Chapter 18
Network Programming and Automation

New protocols, technologies, delivery, and operation and maintenance models continue to emerge in the field of network engineering. Traditional networks are challenged by new connectivity needs such as cloud computing and artificial intelligence. Meanwhile, enterprises are also constantly seeking convenience, flexibility and resiliency in their business. Against this backdrop, network automation is becoming increasingly imperative.

Network programming and automation aim to simplify network engineers' network configuration, management, monitoring, operations and other tasks, and improve the efficiency of their deployment, operation and maintenance. This chapter is designed to guide network engineers through an initial understanding of how to use Python programming to enable network automation.

18.1 Introduction to Network Programming and Automation

The following are some classic network operation and maintenance scenarios that you may have encountered in your work.

1. Device upgrade: there are thousands of network devices in the network, and you need to upgrade the devices in a regular basis and in bulk.
2. Configuration audits: enterprises need to conduct configuration audits on devices, for example, requiring all devices to be enabled with STelnet function and all Ethernet switches to be configured with Spanning Tree security. You need to quickly identify devices that do not meet the requirements.
3. Configuration changes: because of the network security requirements, the accounts and passwords of devices need to be changed every 3 months. You need to delete the original accounts and create new ones on thousands of network devices.

Huawei Technologies Co., Ltd., *Data Communications and Network Technologies*,
https://doi.org/10.1007/978-981-19-3029-4_18

Traditional network operation and maintenance require network engineers to manually login to the network devices, manually view and execute configuration commands, and screen the configuration results with their bare eyes. This working method that heavily relies on "humans" is an enduring and inefficient operation process, and the operation process is not audited.

Network automation, that is, using tools for automatic network deployment, operation and maintenance, can gradually reduce the reliance on "humans". This can be a desirable solution to the traditional network operation and maintenance problems. There are also many open-source tools for network automation, such as Ansible, SaltStack, Puppet, and Chef.

From the perspective of building network engineering capabilities, it is more recommended that engineers have programming skills that focus on network programming. Network programming, in a broad sense, is the development of programs to send and receive information over the network. The main task of network programming is to assemble the information on the sending end through a defined protocol, and to parse the packets according to the specified protocols at the receiving end to extract the corresponding information for communication purposes. The most important work in the process is packet assembly, packet filtering, packet capture and packet analysis, and of course some processing is needed at the end. We may come into contact with five parts, that is, code, development tools, database, server setup and web design.

In recent years, with the rise of network automation technology, the abilities to program, of which Python is a major one, become a new skill requirement for network engineers. Automation scripts written in Python can perform repetitive, time-consuming and regular operations excellently.

What can network automation do? One of the most intuitive examples is the automated configuration of devices. We can break this process into two steps: writing a configuration file and writing Python code to push the configuration file to the device. This approach is easier for network engineers who are new to network programming and automation to understand. This chapter focuses on how this approach enables network automation. As shown in Fig. 18.1, first, write the script using command lines, and then pass it to the device to run via Telnet/SSH.

18.2 Overview of Programming Languages

A computer programming language is the most important tool for programming. It is a language with certain syntax rules that a computer can accept and process. Since the birth of computers, computer languages have gone through several stages, including machine languages, assembly languages and high-level languages.

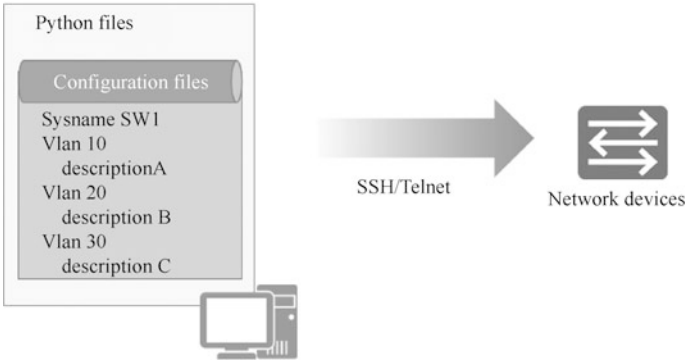
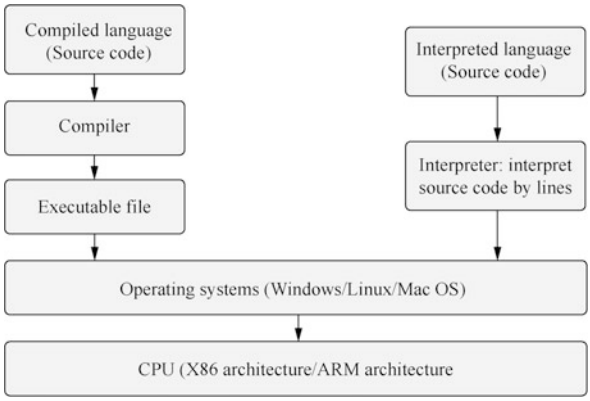


