactive mode, the interpreter waits for you to type Python statements on the keyboard. Once you type a statement, the interpreter executes it and then waits for you to type another statement.
- ▶ In script mode, the interpreter reads the contents of a file that contains Python statements. Such a file is known as a Python program or a Python script.

- ▶ A(n) _____ program translates a high-level language program into a separate machine language program.
 - ☐ compiler
 - ☐ interpreter
- ▶ Machine language is the only language that a CPU understands.
 - ☐ True
 - ☐ False
- ▶ What is the difference between a compiler and an interpreter?
- ▶ An interpreter is a program that both translates and executes the instructions in a high-level language program.
 - ☐ True
 - ☐ False

Questions

Input, Processing, and 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.



Displaying Output 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.

```
print('Hello world')
```
- ▶ When you call the print function, you type the word print, followed by a set of parentheses. Inside the parentheses, you type an argument, which is the data that you want displayed on the screen.
- ▶ In programming terms, a sequence of characters that is used as data is called a **string**. When a string appears in the actual code of a program, it is called a **string literal**.
- ▶ In Python, you can enclose string literals in a set of **single-quote marks (')** or **double-quote marks (")**.
- ▶ If you want a string literal to contain either a single-quote or an apostrophe as part of the string, you can enclose the string literal in double-quote marks.

Displaying Output with the print Function

Program (output.py)

```
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
```

Program (double_quotes.py)

```
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
```

Program (apostrophe.py)

```
1 print("Don't fear!")
2 print("I'm here!")
```

Program Output

```
Don't fear!
I'm here!
```

Program (display_quote.py)

```
1 print('Your assignment is to read "Hamlet" by tomorrow.')
```

Program Output

```
Your assignment is to read "Hamlet" by tomorrow.
```

Comments

- ▶ **Comments** are short notes placed in different parts of a program, explaining how those parts of the program work. Although comments are a critical part of a program, they are ignored by the Python interpreter.
- ▶ Comments are intended for any person reading a program's code, not the computer.
- ▶ 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.

Program

(comment1.py)

```
1 # This program displays a person's
2 # name and address.
3 print('Kate Austen')
4 print('123 Full Circle Drive')
5 print('Asheville, NC 28899')
```

Program Output

```
Kate Austen
123 Full Circle Drive
Asheville, NC 28899
```

- ▶ Write a statement that displays your name.
- ▶ Write a statement that displays the following text:

`Python's the best!`

- ▶ Write a statement that displays the following text:

`The cat said "meow."`

Questions



الجامعة التقنية الجنوبية المعهد التقني القرنة قسم تقنيات أنظمة الحاسوب



البرمجة بلغة بايثون
Python Programming

محاضرة رقم (2)

Variables

- ▶ Programs usually store data in the computer's memory and perform operations on that data.
- ▶ Programs use **variables** to access and manipulate data that is stored in memory.
- ▶ A **variable** is a name that represents a value in the computer's memory.
- ▶ You use an **assignment statement** to create a variable and make it reference a piece of data. `age = 25`
- ▶ After the above statement executes, a variable named age will be created, and it will reference the value 25.
- ▶ An assignment statement is written in the following general format: `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.

Variable Naming Rules

- ▶ Although you are allowed to make up your own names for variables, you must follow these rules:
 - ▶ You cannot use one of Python's 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`.
- ▶ In addition to following these rules, you should always choose names for your variables that give an indication of what they are used for. For example, a variable that holds the temperature might be named `temperature`, and a variable that holds a car's speed might be named `speed`.

Displaying Multiple Items with the print Function

- ▶ Python allows us to display multiple items with one call to the print function. We simply have to separate the items with **commas** as shown in below program.

Program (variable_demo3.py)

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

Program Output

I am staying in room number 503

- ▶ Variables are called “variable” because they can reference different values while a program is running.

Program (variable_demo4.py)

```
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  
6 # Reassign dollars so it references  
7 # a different value.  
8 dollars = 99.95  
9 print('But now I have', dollars, 'in my account!')
```

- ▶ What is a variable?
- ▶ Which of the following are illegal variable names in Python, and why?

```
x  
99bottles  
july2009  
theSalesFigureForFiscalYear  
r&d  
grade_report
```

- ▶ Is the variable name Sales the same as sales? Why ?
- ▶ Is the following assignment statement valid or invalid? If it is invalid, why? 72 = amount
- ▶ What will the following code display?

```
val = 99  
print('The value is', 'val')
```

Questions

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. As you will see, 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.

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.

Program 2-11 (string_variable.py)

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

Program Output

Kathryn Marino

- ▶ 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.

```
1 >>> x = 99 
2 >>> print(x) 
3 99
```

```
4 >>> x = 'Take me to your leader' 
5 >>> print(x) 
6 Take me to your leader.
```

- ▶ After the following assignment statements execute, what is the Python data type of the values

```
value1 = 99  
value2 = 45.9  
value3 = 7.0  
value4 = 7  
value5 = 'abc'
```

- ▶ What will be displayed by the following program?

```
my_value = 99  
my_value = 0  
print(my_value)
```

Questions

Reading Input from the Keyboard

- ▶ Most of the programs that you will write will need to read input and then perform an operation on that input. In the course, 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 that piece of data, as a string, back to the program.

- ▶ You normally use the input function in an assignment statement that follows this general format:

```
variable = input(prompt)    name = input('What is your name? ')
```

- ▶ When the above 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.

Reading Input from the Keyboard

- ▶ The program below shows a complete program that uses the input function to read two strings as input from the keyboard.

Program

(string_input.py)

```
1  # Get the user's first name.
2  first_name = input('Enter your first name: ')
3
4  # Get the user's last name.
5  last_name = input('Enter your last name: ')
6
7  # Print a greeting to the user.
8  print('Hello', first_name, last_name)
```

Program Output (with input shown in bold)

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

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.
- ▶ 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.
- ▶ Fortunately, Python has built-in functions that you can use to convert a string to a numeric type. Table below summarizes two of these functions.

Table Data conversion functions

Function	Description
<code>int(item)</code>	You pass an argument to the <code>int()</code> function and it returns the argument's value converted to an <code>int</code> .
<code>float(item)</code>	You pass an argument to the <code>float()</code> function and it returns the argument's value converted to a <code>float</code> .

Reading Numbers with the input Function

- ▶ The program below a complete program that uses the input function to read a string, an int, and a float, as input from the keyboard.

Program

(input.py)

```
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
6 # Display the data.
7 print('Here is the data you entered:')
8 print('Name:', name)
9 print('Age:', age)
10 print('Income:', income)
```

Program Output (with input shown in bold)

```
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
```

- ▶ You need the user of a 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.
- ▶ You need the user of a 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.

Questions



الجامعة التقنية الجنوبية المعهد التقني القرنة قسم تقنيات أنظمة الحاسوب



البرمجة بلغة بايثون
Python Programming

محاضرة رقم (3)

Performing Calculations

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

Table Python math operators

Symbol	Operation	Description
+	Addition	Adds two numbers
-	Subtraction	Subtracts one number from another
*	Multiplication	Multiplies one number by another
/	Division	Divides one number by another and gives the result as a floating-point number
//	Integer division	Divides one number by another and gives the result as a whole number
%	Remainder	Divides one number by another and gives the remainder
**	Exponent	Raises a number to a power

Performing Calculations

- ▶ When we use a math expression to calculate a value, normally we want to save that value in memory so we can use it again in the program. We do this with an assignment statement. Program 2-14 shows an example.

