



Modules and Libraries

Python Fundamentals

Name of presenter

Date

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Welcome to Modules and Libraries.

What you will learn

At the core of the lesson

You will learn how to:

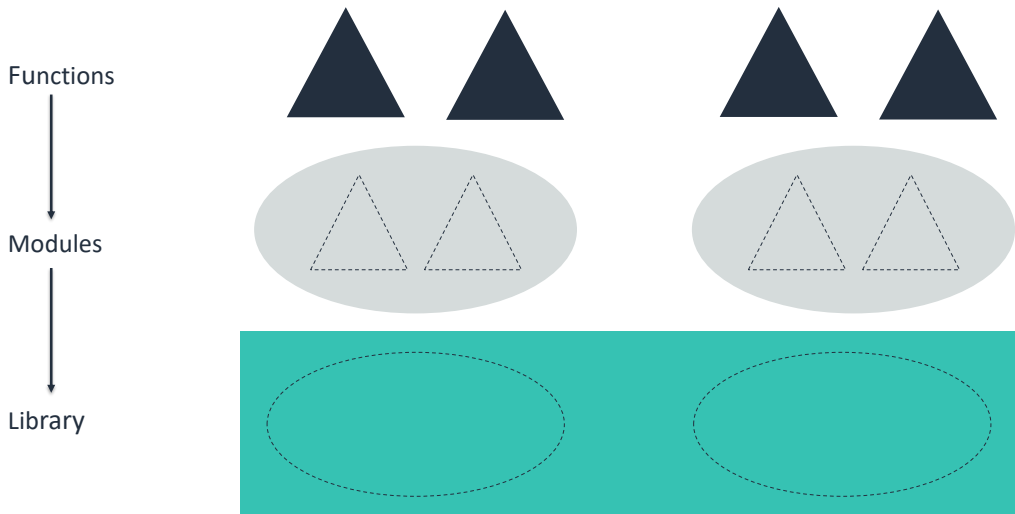
- Explain the purpose of a module in Python
- Describe the Python standard library
- Import modules from a library
- Use file handlers to open, read, write, and close files from within Python code
- Use exception handlers to catch errors in code
- Import the JavaScript Object Notation (JSON) module and use JSON functions
- Use pip to download and install third-party modules for Python



In this module, you will learn how to:

- Explain the purpose of a module in Python
- Describe the Python standard library
- Import modules from a library
- Use file handlers to open, read, write, and close files from within Python code
- Use exception handlers to catch errors in code
- Import the JavaScript Object Notation (JSON) module and use JSON functions
- Use pip to download and install third-party modules for Python

What are modules and libraries?



Functions are blocks of code that do specific tasks.

Functions can be put into modules. Modules are separate Python files that can be imported into other Python applications. Importing modules makes it possible to reuse code.

Libraries are collections of modules. By putting modules into libraries, it is possible to import a large amount of programming capability quickly.

Standard library

The Python standard library is a collection of script modules that a Python program can access. It simplifies the programming process and reduces the need to rewrite commonly used commands.

Python libraries can also consist of modules that are written in C.

Practical example: Saanvi Sarkar did not invent the wheel. Instead, she imported a wheel module from a library to help create something new—a car.