Fig. 18.1 Write a Python script to pass the configuration to the device for automated configuration

Fig. 18.2 High-level language classification



18.2.1 Classification of High-Level Programming Languages

According to whether the language needs to be compiled before execution, high-level languages can be classified into compiled languages that require compilation, and interpreted languages that do not need compilation, as shown in Fig. 18.2.

1. Compiled language

   Compiled languages have separate compilation and execution processes. Before a program is executed, a compilation process is needed to compile the source code into a binary file of machine language. The executable programs generated by compiled languages do not need recompilation to run and they directly use the compiled results, which makes them more efficient. However, executable programs are platform-specific and cannot be executed in other platforms. C, C++ and Go languages are typical compiled languages.

   Figure 18.3 shows the process of compiled language from source code to program: the source code needs to be compiled into machine instructions by compiler and assembler, and then the machine language program is generated by

link library functions of the linker. The machine language program has to match the instruction set of the CPU, and is loaded into memory by the loader when it is run. Then the CPU executes the instructions. The source code of a compiled language is compiled and converted into a format that the computer can execute, such as .exe, .dll, and .ocx.

2. Interpreted languages

   Interpreted languages do not require prior compilation, and the source code is directly interpreted into machine code, so the program can be run as long as the platform provides the corresponding interpreter. Interpreted languages require the source code to be interpreted into machine code and executed every time the program is run, which is inefficient. Both Python and Perl are typical interpreted languages.

   Figure 18.4 shows the process of interpreted languages from source code to program: the source code file (Python file) is converted into a byte code file (.pyc file) by an interpreter and then run on a Python virtual machine (Python VM, PVM).

18.2.2 Computing Technology Stack and Program Execution Process

As mentioned earlier, programming languages can be classified as machine languages, assembly languages and high-level languages. Machine languages consist of binary machine code that can be directly recognized by computers. Since machine language is obscure and difficult to understand, people have made a simple

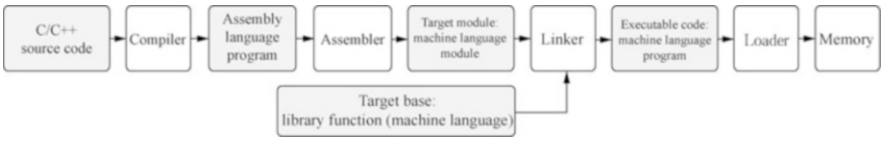
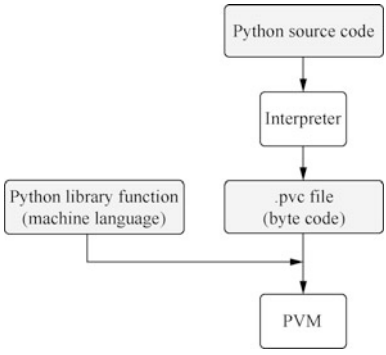