Program

(simple_math.py)

```
1  # Assign a value to the salary variable.
2  salary = 2500.0
3
4  # Assign a value to the bonus variable.
5  bonus = 1200.0
6
7  # Calculate the total pay by adding salary
8  # and bonus. Assign the result to pay.
9  pay = salary + bonus
10
11 # Display the pay.
12 print('Your pay is', pay)
```

Program Output

Your pay is 3700.0

Floating-Point and Integer Division

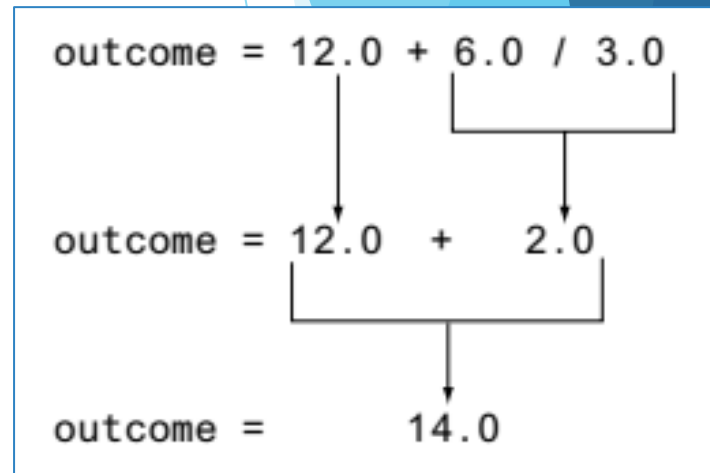
- ▶ Notice that Python has two different division operators. The **/ operator** performs **floating-point division**, and the **// operator** performs **integer division**.
- ▶ Both operators divide one number by another. The difference between them is that the **/ operator** gives the result as a floating-point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

<pre>>>> 5 / 2 <input type="button" value="Enter"/> 2.5</pre>	<pre>>>> 5 // 2 <input type="button" value="Enter"/> 2</pre>	<pre>>>> -5 // 2 <input type="button" value="Enter"/> -3</pre>
--	---	---

- ▶ 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.

Operator Precedence

- ▶ First, operations that are enclosed in parentheses are performed first. 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:
 - ▶ Exponentiation: ******
 - ▶ Multiplication, division, and remainder: *** / // %**
 - ▶ Addition and subtraction: **+ -**
- ▶ Notice the multiplication (*), floating-point division (/), integer division (//), and remainder (%) operators have the same precedence. The addition (+) and subtraction (-) operators also have the same precedence.
- ▶ When two operators with the same precedence share an operand, the operators execute from left to right.



The Augmented Assignment Operators

- ▶ Quite often, programs have assignment statements in which the variable that is on the left side of the = operator also appears on the right side of the = operator. Here are examples: $x = x + 1$, $y = y - 5$.
- ▶ On the right side of the assignment operator, 1 is added to x. The result is then assigned to x, replacing the value that x previously referenced. Effectively, this statement adds 1 to x.
- ▶ These types of operations are common in programming. For convenience, Python offers a special set of operators designed specifically for these jobs. Table below shows the augmented assignment operators.

Table Augmented assignment operators

Operator	Example Usage	Equivalent To
<code>+=</code>	<code>x += 5</code>	<code>x = x + 5</code>
<code>-=</code>	<code>y -= 2</code>	<code>y = y - 2</code>
<code>*=</code>	<code>z *= 10</code>	<code>z = z * 10</code>
<code>/=</code>	<code>a /= b</code>	<code>a = a / b</code>
<code>%=</code>	<code>c %= 3</code>	<code>c = c % 3</code>

- ▶ 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`

Questions

Breaking Long Statements into Multiple Lines

- ▶ Python allows you to break too long statement into multiple lines by using the **line continuation character**, which is a **backslash (\)**.
- ▶ You simply type the backslash character at the point you want to break the statement, then press the Enter key.

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

- ▶ Python also allows you to break any part of a statement that is enclosed in parentheses into multiple lines without using the line continuation character.

```
print("Monday's sales are", monday,  
      "and Tuesday's sales are", tuesday,  
      "and Wednesday's sales are", wednesday)
```

```
total = (value1 + value2 +  
        value3 + value4 +  
        value5 + value6)
```

Suppressing the print Function's Ending Newline

- ▶ The print function normally displays a line of output. For example, the following three statements will produce three lines of output:

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

- ▶ Each of the statements shown here displays a string and then prints a newline character. It causes the output to advance to the next line.
- ▶ If you do not want the print function to start a new line of output when it finishes displaying its output, you can pass the **special argument** `end= ' '` to the function, as shown in the following code:

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

One Two Three

- ▶ Sometimes, you might not want the print function to print anything at the end of its output, not even a space. If that is the case, you can pass the **argument** `end=""` to the print function.

Escape Characters

- ▶ An **escape character** is a special character that is preceded with a **backslash** (\), appearing inside a string literal.
- ▶ When a string literal that contains escape characters is printed, the escape characters are treated as special commands that are embedded in the string.
- ▶ Python recognizes several escape characters, some of which are listed in Table below.

Table 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.

Escape Characters

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

```
One
Two
Three
```

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

```
Mon    Tues    Wed
Thur   Fri     Sat
```

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

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

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

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

- ▶ You saw that the **+** operator is used to add two numbers. When the + operator is used with two strings, however, it performs string concatenation. This means that it appends one string to another.

```
print('This is ' + 'one string.')
```

```
This is one string.
```

Formatting Numbers

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

Program Output

```
The monthly payment is 416.666666667
```

- ▶ Sometimes, it would be nice to see that amount rounded to two decimal places. Fortunately, Python gives us a way to do just that, and more, with the built-in **format function**.
- ▶ When you call the built-in format function, you pass two arguments to the function: a **numeric value** and a **format specifier**. The format specifier is a string that contains special characters specifying how the numeric value should be formatted.

```
>>> print(format(12345.6789, '.2f')) Enter  
12345.68
```

```
>>> print(format(12345.6789, '.1f')) Enter  
12345.7
```

- ▶ The **f** specifies that the data type of the number we are formatting is a **floating-point number**.

Formatting Numbers

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

```
>>> print(format(12345.6789, '.2e'))   
1.23e+04
```

- ▶ If you want the number to be formatted with **comma separators**, you can insert a comma into the format specifier.

```
>>> print(format(123456789.456, ',.2f'))   
123,456,789.46
```

- ▶ Instead of using **f** as the type designator, you can use the **% symbol** to format a floating-point number as a percentage.

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

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

- ▶ You can also use the format function to **format integers**.

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

- ▶ How do you suppress the print function's ending newline?
- ▶ How can you change the character that is automatically displayed between multiple items that are passed to the print function?
- ▶ What is the '\n' escape character?
- ▶ What does the + operator do when it is used with two strings?
- ▶ What does the statement `print(format(65.4321, '.2f'))` display?
- ▶ What does the statement `print(format(987654.129, ',.2f'))` display?

Questions



الجامعة التقنية الجنوبية المعهد التقني القرنة قسم تقنيات أنظمة الحاسوب

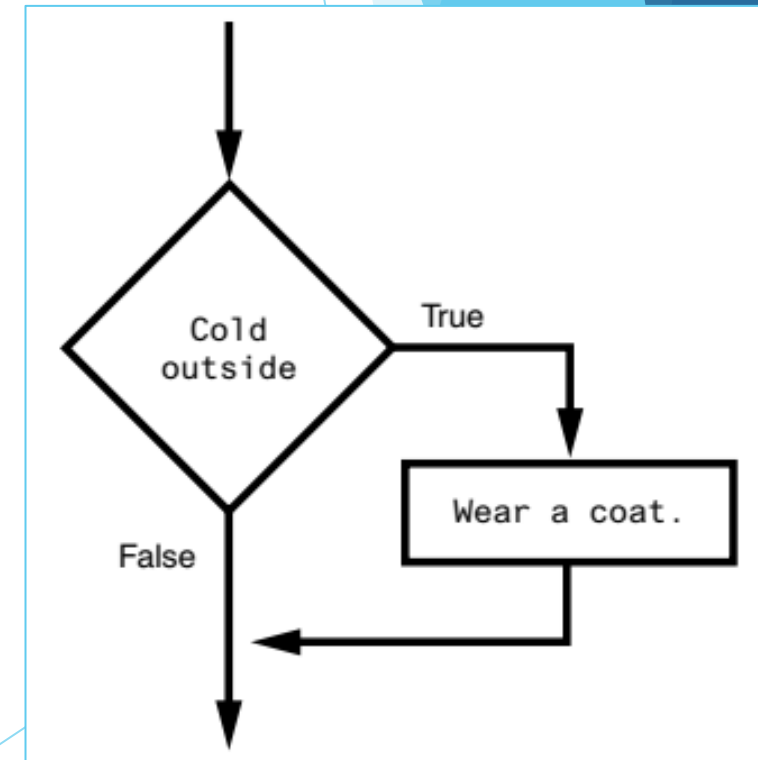


البرمجة بلغة بايثون
Python Programming

محاضرة رقم (4)

Decision Structure

- ▶ A **control structure** is a logical design that controls the order in which a set of statements execute.
- ▶ Some programs require a different type of control structure: one that can execute a set of statements only under certain circumstances. This is accomplished with a **decision structure** (**selection structure**)
- ▶ In a decision structure's simplest form, a specific action is performed only if a certain condition exists. If the condition does not exist, the action is not performed.
- ▶ Programmers call the type of decision structure shown in the Figure a single alternative decision structure. This is because it provides only one alternative path of execution.
- ▶ If the condition in the diamond symbol is true, we take the alternative path. Otherwise, we exit the structure



The if Statement

- ▶ In Python, we use the **if statement** to write a **single alternative decision structure**. Here is the general format of the if statement:

```
if condition:  
    statement  
    statement  
    etc.
```

- ▶ When the if statement executes, the condition is tested. If the condition is true, the statements that appear in the block following the if clause are executed. If the condition is false, the statements in the block are skipped.
- ▶ The expressions that are tested by the if statement are called Boolean expressions. Typically, the Boolean expression that is tested by an if statement is formed with a relational operator.
- ▶ A relational operator determines whether a specific relationship exists between two values. For example, the greater than operator (>) determines whether one value is greater than another.