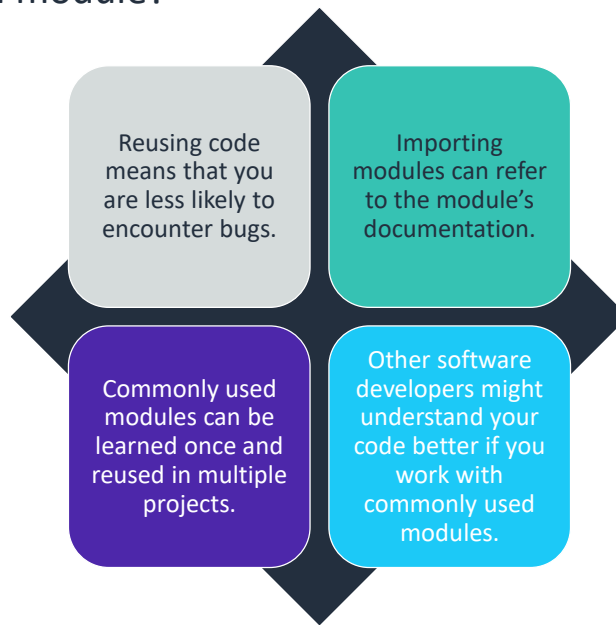
Navigating the Python standard library

The Python standard library is a collection of modules that make programming easier by removing the need to rewrite commonly used commands

The library contains built-in modules to access such functionality as time (time), getting system information (sys), querying the operating system (os), and many more.

Using the standard library where possible makes code easier to maintain and port to other platforms.

Why import a module?



Module types

Modules can be:

1. Created by you. You might need Python to complete a specific set of tasks or functions that are grouped together. If so, it can be easier to bind them together into a module.
2. External sources (created by others).
3. Prepackaged with Python and part of the standard library.
 - Examples: **math**, **time**, and **random**.

Importing modules

Standard library modules are imported by using the `import` command. You can also import specific functions or constants from a module by using the `from` command.

Examples include:

```
import math
```

```
-> use math.pi to access the pi constants
```

```
-> use math.exp(x) to access the exp function
```

```
from math import pi
```

```
-> use pi directly in your code
```

```
from math import exp
```

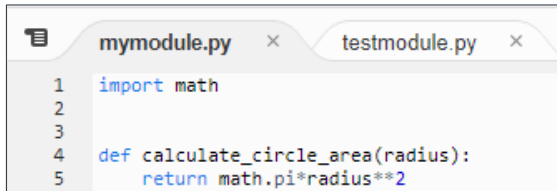
```
-> use exp directly in your code
```

You must import the module or the function before you can use it, even if the module is part of the standard library.

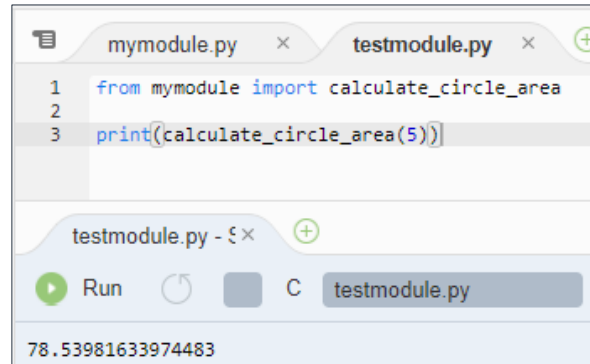
Creating modules

To make your own module :

- Create a file name with a .py extension (for instance mymodule.py)
- Add your code to define some functions
- You can now import mymodule in other python files and use the code defined in the module



```
1 import math
2
3
4 def calculate_circle_area(radius):
5     return math.pi*radius**2
```



```
1 from mymodule import calculate_circle_area
2
3 print(calculate_circle_area(5))
```

testmodule.py - 5x

Run C testmodule.py

78.53981633974483

Demonstration: Math Module



10

Instructions

1. Create a file that is called: **modules.py**
2. Import the **math** module.
3. Try the absolute value function: `math.fabs(x)`
4. Next, try the floor function: `math.floor(x)`
 - This function takes a float and returns the largest integer that is less than or equal to `x`.
5. Print your results and run the file.

Extra challenge:

1. Can you customize your **print** statement to describe what you did?
2. What happens when you try to put a string into `math.floor(x)`?