Fig. 18.3 The process of compiling a language from source code to program

Fig. 18.4 The process of interpreted languages from source code to program



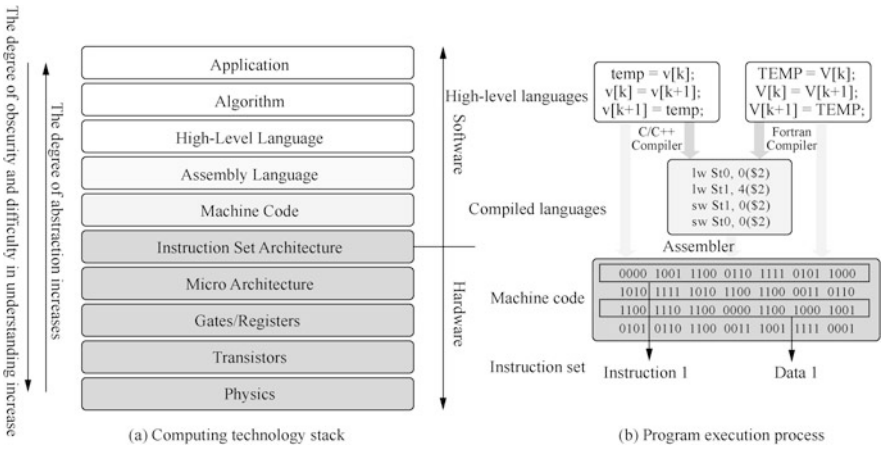


Fig. 18.5 Computing technology stack and program execution process

encapsulation of hardware instructions consisting of 0s and 1s to make them easier to be recognized and memorized by programmers (e.g., MOV, ADD), which is the assembly language. Both of these languages are low-level languages, while languages such as C, C++, Java, Python, Pascal, Lisp, Prolog and Fortran are high-level languages.

Among all programming languages, only the source programs written in machine language can be directly understood and executed by computers; programs written in other programming languages must first be "translated" into machine language programs that computers can recognize using language processing programs.

Figure 18.5 illustrates the computational technology stack and the program execution process.

In the computing stack shown in Fig. 18.5a, we can see the bottom layer of hardware, consisting of physics and transistors for implementing gate circuits and registers, which then constitute the CPU micro architecture, whose instruction set is the interface of the hardware and the software. Applications use the instructions defined in the instruction set to drive the hardware to perform the computation. Applications are usually developed using high-level languages such as C, C++, Java, Go and Python.

In the execution of a program shown in Fig. 18.5b, the high-level language first needs to be compiled into assembly language and then converted into binary machine code by the assembler according to the CPU instruction set. A program exists on a disk in the form of a stack of binary machine code consisting of instructions and data, which is also commonly referred to as a binary file.

18.3 Python Language

Python is a high-level object-oriented programming language. Programs written in Python language are cross-platform, and Python can be seen everywhere from clients, servers to Webs and mobile.

18.3.1 What Is Python

Python is an object-oriented interpreted computer programming language invented by Dutchman Guido van Rossum in 1989, and the first publicly available version of Python was released in 1991.

Python is a purely free software, whose source code and interpreter are under the GPL (GNU General Public License) protocol. Its syntax is clean and clear, with the mandatory use of White Space as a statement indent as one of its features.

Python has a rich and powerful library. It is often referred to as a glue language that can easily integrate various modules made in other languages (especially C/C++). A common application scenario is to use Python to quickly generate a prototype of a program (and sometimes even the final interface of the program) and then rewrite parts of it that have special requirements in a more appropriate programming language. For example, a graphics rendering module in a 3D game with particularly high-performance requirements can be rewritten in C/C++ and then wrapped into an extended class library that Python can call. It is important to note that you may need to consider platform issues when using extended class libraries, and some libraries may not provide cross-platform implementations.

Although Python source code files (.py) can be directly executed using Python commands, Python does not actually interpret Python source code straightly; instead, Python source code is compiled to generate Python byte code (the extensions of byte code files are generally .pyc), which is then executed by the Python Virtual Machine (PVM). The Python here is an interpreted language, meaning that it interprets the Python byte code, rather than the Python source code. The basic idea of this mechanism is the same as Java and .NET.

Although Python also has its own virtual machine, unlike the virtual machines of Java or .NET, the Python virtual machine here is a more advanced virtual machine. The "advanced" here is unlike the advanced in the usual sense, and it does not mean the Python virtual machine is more powerful than Java or .NET virtual machines. Instead, it means that Python virtual machine is farther away from the real machine than Java or .NET. Or, you could say that the Python virtual machine is a more abstract virtual machine.