Boolean Expressions and Relational Operators

- ▶ Tables below show the relational operators that are available in Python and examples of several Boolean expressions that compare the variables x and y.

Table Relational operators	
Operator	Meaning
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

Table Boolean expressions using relational operators	
Expression	Meaning
x > y	Is x greater than y?
x < y	Is x less than y?
x >= y	Is x greater than or equal to y?
x <= y	Is x less than or equal to y?
x == y	Is x equal to y?
x != y	Is x not equal to y?

- ▶ Two of the operators, >= and <=, test for more than one relationship. They determine whether the operand on its left is greater than (less than) or equal to the operand on its right.

- ▶ What is a control structure?
- ▶ What is a decision structure?
- ▶ What is a single alternative decision structure?
- ▶ What is a Boolean expression?
- ▶ What types of relationships between values can you test with relational operators?
- ▶ Write an if statement that assigns 0 to x if y is equal to 20.
- ▶ Write an if statement that assigns 0.2 to commissionRate if sales is greater than or equal to 10000.

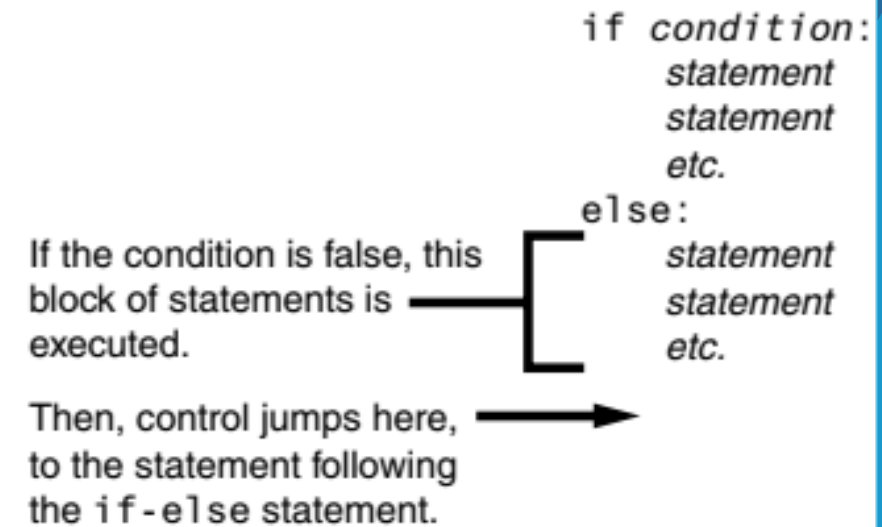
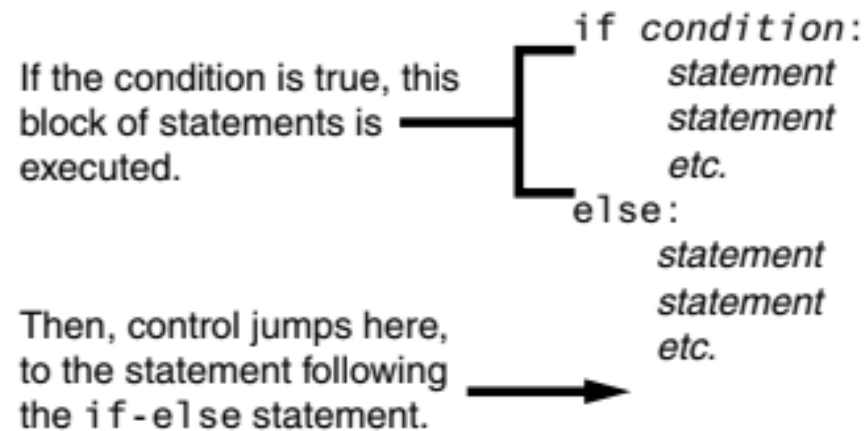
Questions

The if-else Statement

- ▶ Now, we will look at the **dual alternative decision structure**, which has two possible paths of execution—one path is taken if a condition is true, and the other path is taken if the condition is false.
- ▶ In code, we write a dual alternative decision structure as an **if-else statement**. Here is the general format of the if-else statement:

```
if condition:  
    statement  
    statement  
    etc.  
else:  
    statement  
    statement  
    etc.
```

Figure Conditional execution in an if-else statement



- ▶ How does a dual alternative decision structure work?
- ▶ What statement do you use in Python to write a dual alternative decision structure?
- ▶ When you write an if-else statement, under what circumstances do the statements that appear after the else clause execute ?

Questions

Nested Decision Structures and the if-elif-else Statement

- ▶ To test more than one condition, a decision structure can be nested inside another decision structure.
- ▶ Python provides a special version of the decision structure known as the **if-elif-else statement**, which makes programs with many conditions simpler to write.
- ▶ When the statement executes, condition_1 is tested. If condition_1 is true, the block of statements that immediately follow is executed, up to the elif clause. The rest of the structure is ignored.
- ▶ If condition_1 is false, however, the program jumps to the very next elif clause and tests condition_2.
- ▶ This process continues until a condition is found to be true, or no more elif clauses are left.

```
if condition_1:  
    statement  
    statement  
    etc.  
elif condition_2:  
    statement  
    statement  
    etc.
```

Insert as many elif clauses as necessary . . .

```
else:  
    statement  
    statement  
    etc.
```

- Convert the following code to an if-elif-else statement:

```
if number == 1:  
    print('One')  
else:  
    if number == 2:  
        print('Two')  
    else:  
        if number == 3:  
            print('Three')  
        else:  
            print('Unknown')
```

Questions

Logical Operators

- ▶ Python provides a set of operators known as **logical operators**, which you can use to create **complex Boolean expressions**. Table below describes these operators.

Table Logical operators

Operator	Meaning
and	The and operator connects two Boolean expressions into one compound expression. Both subexpressions must be true for the compound expression to be true.
or	The or operator connects two Boolean expressions into one compound expression. One or both subexpressions must be true for the compound expression to be true. It is only necessary for one of the subexpressions to be true, and it does not matter which.
not	The not operator is a unary operator, meaning it works with only one operand. The operand must be a Boolean expression. The not operator reverses the truth of its operand. If it is applied to an expression that is true, the operator returns false. If it is applied to an expression that is false, the operator returns true.

Boolean Variables

- ▶ So far, we have worked with **int**, **float**, and **str** (string) variables. In addition to these data types, Python also provides a **bool data type**.
- ▶ The bool data type allows you to create variables that may reference one of two possible values: **True** or **False**. Here are examples of how we assign values to a **bool variable**:

```
hungry = True  
sleepy = False
```
- ▶ Boolean variables are most commonly used as **flags**. A flag is a variable that signals when some condition exists in the program. When the flag variable is set to False, it indicates the condition does not exist. When the flag variable is set to True, it means the condition does exist.

```
if sales >= 50000.0:  
    sales_quota_met = True  
else:  
    sales_quota_met = False
```

- ▶ Assume the variables $a = 2$, $b = 4$, and $c = 6$. Circle T or F for each of the following conditions to indicate whether its value is true or false.

$a == 4$ or $b > 2$	T	F
$6 \leq c$ and $a > 3$	T	F
$1 \neq b$ and $c \neq 3$	T	F
$a \geq -1$ or $a \leq b$	T	F
not ($a > 2$)	T	F

- ▶ Write an if statement that displays the message “The number is valid” if the value referenced by speed is within the range 0 through 200.
- ▶ Write an if statement that displays the message “The number is not valid” if the value referenced by speed is outside the range 0 through 200.

Questions



الجامعة التقنية الجنوبية المعهد التقني القرنة قسم تقنيات أنظمة الحاسوب



البرمجة بلغة بايثون
Python Programming

محاضرة رقم (5)

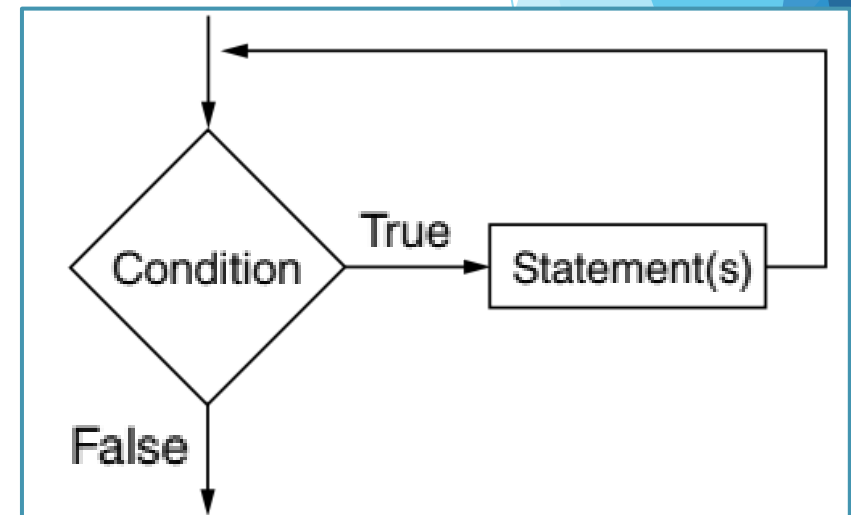
Introduction to Repetition Structures

- ▶ Programmers commonly have to write code that performs the same task over and over.
- ▶ Instead of writing the same sequence of statements over and over, a better way to repeatedly perform an operation is to write the code for the operation once, then place that code in a structure that makes the computer repeat it as many times as necessary.
- ▶ This can be done with a **repetition structure**, which is more commonly known as a **loop**.
- ▶ In this course, we will look at two broad categories of loops: **condition-controlled** and **count-controlled**.
- ▶ A condition-controlled loop uses a true/false condition to control the number of times that it repeats. A count-controlled loop repeats a specific number of times.
- ▶ In Python, you use the **while statement** to write a condition-controlled loop, and you use the **for statement** to write a count-controlled loop.

The while Loop: A Condition-Controlled Loop

- ▶ The while loop gets its name from the way it works: while a condition is true, do some task.
- ▶ The loop has two parts:
 - ▶ (1) a condition that is tested for a true or false value.
 - ▶ (2) a statement or set of statements that is repeated as long as the condition is true.
- ▶ Here is the general format of the while loop in Python:
- ▶ When the while loop executes, the condition is tested. If the condition is true, the statements that appear in the block following the while clause are executed, and the loop starts over. If the condition is false, the program exits the loop.

```
while condition:  
    statement  
    statement  
    etc.
```



The while Loop: A Condition-Controlled Loop

Program (commission.py)

```
1 # This program calculates sales commissions.
2
3 # Create a variable to control the loop.
4 keep_going = 'y'
5
6 # Calculate a series of commissions.
7 while keep_going == 'y':
8     # Get a salesperson's sales and commission rate.
9     sales = float(input('Enter the amount of sales: '))
10    comm_rate = float(input('Enter the commission rate: '))
11
12    # Calculate the commission.
13    commission = sales * comm_rate
14
```

- ▶ This program shows how we use a while loop to write the commission calculating program

```
15 # Display the commission.
16 print('The commission is $',
17       format(commission, ',.2f'), sep='')
18
19 # See if the user wants to do another one.
20 keep_going = input('Do you want to calculate another ' +
21                   'commission (Enter y for yes): ')
```

Program Output (with input shown in bold)

```
Enter the amount of sales: 10000.00 
Enter the commission rate: 0.10 
The commission is $1,000.00
Do you want to calculate another commission (Enter y for yes): y
Enter the amount of sales: 20000.00 
Enter the commission rate: 0.15 
The commission is $3,000.00
Do you want to calculate another commission (Enter y for yes): y
Enter the amount of sales: 12000.00 
Enter the commission rate: 0.10 
The commission is $1,200.00
Do you want to calculate another commission (Enter y for yes): n
```

- ▶ What is a loop iteration?
- ▶ Does the while loop test its condition before or after it performs an iteration?
- ▶ How many times will 'Hello World' be printed in the following program?

```
count = 10  
while count < 1:  
    print('Hello World')
```

Questions

The for Loop: A Count-Controlled Loop

- ▶ In Python, the **for statement** is designed to work with a sequence of data items. When the statement executes, it iterates once for each item in the sequence. Here is the general format:

```
for variable in [value1, value2, etc.]:  
    statement  
    statement  
    etc.
```

- ▶ The for statement executes in the following manner: The **variable** is assigned the **first value** in the **list**, then the statements that appear in the block are executed. Then, **variable** is assigned the **next value** in the **list**, and the statements in the block are executed again.
- ▶ This continues until **variable** has been assigned the **last value** in the **list**.
- ▶ Python programmers commonly refer to the **variable** that is used in the **for** clause as the **target variable** because it is the target of an assignment at the beginning of each loop iteration.

The for Loop: A Count-Controlled Loop

Program (simple_loop2.py)

```
1 # This program also demonstrates a simple for
2 # loop that uses a list of numbers.
3
4 print('I will display the odd numbers 1 through 9.')
5 for num in [1, 3, 5, 7, 9]:
6     print(num)
```

Program Output

```
I will display the odd numbers 1 through 9.
1
3
5
7
9
```

Program (simple_loop3.py)

```
1 # This program also demonstrates a simple for
2 # loop that uses a list of strings.
3
4 for name in ['Winken', 'Blinken', 'Nod']:
5     print(name)
```

Program Output

```
Winken
Blinken
Nod
```

Using the range Function with the for Loop

- ▶ Python provides a built-in function named **range** that simplifies the process of writing a count-controlled for **loop**.
- ▶ The **range function** creates a type of object known as an **iterable**. An **iterable** is an object that is similar to a **list**. It contains a sequence of values that can be iterated over with something like a loop.
- ▶ In the below statement, the range function will generate an iterable sequence of integers in the range of 0 up to (but not including) 5.
- ▶ If you pass two arguments to the range function, the first argument is used as the starting value of the sequence, and the second argument is used as the ending limit.
- ▶ If you pass a third argument to the range function, that argument is used as step value.

```
for num in range(5):  
    print(num)
```

```
for num in range(1, 5):  
    print(num)
```

```
for num in range(1, 10, 2):  
    print(num)
```

Using the range Function with the for Loop

Program (squares.py)

```
1 # This program uses a loop to display a
2 # table showing the numbers 1 through 10
3 # and their squares.
4
5 # Print the table headings.
6 print('Number\tSquare')
7 print('-----')
8
9 # Print the numbers 1 through 10
10 # and their squares.
11 for number in range(1, 11):
12     square = number**2
13     print(number, '\t', square)
```

Program Output

Number Square

1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

- ▶ Rewrite the following code so it calls the range function instead of using the list

```
[0, 1, 2, 3, 4, 5]:  
for x in [0, 1, 2, 3, 4, 5]:  
    print('I love to program!')
```

- ▶ What will the following code display?

```
1 for number in range(6):  
  print(number)  
2 for number in range(2, 6):  
  print(number)  
3 for number in range(0, 501, 100):  
  print(number)  
4 for number in range(10, 5, -1):  
  print(number)
```

Questions



الجامعة التقنية الجنوبية المعهد التقني القرنة قسم تقنيات أنظمة الحاسوب



البرمجة بلغة بايثون
Python Programming

محاضرة رقم (6)

Introduction to Functions

- ▶ Most programs perform tasks that are **large** enough to be broken down into **several subtasks**. For this reason, programmers usually break down their programs into small manageable pieces known as **functions**.
- ▶ A **function** is a group of statements that exist within a program for the purpose of performing a specific task.
- ▶ Instead of writing a large program as one long sequence of statements, it can be written as several small functions, each one performing a specific part of the task.
- ▶ These small functions can then be executed in the desired order to perform the overall task.
- ▶ This approach is sometimes called **divide and conquer** because a **large task** is divided into **several smaller tasks** that are easily performed.