The execution of the Python program source code is shown in Fig. 18.6.

1. Install Python and the runtime environment on the operating system.
2. Write the Python source code.

Fig. 18.6 Python source code execution process

3. The interpreter runs Python source code and interprets it to generate a .pyc file (byte code).
4. The Python virtual machine converts the byte code into the machine language.
5. The hardware executes the machine language.

18.3.2 Application Areas of Python

Python is a cross-platform programming language, and in theory, it can run on any operating system platform. The current common operating system platforms are Windows, Mac OS X and Linux.

Python is easy to learn, has various third-party libraries and a fast operation speed, so it has an exceptionally wide range of applications. Python's areas of application mainly include the following.

- Linux/Unix operations and maintenance.
- Command-line program development.
- GUI program development (PyQt, Kivy, etc.).
- Web program development (Django and other frameworks).
- Mobile App development (PyQt, Kivy, etc.).
- Server program development (based on protocols such as Socket)
- Web crawlers (data sources for search engines, deep learning, etc.)
- Data analysis.
- Deep learning.
- Scientific computing.

Although not all of Python's application areas are listed here, these listed areas alone cover the vast majority of the development scenarios. Readers who have used Mac OS X or Linux will find that the Python interpreter is already built into these two operating systems, which means that Python programs can be run directly on Mac OS X and Linux. Therefore, many operations and maintenance engineers are accustomed to using Python for many automated operations. For the currently popular deep learning, Python has become its first language. Therefore, from all angles, whether you are a student, a programmer, a data analyst or a scientist, you cannot live without Python. Python has become the world language of programming languages.

This chapter focuses on how to use Python for network programming and to manage network devices.

18.3.3 How Python Code Works

Python can run codes in two ways: interactive mode and scripted mode.

Programming in the interactive mode (i.e., interactive programming) does not require the creation of a script file, and code can be written by the interactive mode of the Python interpreter. Figure 18.7 illustrates the process of interactive programming on a Windows system. [Note: in Fig. 18.7, Print( ) is a built-in Python function that serves to output the contents in the parentheses.]

Code written in the scripted mode of programming (i.e., scripted programming) can be run on a variety of Python interpreters or integrated development environments. For example, IDLE, Atom, Visual Studio, Pycharm, and Anaconda that come with Python. A typical script-based programming process is shown in Fig. 18.8. First, write a Python script using Notepad software, save the script file and change its extension to .py, and then execute the script file in the Python interpreter.

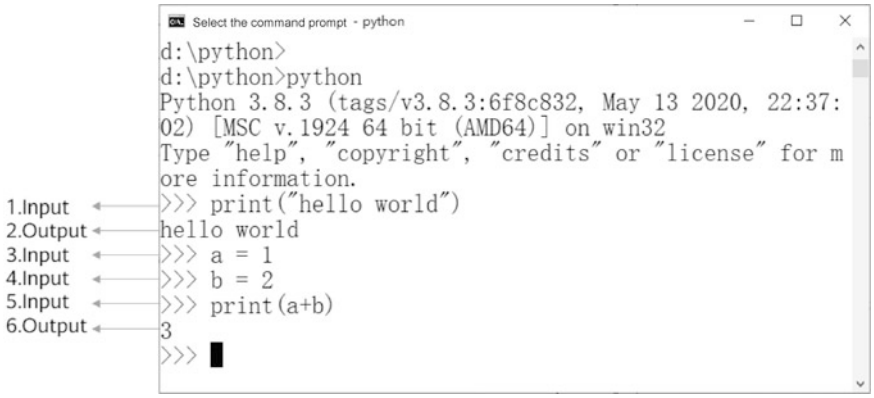


Fig. 18.7 The interactive programming process

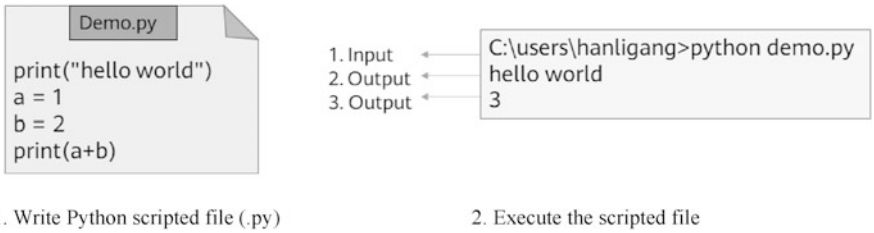


Fig. 18.8 The scripted programming process

18.3.4 Coding Specifications of Python

Coding specifications are the naming rules, code indentation, code and statement splitting and so on that should be followed when writing code in Python. Good coding specifications help improve the readability of code and make it easier to maintain and modify.