- ▶ **Simpler Code**
- ▶ **Code Reuse**
- ▶ **Better Testing**
- ▶ **Faster Development**
- ▶ **Easier Facilitation of Teamwork**

[illegible]

```
def function1():  
    statement  
    statement  
    statement
```

function

```
def function2():  
    statement  
    statement  
    statement
```

function

```
def function3():  
    statement  
    statement  
    statement
```

function

```
def function4():  
    statement  
    statement  
    statement
```

function

Void Functions and Value-Returning Functions

- ▶ We will learn to write two types of functions: **void functions** and **value returning functions**.
- ▶ When you call a void function, it simply executes the statements it contains and then terminates.
- ▶ When you call a value-returning function, it executes the statements that it contains, then returns a value back to the statement that called it.
- ▶ The **input function** is an example of **a value-returning function**. When you call the input function, it gets the data that the user types on the keyboard and returns that data as a string.
- ▶ The **int and float functions** are also examples of **value-returning functions**. You pass an **argument** to the int function, and it returns that argument's value converted to an integer.
- ▶ The first type of function that you will learn to write is the void function.

- ▶ What is a function?
- ▶ What is meant by the phrase “divide and conquer”?
- ▶ How do functions help you reuse code in a program?
- ▶ How can functions make the development of multiple programs faster?
- ▶ How can functions make it easier for programs to be developed by teams of programmers?

Questions

Defining and Calling a Void Function

- ▶ To create a function, you write its definition. Here is the general format of a function definition in Python:

```
def function_name():  
    statement  
    statement  
    etc.
```

```
def message():  
    print('I am Arthur,')  
    print('King of the Britons.')
```

- ▶ The first line is known as the **function header**. It marks the beginning of the function definition. The function header begins with the key word **def**, followed by the name of the function, followed by a set of parentheses, followed by **a colon**.
- ▶ Beginning at the next line is a set of statements known as **a block**. A block is simply a set of statements that belong together as a group. These statements are performed any time the function is executed.
- ▶ Notice in the general format that all of the statements in the block are indented.

Defining and Calling a Void Function

- ▶ A function definition specifies what a function does, but it does not cause the function to execute. To execute a function, you must call it. This is how we would call the message function: `message()`
- ▶ When a function is called, the interpreter jumps to that function and executes its block.
- ▶ Then, when the end of the block is reached, the interpreter jumps back to the part of the program that called the function, and the program resumes execution at that point. When this happens, we say that the function returns.

Program (function_demo.py)

```
1 # This program demonstrates a function.
2 # First, we define a function named message.
3 def message():
4     print('I am Arthur,')
5     print('King of the Britons.')
6
7 # Call the message function.
8 message()
```

Program Output

```
I am Arthur,
King of the Britons.
```

Defining and Calling a Void Function

- ▶ The previous program has only one function, but it is possible to define many functions in a program.
- ▶ In fact, it is common for a program to have a main function that is called when the program starts. The main function then calls other functions in the program as they are needed.

Program

(two_functions.py)

```
1 # This program has two functions. First we
2 # define the main function.
3 def main():
4     print('I have a message for you.')
5     message()
6     print('Goodbye!')
7
8 # Next we define the message function.
9 def message():
10     print('I am Arthur,')
11     print('King of the Britons.')
```

```
12
13 # Call the main function.
14 main()
```

Program Output

```
I have a message for you.
I am Arthur,
King of the Britons.
Goodbye!
```

Indentation in Python

- ▶ In Python, each line in a block must be **indented**. As shown in Figure below, the last indented line after a function header is the last line in the function's block.

Figure All of the statements in a block are indented

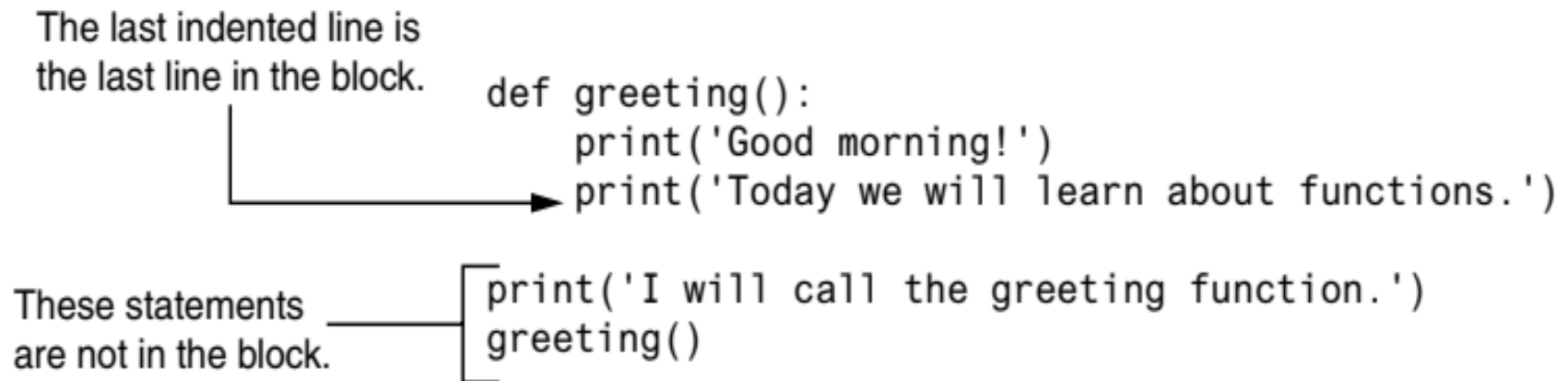
The last indented line is
the last line in the block.

```
def greeting():  
    print('Good morning!')  
    print('Today we will learn about functions.')
```

These statements
are not in the block.

```
print('I will call the greeting function.')
```

greeting()



- ▶ When you indent the lines in a block, make sure each line begins with the same number of spaces. Otherwise, an error will occur.
- ▶ Blank lines that appear in a block are ignored.

- ▶ A function definition has what two parts?
- ▶ What does the phrase “calling a function” mean?
- ▶ When a function is executing, what happens when the end of the function’s block is reached?
- ▶ Why must you indent the statements in a block?

Questions



الجامعة التقنية الجنوبية المعهد التقني القرنة قسم تقنيات أنظمة الحاسوب



البرمجة بلغة بايثون
Python Programming

محاضرة رقم (7)

Local Variables

- ▶ Anytime you assign a value to a variable inside a function, you create a **local variable**.
- ▶ A local variable belongs to the function in which it is created, and only statements inside that function can access the variable.
- ▶ An error occurs if a statement in one function tries to access a local variable that belongs to another function.
- ▶ A **variable's scope** is the part of a program in which the variable may be accessed.

Program

(bad_local.py)

```
1 # Definition of the main function.
2 def main():
3     get_name()
4     print('Hello', name)      # This causes an error!
5
6 # Definition of the get_name function.
7 def get_name():
8     name = input('Enter your name: ')
9
10 # Call the main function.
11 main()
```

Passing Arguments to Functions

- ▶ Sometimes it is useful not only to call a function, but also to send one or more pieces of data into the function. Pieces of data that are sent into a function are known as **arguments**.
- ▶ If you want a function to receive arguments when it is called, you must equip the function with one or more parameter variables. A **parameter variable**, often simply called **a parameter**, is a special variable that is assigned the value of an argument when a function is called.

```
value = 5  
show_double(value)
```

```
def show_double(number):  
    result = number * 2  
    print(result)
```

- ▶ A variable's scope is the part of a program in which the variable may be accessed. A parameter variable's scope is the function in which the parameter is used.
- ▶ All of the statements inside the function can access the parameter variable, but no statement outside the function can access it.

Passing Multiple Arguments

- ▶ Often it's useful to write functions that can accept **multiple arguments**.
- ▶ Program below shows a function named `show_sum`, that accepts two arguments. The function adds the two arguments and displays their sum.
- ▶ The arguments, 12 and 45, are passed by position to the parameter variables in the function.

Program

(multiple_args.py)

```
1 # This program demonstrates a function that accepts
2 # two arguments.
3
4 def main():
5     print('The sum of 12 and 45 is')
6     show_sum(12, 45)
7
8 # The show_sum function accepts two arguments
9 # and displays their sum.
```

```
10 def show_sum(num1, num2):
11     result = num1 + num2
12     print(result)
13
14 # Call the main function.
15 main()
```

Program Output

```
The sum of 12 and 45 is
57
```

- ▶ What is a local variable? How is access to a local variable restricted?
- ▶ What is a variable's scope?
- ▶ Is it permissible for a local variable in one function to have the same name as a local variable in a different function?
- ▶ What are the pieces of data that are passed into a function called?
- ▶ What are the variables that receive pieces of data in a function called?