1. Suggestions for the use of semicolons, blank lines, parentheses and spaces.
   Semicolons: Python programs allow semicolons at the end of lines, but it is not suggested to use semicolons to isolate statements. It is recommended that each statement occupy a separate line. If there are multiple statements on a line, they can be separated by semicolons.
   Blank lines: blank lines can be used to separate different functions or statement blocks so as to distinguish between two pieces of code and improve the readability of the code.
   Parentheses: parentheses can be used to continue long statements, and unnecessary brackets are generally eliminated.
   Spaces: it is not recommended to use spaces within the parentheses. You can decide whether to add spaces on both sides of an operator according to your personal habits.
2. Identifier naming conventions.
   Python identifiers are commonly used to represent the names of constants, variables, functions, and other objects. Identifiers usually consist of letters, numbers and underlines, but they cannot begin with a number. Identifiers are case-sensitive and do not allow renaming. If an identifier does not conform to the naming convention, the compiler will output a SyntaxError error message when it runs the code. As shown in Fig. 18.9, the fifth identifier starts with a number, which is an incorrect identifier.
3. Code indentation.
   When writing conditional and looping statements in Python, you need to use the concept of code blocks. A code block is a set of statements that are executed when certain conditions are met. In Python programs, code indentation can be used to delimit a code block. If a code block contains two or more statements, they must have the same amount of indentation, and the Python language uses code indentation and colons to distinguish between levels of code. For Python, code indentation is a syntax rule.

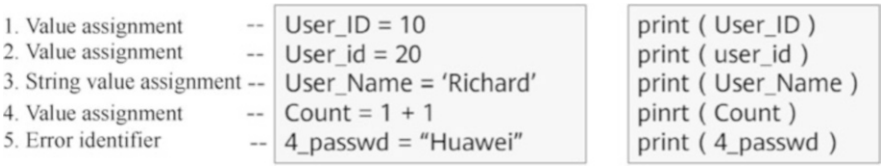


Fig. 18.9 Identifier naming specification

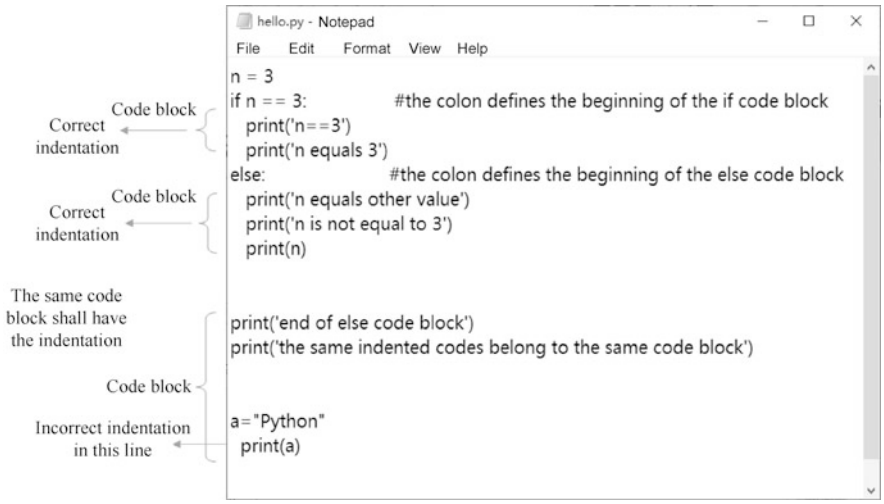


Fig. 18.10 Code blocks and indentations

When writing code, it is recommended to use four spaces for indentation. If the wrong indentation is used in the code, an IndentationError message will be returned when the program is run. The judgment statements shown in Fig. 18.10 list the beginning and end of code blocks, as well as examples of correct and incorrect indentation. The if line and the else line in the figure belong to the same code block and have the same indentation. The last line print (a) belongs to the same block as the if and else lines, so the indentation should be the same.