Questions

Global Variables and Global Constants

- ▶ When a variable is created by an assignment statement that is written outside all the functions in a program file, the variable is global. A **global variable** can be accessed by any statement in the program file, including the statements in any function.
- ▶ A global constant is a global name that references a value that cannot be changed.

Program (global2.py)

```
1 # Create a global variable.
2 number = 0
3
4 def main():
5     global number
6     number = int(input('Enter a number: '))
7     show_number()
8
9 def show_number():
```

```
10     print('The number you entered is', number)
11
12 # Call the main function.
13 main()
```

Program Output

```
Enter a number: 55 
The number you entered is 55
```

Introduction to Value-Returning Functions

- ▶ You have learned about void functions. A void function is a group of statements that exist within a program for the purpose of performing a specific task. When the function is finished, control of the program returns to the statement appearing immediately after the function call.
- ▶ A **value-returning function** is a special type of function. It is a group of statements that perform a specific task. When you want to execute the function, you call it.
- ▶ When a value-returning function finishes, however, it returns a value back to the part of the program that called it. The value that is returned from a function can be used like any other value.
- ▶ Python, as well as most programming languages, comes with a standard library of functions that have already been written for you.
- ▶ These functions, known as **library functions**, make a programmer's job easier because they perform many of the tasks that programmers commonly need to perform.

Standard Library Functions and the import Statement

- ▶ In fact, you have already used several of Python's library functions. Some of the functions that you have used are `print`, `input`, and `range`. Python has many other library functions.
- ▶ Some of Python's library functions are built into the Python interpreter. This is the case with the `print`, `input`, `range`, and other functions about which you have already learned.
- ▶ Many of the functions in the standard library, however, are stored in files that are known as **modules**.
- ▶ These modules help organize the standard library functions. For example, functions for performing math operations are stored together in a module, functions for working with files are stored together in another module, and so on.
- ▶ In order to call a function that is stored in a module, you have to write an import statement at the top of your program. An **import statement** tells the interpreter the name of the module that contains the function.

Generating Random Numbers

- ▶ **Random numbers** are useful for lots of different programming tasks. Python provides several library functions for working with random numbers. These functions are stored in a module named **random**.
- ▶ To use these functions, you first need to write import random statement at the top of your program.
- ▶ The first random-number generating function that we will discuss is named **randint**.

Program (random_numbers.py)

```
1 # This program displays a random number
2 # in the range of 1 through 10.
3 import random
4
5 def main():
6     # Get a random number.
7     number = random.randint(1, 10)
```

```
8     # Display the number.
9     print('The number is', number)
10
11 # Call the main function.
12 main()
```

Program Output

The number is 7

- ▶ What is the scope of a global variable?
- ▶ How does a value-returning function differ from the void functions?
- ▶ What is a library function?
- ▶ Why are library functions like “black boxes”?
- ▶ What does the following statement do?

```
x = random.randint(1, 100)
```

- ▶ What does the following statement do?

```
print(random.randint(1, 20))
```

Questions



الجامعة التقنية الجنوبية المعهد التقني القرنة قسم تقنيات أنظمة الحاسوب



البرمجة بلغة بايثون
Python Programming

محاضرة رقم (8)

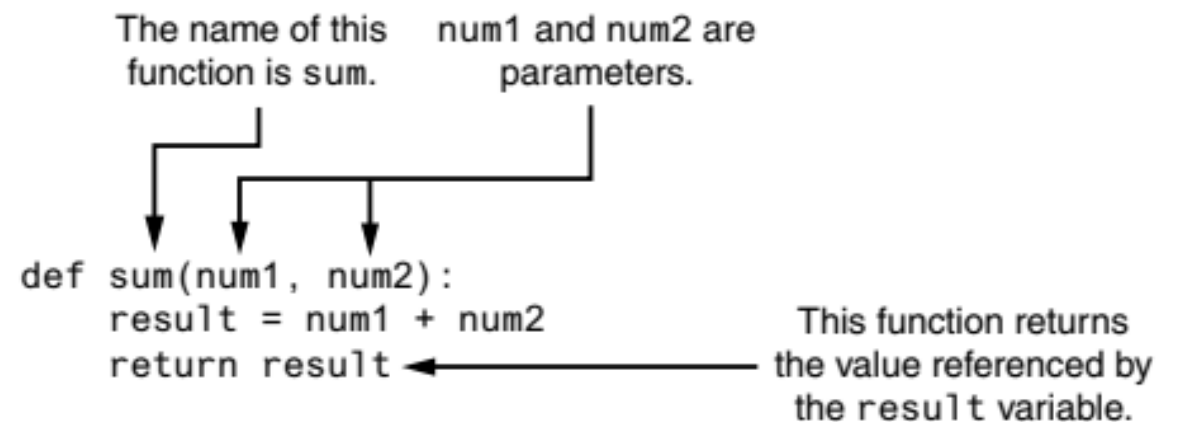
Writing Your Own Value-Returning Functions

- ▶ You write a value-returning function in the same way that you write a void function, with one exception: a value-returning function must have a **return statement**.
- ▶ Here is the general format of a value-returning function definition:

```
def function_name():  
    statement  
    statement  
    etc.  
    return expression
```

Figure

Parts of the function



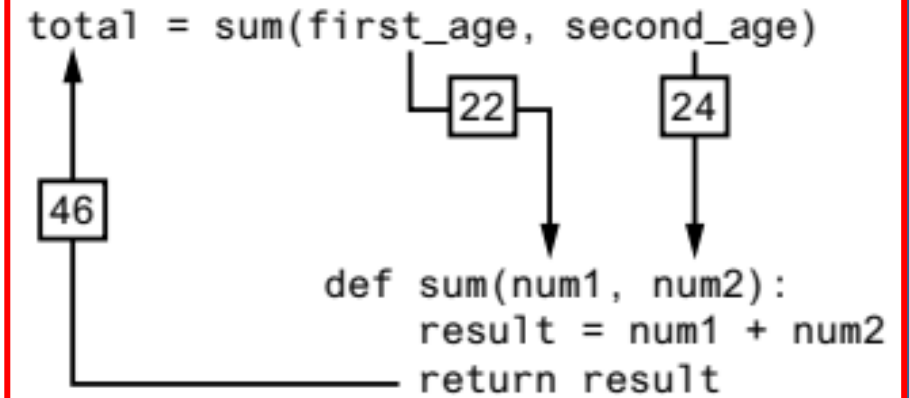
- ▶ The value of the expression that follows the key word return will be sent back to the part of the program that called the function. This can be any value, variable, or expression that has a value.

Writing Your Own Value-Returning Functions

Program

(total_ages.py)

```
1  # This program uses the return value of a function.
2
3  def main():
4      # Get the user's age.
5      first_age = int(input('Enter your age: '))
6
7      # Get the user's best friend's age.
8      second_age = int(input("Enter your best friend's age: "))
9
10     # Get the sum of both ages.
11     total = sum(first_age, second_age)
12
13     # Display the total age.
14     print('Together you are', total, 'years old.')
15
16     # The sum function accepts two numeric arguments and
17     # returns the sum of those arguments.
```



```
18 def sum(num1, num2):
19     result = num1 + num2
20     return result
21
22 # Call the main function.
23 main()
```

Program Output (with input shown in bold)

```
Enter your age: 22 
Enter your best friend's age: 24 
Together you are 46 years old.
```

Returning Strings and Boolean Values

- ▶ So far, you've seen examples of functions that return numbers. You can also write functions that return strings. For example, the following function prompts the user to enter his or her name, then returns the string that the user entered.
- ▶ Python allows you to write **Boolean functions**, which return either True or False. For example, suppose you are designing a program that will ask the user to enter a number, then determine whether that number is even or odd.

```
def get_name():  
    # Get the user's name.  
    name = input('Enter your name: ')  
    # Return the name.  
    return name
```

```
def is_even(number):  
    # Determine whether number is even. If it is,  
    # set status to true. Otherwise, set status  
    # to false.  
    if (number % 2) == 0:  
        status = True  
    else:  
        status = False  
    # Return the value of the status variable.  
    return status
```

Returning Multiple Values

- ▶ The examples of value-returning functions that we have looked at so far return a single value. In Python, however, you are not limited to returning only one value.
- ▶ You can specify **multiple expressions** separated by commas after the return statement, as shown in this general format: `return expression1, expression2, etc.`
- ▶ As an example, look at the following definition for a function named `get_name`. The function prompts the user to enter his or her first and last names. These names are stored in two local variables: `first` and `last`. The return statement returns both variables.

```
first_name, last_name = get_name()
```

```
def get_name():  
    # Get the user's first and last names.  
    first = input('Enter your first name: ')  
    last = input('Enter your last name: ')  
    # Return both names.  
    return first, last
```

- ▶ What is the purpose of the return statement in a function?
- ▶ Look at the following function definition:

```
def do_something(number):  
    return number * 2
```

- ▶ a. What is the name of the function?
- ▶ b. What does the function do?
- ▶ c. Given the function definition, what will the following statement display? `print(do_something(10))`

Questions

The math Module

- ▶ The **math module** in the Python standard library contains several functions that are useful for performing mathematical operations. Table below lists many of the functions in the math module.