4. Use comments

   Comments are explanatory notes added to the program that can enhance the readability of the program. As shown in Fig. 18.11, in Python programs, comments are divided into single-line comments and multi-line comments. Single-line comments begin with # and end at the end of the line. Multi-line comments can contain multiple lines of contents that are contained within a pair of triple quotes (“‘. . .”’ or “”“. . .”””).

5. The structure of code file.

   A complete Python source code file generally contains an interpreter and encoding format declarations, documentation strings, module imports and runtime code.

   If you need to call classes from the standard library or other third-party libraries in your program, you need to first use the “import” or “from . . . import” statement to import the relevant module. The import statement is always located at the top of the file after the module comment or documentation string. Figure 18.12 is an example of the structure of a source code file.

   - The interpreter declaration is used to specify the path to the compiler that runs this file (the compiler is installed via a non-default path or there are multiple

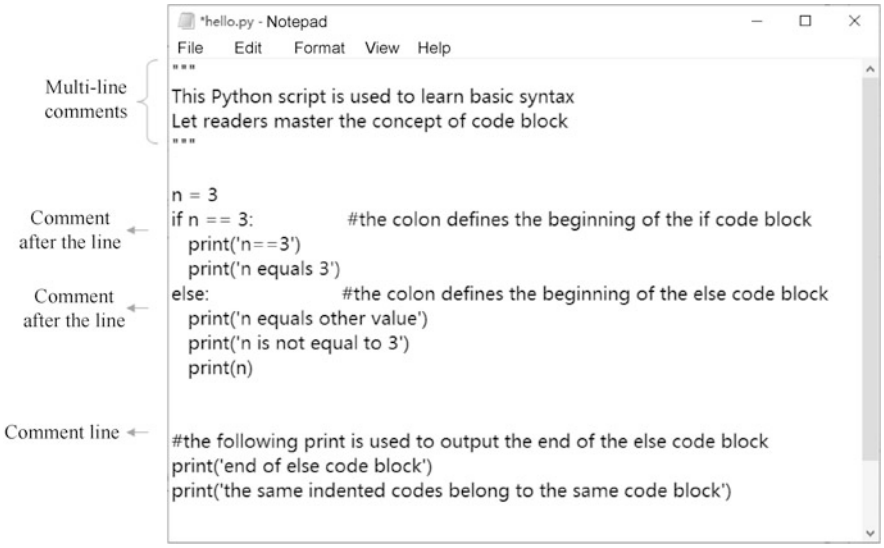


Fig. 18.11 Comments

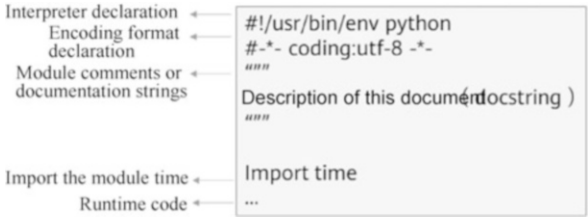


Fig. 18.12 Structure of source code file

Python compilers). The interpreter declaration in the first line of this example can be omitted on Windows operating systems.

- The encoding format declaration is used to specify the encoding type used by this program and to read the source code in the specified encoding type. Python 2 uses ASCII encoding by default (Chinese is not supported), and Python 3 supports UTF-8 encoding by default (Chinese is supported).
- The documentation string serves as a general introduction to the program's functionality.
- The module import section imports the time module, which is a built-in Python module that provides functions for handling time-related issues.

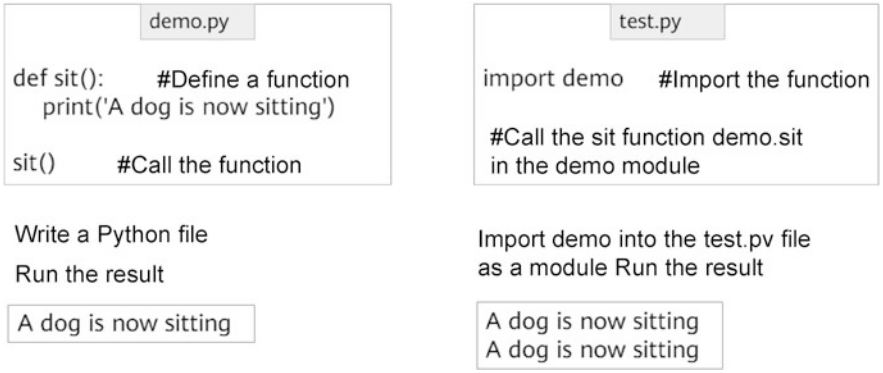


Fig. 18.13 Functions and modules

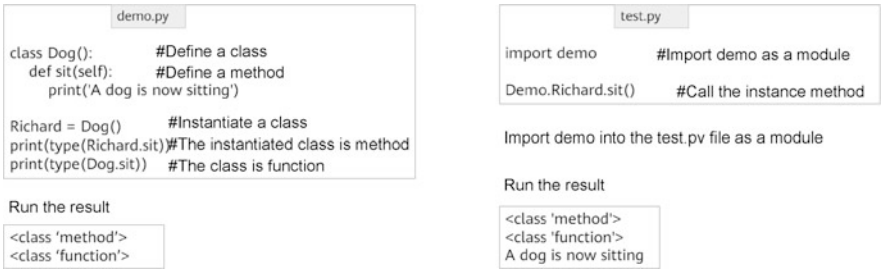


Fig. 18.14 Classes and methods

18.3.5 Basic Concepts of Python

1. Data types of Python
   The most basic data types in Python are boolean (True/False), integers, floating-point numbers and strings, and all data in Python (boolean, integers, floating point numbers, strings and even large data structures, functions and programs) exists as objects. This makes the Python language extremely uniform.
2. Functions and modules of Python
   A function is an organized, reusable piece of code that improves the modularity and code utilization of a program. Functions can be defined using the keyword def.
   A module is a saved Python file that can be composed of functions or classes. The only difference between a module and a regular Python program is that they serve different purposes. Modules can be called by other programs and therefore usually do not have a main function. The definition and call of functions and the import and call of modules are shown in Fig. 18.13.

3. Classes and methods of Python

   A class is a description of a collection that has the same properties and methods. Classes can be defined using the keyword class. The functions of an instantiated class are called methods. When defining methods, the class must carry the keyword self and it represents the created class instance itself. Figure 18.14 illustrates how a defining class and instantiable class call the instance.

18.4 Manage Network Devices with Python

18.4.1 Introduction to Telnet

Telnet defines the Network Virtual Terminal (NVT). It describes a standard representation of data and command sequences transmitted over the Internet, thus shielding differences across platforms and operating systems, such as different commands for line breaks on different platforms. In order to distinguish Telnet commands from ordinary data, Telnet uses escape sequences. Each escape sequence consists of two bytes. The first byte is 0xFF, called IAC (Interpret as Command, which is an escape character indicating that the byte following the character is the command code); and the second byte is the code of the command to be executed. Telnet can be used on Windows or Linux systems to remotely configure network devices such as Huawei routers and switches.

The telnetlib is a module in the Python standard library. It provides a class telnetlib.Telnet that implements the functions of Telnet. Table 18.1 shows the

Table 18.1 Methods of telnetlib module

Methods	Functions
Telnet.open(host,port = 0 [, timeout])	Connect to a host. The optional second argument is the port number, which defaults to the standard Telnet port (23). The optional timeout parameter specifies a timeout in seconds for blocking operations like the connection attempt (if not specified, the global default timeout setting will be used).
Telnet.read_until(expected, timeout = None)	Read until a given byte string (b”) expected is encountered or until timeout seconds have passed. When no match is found, return whatever is available instead, possibly empty bytes. Raise EOFError if the connection is closed and no cooked data is available.
Telnet.read_all( )	Read all data until EOF as bytes; block until connection closed.
Telnet.read_very_eager( )	Read everything that can be without blocking in I/O (eager), return the byte string Raise EOFError if connection closed and no cooked data available.
Telnet.write(buffer)	Write a byte string to the socket, doubling any IAC characters. This can block if the connection is blocked. May raise OSError if the connection is closed.
Telnet.close( )	Close the connection.

methods defined by the telnetlib module, where different functions can be enabled by calling different methods of the Telnet class of the telnetlib module.