Table Many of the functions in the math module

math Module Function	Description
<code>acos(x)</code>	Returns the arc cosine of x, in radians.
<code>asin(x)</code>	Returns the arc sine of x, in radians.
<code>atan(x)</code>	Returns the arc tangent of x, in radians.
<code>ceil(x)</code>	Returns the smallest integer that is greater than or equal to x.
<code>cos(x)</code>	Returns the cosine of x in radians.
<code>degrees(x)</code>	Assuming x is an angle in radians, the function returns the angle converted to degrees.
<code>exp(x)</code>	Returns e^x

The math Module

Table Many of the functions in the math module

math Module Function	Description
<code>floor(x)</code>	Returns the largest integer that is less than or equal to x .
<code>hypot(x, y)</code>	Returns the length of a hypotenuse that extends from $(0, 0)$ to (x, y) .
<code>log(x)</code>	Returns the natural logarithm of x .
<code>log10(x)</code>	Returns the base-10 logarithm of x .
<code>radians(x)</code>	Assuming x is an angle in degrees, the function returns the angle converted to radians.
<code>sin(x)</code>	Returns the sine of x in radians.
<code>sqrt(x)</code>	Returns the square root of x .
<code>tan(x)</code>	Returns the tangent of x in radians.

The math Module

- ▶ These functions typically accept one or more values as arguments, perform a mathematical operation using the arguments, and return the result.
- ▶ All of the functions listed in the previous Table return a **float value**, **except** the **ceil and floor functions**, which return int values.
- ▶ For example, one of the functions is named **sqrt**. The sqrt function accepts an argument and returns the square root of the argument.

```
result = math.sqrt(16)
```
- ▶ The math module defines two variables, **pi** and **e**, which are assigned mathematical values for pi and e.
- ▶ You can use these variables in equations that require their values. For example, the following statement, which calculates the area of a circle, uses pi.

```
area = math.pi * radius**2
```

The math Module

Program (square_root.py)

```
1  # This program demonstrates the sqrt function.
2  import math
3
4  def main():
5      # Get a number.
6      number = float(input('Enter a number: '))
7
8      # Get the square root of the number.
9      square_root = math.sqrt(number)
10
11     # Display the square root.
12     print('The square root of', number, 'is', square_root)
13
14     # Call the main function.
15     main()
```

Program Output (with input shown in bold)

Enter a number: **25** **Enter**
The square root of 25.0 is 5.0

- ▶ What import statement do you need to write in a program that uses the math module?
- ▶ Write a statement that uses a math module function to get the square root of 100 and assigns it to a variable.
- ▶ Write a statement that uses a math module function to convert 45 degrees to radians and assigns the value to a variable.

Questions



الجامعة التقنية الجنوبية المعهد التقني القرنة قسم تقنيات أنظمة الحاسوب



البرمجة بلغة بايثون
Python Programming

محاضرة رقم (9)

Sequences

- ▶ A **sequence** is an object that contains multiple items of data. The items that are in a sequence are stored one after the other.
- ▶ Python provides various ways to perform operations on the items that are stored in a sequence. You can perform operations on a sequence to examine and manipulate the items stored in it.
- ▶ There are several different types of sequence objects in Python. In this course, we will look at two of the fundamental sequence types: **lists** and **tuples**.
- ▶ Both lists and tuples are sequences that can hold various types of data.
- ▶ The difference between lists and tuples is simple: a **list** is **mutable**, which means that a program can change its contents, but a **tuple** is **immutable**, which means that once it is created, its contents cannot be changed.
- ▶ We will explore some of the operations that you may perform on these sequences.

Introduction to Lists

- ▶ A **list** is an object that contains multiple data items. Each item that is stored in a list is called an element.
- ▶ Here is a statement that creates a list of integers: `numbers = [2, 4, 6, 8, 10]`
- ▶ The items that are enclosed in brackets and separated by commas are the list elements.
- ▶ A list can hold items of different types, as shown here: `info = ['Alicia', 27, 1550.87]`
- ▶ You can use the print function to display an entire list, as shown here: `print(numbers)`
- ▶ Python also has a built-in list() function that can convert certain types of objects to lists.
- ▶ Here is an example: `numbers = list(range(1, 10, 2))`
- ▶ Recall from Week 5 that when you pass three arguments to the **range function**, the first argument is the starting value, the second argument is the ending limit, and the third argument is the step value.
- ▶ This statement will assign the list [1, 3, 5, 7, 9] to the numbers variable.

The Repetition Operator and len Function

- ▶ You learned in Week 3 that the ***** symbol multiplies two numbers. However, when the operand on the left side of the ***** symbol is a sequence (such as a list) and the operand on the right side is an integer, it becomes the **repetition operator**.
- ▶ The repetition operator makes multiple copies of a list and joins them all together.

```
1 >>> numbers = [0] * 5   
2 >>> print(numbers)   
3 [0, 0, 0, 0, 0]
```

```
1 >>> numbers = [1, 2, 3] * 3   
2 >>> print(numbers)   
3 [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

- ▶ You can iterate over a list with the for loop. Also, Python has a built-in function named **len** that returns the length of a sequence, such as a list.

```
numbers = [99, 100, 101, 102]  
for n in numbers:  
    print(n)
```

```
my_list = [10, 20, 30, 40]  
size = len(my_list)
```

Indexing

- ▶ Another way that you can access the individual elements in a list is with an **index**. Each element in a list has an index that specifies its position in the list.
- ▶ Indexing starts at **0**, so the index of the first element is 0, the index of the second element is 1, and so forth. The index of the last element in a list is 1 less than the number of elements in the list.
- ▶ For example, the following statement creates a list with 4 elements: `my_list = [10, 20, 30, 40]`
- ▶ The indexes of the elements in this list are 0, 1, 2, and 3. We can print the elements of the list with the following statement: `print(my_list[0], my_list[1], my_list[2], my_list[3])`
- ▶ You can also use **negative indexes** with lists to identify element positions relative to the end of the list. The index -1 identifies the last element in a list, -2 identifies the next to last element, and so forth.

```
print(my_list[-1], my_list[-2], my_list[-3], my_list[-4])
```

40	30	20	10
----	----	----	----

Changing Lists and Concatenating Lists

- ▶ Lists in Python are mutable, which means their elements can be changed. Consequently, an expression in the form `list[index]` can appear on the left side of an **assignment operator**.