18.4.2 Manage Huawei Routers with Telnet

This case will show how to import the telnetlib module using Python script file, how to configure Huawei router via Telnet and how to change the device name, create VLAN, and set the interface IP address of a Huawei router.

1. First, configure the interface IP address of the Huawei router.

```
Enter system view, return user view with Ctrl+Z.
[Huawei]interface GigabitEthernet 0/0/0
[Huawei-GigabitEthernet0/0/0]ip address 192.168.80.99 24
[Huawei-GigabitEthernet0/0/0]quit
```

2. Configure the router to allow Telnet.

```
[Huawei]user-interface vty 0 4
[Huawei-ui-vty0-4]authentication-mode password
Please configure the login password (maximum length 16):huawei@123
[Huawei-ui-vty0-4]user privilege level 15
[Huawei-ui-vty0-4]quit
```

3. Use Telnet to login to the router on Windows, and observe the interaction process.

```
C:\Users\hanlg>telnet 192.168.80.99
Login authentication
Password:
<Huawei>system-view
Enter system view, return user view with Ctrl+Z.
[Huawei]quit
```

4. Based on the Telnet interaction in the previous step, write a Python script to read the Telnet output using the telnetlib module, and then enter the Telnet command to configure the network device.

```
import telnetlib                    #Import telnetlib module
host = '192.168.80.99'              #Specify the IP address to login to the
device
password = 'huawei@123'             #Specify the password to login to the
device
tn = telnetlib.Telnet(host)         #Login to the device via Telnet
tn.read_until(b'Password:')         #Read until the echo message (the
message returned by the device) is "Password:"
```

```
 tn.write(password.encode('ascii')+b'\n')   #Enter the password of
code ASCII and enter a new line
 #Enter the system view, and change the device name
 tn.read_until(b'<Huawei>')          #Output and read until the message
"<Huawei>"
 tn.write(b'system-view'+b'\n')         #Enter the command system and
enter a new line
 tn.read_until(b'[Huawei]')          #Output and read until the message
"[Huawei]"
 tn.write(b'sysname R1'+b'\n')         #Change the router name to R1
 tn.read_until(b'[R1]')             #Read until the echo message "[R1]"
 #Create VLAN 2
 tn.write(b'vlan 2'+b'\n')           #Enter the command to create vlan 2
 tn.read_until(b'vlan2]')            #Output and read until the message
"vlan2]"
 tn.write(b'quit'+b'\n')            #Enter the command to exit the vlan2
view
 tn.read_until(b'[R1]')              #Output and read until the message
"[R1]"
 #Enter the interface view, and configure the interface IP address
 tn.write(b'interface GigabitEthernet 0/0/1'+b'\n')  #Enter the
command and enter the interface configuration mode
 tn.read_until(b'1]')             #Output and read until the message "1]"
 tn.write(b'ip address 10.1.1.1 24'+b'\n')   #Enter the command to
configure the interface IP address and subnet mask
 tn.read_until(b'1]')             #Output and read until the message "1]"
 tn.close()                    #Close the Telnet connection
```

The encode() and decode() functions of Python are used to encode and decode strings in the specified way. In this case, password.encode(' ascii') means to convert the string huawei@123 to ASCII. Here the encoding format follows the official requirements of the telnetlib module.

Python prefixes the string with a b, such as b'string', to convert the string to bytes. In this case, b'Password:' indicates that the string Password: is converted to a byte-type string. The encoding format here follows the official requirements of the telnetlib module.

18.5 Exercises

1. Python is a compiled language. ( )

   A. Correct
   B. Incorrect

2. Briefly describe the naming specification for Python identifiers.