```
1 numbers = [1, 2, 3, 4, 5]
2 print(numbers)
3 numbers[0] = 99
4 print(numbers)
```

```
[1, 2, 3, 4, 5]
```

```
[99, 2, 3, 4, 5]
```

- ▶ To **concatenate** means to join two things together. You can use the **+ operator** to concatenate two lists. Below is an example:

- ▶ You can also use the **+= augmented assignment operator** to concatenate one list to another. Below is an example:

```
list1 = [1, 2, 3, 4]
list2 = [5, 6, 7, 8]
list3 = list1 + list2
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

```
list1 = [1, 2, 3, 4]
list2 = [5, 6, 7, 8]
list1 += list2
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

- ▶ What will the following code display?

1

```
numbers = [1, 2, 3, 4, 5]  
numbers[2] = 99  
print(numbers)
```

2

```
numbers = list(range(3))  
print(numbers)
```

3

```
numbers = [10] * 5  
print(numbers)
```

4

```
numbers = list(range(1, 10, 2))  
for n in numbers:  
    print(n)
```

5

```
numbers = [1, 2, 3, 4, 5]  
print(numbers[-2])
```

- ▶ How do you find the number of elements in a list?

Questions

- What will the following code display?

1

```
numbers1 = [1, 2, 3]
numbers2 = [10, 20, 30]
numbers3 = numbers1 + numbers2
print(numbers1)
print(numbers2)
print(numbers3)
```

2

```
numbers1 = [1, 2, 3]
numbers2 = [10, 20, 30]
numbers2 += numbers1
print(numbers1)
print(numbers2)
```

Questions



الجامعة التقنية الجنوبية المعهد التقني القرنة قسم تقنيات أنظمة الحاسوب



البرمجة بلغة بايثون
Python Programming

محاضرة رقم (10)

List Slicing

- ▶ You have seen how indexing allows you to select a specific element in a sequence.
- ▶ Sometimes you want to select more than one element from a sequence. In Python, you can write expressions that select subsections of a sequence, known as **slices**.
- ▶ A **slice** is a span of items that are taken from a sequence. When you take a slice from a list, you get a span of elements from within the list.
- ▶ To get a slice of a list, you write an expression in the following format: `list_name[start : end]`
- ▶ In the format, **start** is the index of the **first element** in the slice, and **end** is the index marking the **end of the slice**. The expression returns a list containing a copy of the elements from start up to (but not including) end.

```
1 >>> numbers = [1, 2, 3, 4, 5]   
2 >>> print(numbers)   
3 [1, 2, 3, 4, 5]
```

```
4 >>> print(numbers[1:3])   
5 [2, 3]  
6 >>>
```

List Slicing

- ▶ If you leave out the start index in a slicing expression, Python uses 0 as the starting index.

<pre>1 >>> numbers = [1, 2, 3, 4, 5] <input type="button" value="Enter"/> 2 >>> print(numbers) <input type="button" value="Enter"/> 3 [1, 2, 3, 4, 5]</pre>	<pre>4 >>> print(numbers[:3]) <input type="button" value="Enter"/> 5 [1, 2, 3] 6 >>></pre>
---	--

- ▶ If you leave out the end index in a slicing expression, Python uses the length of the list as the end index.

<pre>1 >>> numbers = [1, 2, 3, 4, 5] <input type="button" value="Enter"/> 2 >>> print(numbers) <input type="button" value="Enter"/> 3 [1, 2, 3, 4, 5]</pre>	<pre>4 >>> print(numbers[2:]) <input type="button" value="Enter"/> 5 [3, 4, 5] 6 >>></pre>
---	--

- ▶ If you leave out both the start and end index in a slicing expression, you get a copy of the entire list.

<pre>1 >>> numbers = [1, 2, 3, 4, 5] <input type="button" value="Enter"/> 2 >>> print(numbers) <input type="button" value="Enter"/> 3 [1, 2, 3, 4, 5]</pre>	<pre>4 >>> print(numbers[:]) <input type="button" value="Enter"/> 5 [1, 2, 3, 4, 5] 6 >>></pre>
---	---

List Slicing

- ▶ The slicing examples we have seen so far get slices of consecutive elements from lists.
- ▶ Slicing expressions can also have step value, which can cause elements to be skipped in the list.
- ▶ The following interactive mode session shows an example of a slicing expression with a step value:

```
1 >>> numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] Enter
2 >>> print(numbers) Enter
3 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
4 >>> print(numbers[1:8:2]) Enter
5 [2, 4, 6, 8]
6 >>>
```

- ▶ You can also use negative numbers as indexes in slicing expressions to reference positions relative to the end of the list. Python adds a negative index to the length of a list to get the position referenced by that index.

```
1 >>> numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] Enter
2 >>> print(numbers) Enter
3 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
4 >>> print(numbers[-5:]) Enter
5 [6, 7, 8, 9, 10]
6 >>>
```

► What will the following code display?

1

```
numbers = [1, 2, 3, 4, 5]
my_list = numbers[1:3]
print(my_list)
```

2

```
numbers = [1, 2, 3, 4, 5]
my_list = numbers[1:]
print(my_list)
```

3

```
numbers = [1, 2, 3, 4, 5]
my_list = numbers[:1]
print(my_list)
```

4

```
numbers = [1, 2, 3, 4, 5]
my_list = numbers[:]
print(my_list)
```

5

```
numbers = [1, 2, 3, 4, 5]
my_list = numbers[-3:]
print(my_list)
```

Questions

Finding Items in Lists with the in Operator

- ▶ In Python, you can use the **in operator** to determine whether an item is contained in a list. Here is the format of an expression written with the in operator to search for an item in a list: `item in list`
- ▶ In the general format, item is the item for which you are searching, and list is a list. The expression returns true if item is found in the list, or false otherwise.

Program (in_list.py)

```
1  # This program demonstrates the in operator
2  # used with a list.
3
4  def main():
5      # Create a list of product numbers.
6      prod_nums = ['V475', 'F987', 'Q143', 'R688']
7
8      # Get a product number to search for.
9      search = input('Enter a product number: ')
10
11
12     if search in prod_nums:
13         print(search, 'was found in the list.')
14     else:
15         print(search, 'was not found in the list.')
16
17 # Call the main function.
18 main()
```

List Methods and Useful Built-in Functions

- ▶ Lists have numerous methods that allow you to add elements, remove elements, change the ordering of elements, and so forth. We will look at a few of these methods, which are listed in next Table .

Table 7-1 A few of the list methods

Method	Description
<code>append(<i>item</i>)</code>	Adds <i>item</i> to the end of the list.
<code>index(<i>item</i>)</code>	Returns the index of the first element whose value is equal to <i>item</i> . A <code>ValueError</code> exception is raised if <i>item</i> is not found in the list.
<code>insert(<i>index</i>, <i>item</i>)</code>	Inserts <i>item</i> into the list at the specified <i>index</i> . When an item is inserted into a list, the list is expanded in size to accommodate the new item. The item that was previously at the specified index, and all the items after it, are shifted by one position toward the end of the list. No exceptions will occur if you specify an invalid index. If you specify an index beyond the end of the list, the item will be added to the end of the list. If you use a negative index that specifies an invalid position, the item will be inserted at the beginning of the list.
<code>sort()</code>	Sorts the items in the list so they appear in ascending order (from the lowest value to the highest value).
<code>remove(<i>item</i>)</code>	Removes the first occurrence of <i>item</i> from the list. A <code>ValueError</code> exception is raised if <i>item</i> is not found in the list.
<code>reverse()</code>	Reverses the order of the items in the list.

The insert Method

Program

(insert_list.py)

```
1  # This program demonstrates the insert method.
2
3  def main():
4      # Create a list with some names.
5      names = ['James', 'Kathryn', 'Bill']
6
7      # Display the list.
8      print('The list before the insert:')
9      print(names)
10
11     # Insert a new name at element 0.
12     names.insert(0, 'Joe')
13
```

```
14     # Display the list again.
15     print('The list after the insert:')
16     print(names)
17
18     # Call the main function.
19     main()
```

Program Output

The list before the insert:

```
['James', 'Kathryn', 'Bill']
```

The list after the insert:

```
['Joe', 'James', 'Kathryn', 'Bill']
```

The index Method

Program

(index_list.py)

```
1  # This program demonstrates how to get the
2  # index of an item in a list and then replace
3  # that item with a new item.
4
5  def main():
6      # Create a list with some items.
7      food = ['Pizza', 'Burgers', 'Chips']
8
9      # Display the list.
10     print('Here are the items in the food list:')
11     print(food)
12
13     # Get the item to change.
14     item = input('Which item should I change? ')
15
16     try:
17         # Get the item's index in the list.
```

```
18         item_index = food.index(item)
19
20         # Get the value to replace it with.
21         new_item = input('Enter the new value: ')
22
23         # Replace the old item with the new item.
24         food[item_index] = new_item
25
26         # Display the list.
27         print('Here is the revised list:')
28         print(food)
29     except ValueError:
30         print('That item was not found in the list.')
31
32     # Call the main function.
33     main()
```

- ▶ What will the following code display?

```
names = ['Jim', 'Jill', 'John', 'Jasmine']  
if 'Jasmine' not in names:  
    print('Cannot find Jasmine.')  
else:  
    print("Jasmine's family:")  
    print(names)
```

- ▶ Assume the following statement appears in a program:

```
names = []
```

Which of the following statements would you use to add the string 'Wendy' to the list at index 0? Why would you select this statement instead of the other?

- a. `names[0] = 'Wendy'`
- b. `names.append('Wendy')`

Questions