The following sample code can be used for this demonstration:

```
import math

absolute = -5.999
floor_test = 198.42

result1 = math.fabs(absolute)
result2 = math.floor(floor_test)

print(result1, " is the absolute value of ", absolute)
print(result2, " is the flow of ", floor_test)
```

File handlers

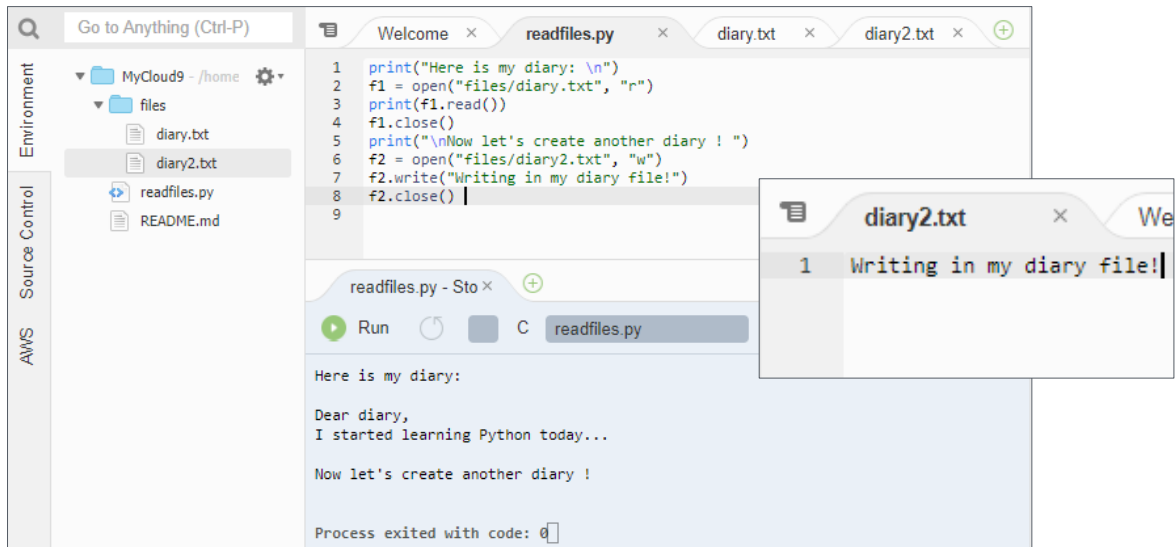
Use the built-in `open()` function.

Two arguments, both strings—the path to the file, and the file mode (read or write)—return a file handler object that represents that file.

File handlers can read data from the file or write data to the file. They can also pass data to other functions that can do so on your behalf (like **`json.load`** and **`json.dump`**).

File handlers must be closed after use, by calling the attached `.close()` function.

Example: File handlers



12

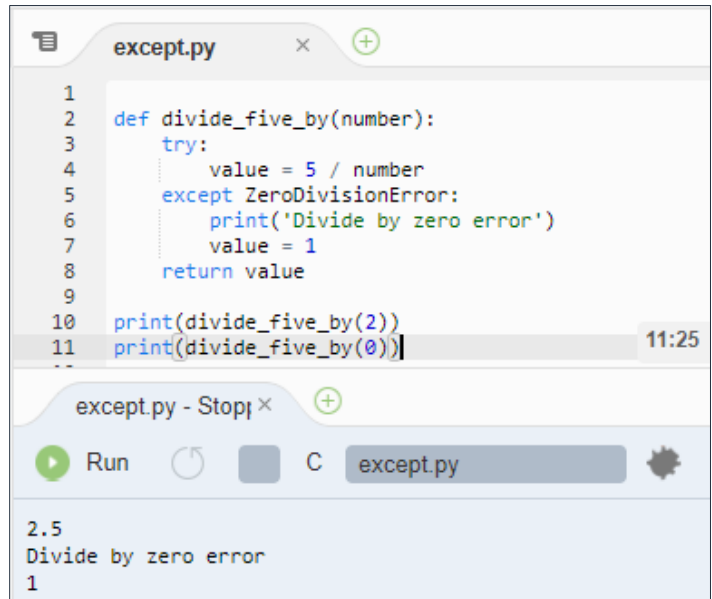
aws re/start

This example shows how file handlers are used to open, read, write, and close files. Line by line, the code proceeds as follows:

1. A file handler that is named `f1` is created by calling `open` on the file `diary.txt` in read mode (which is designated by the `r` argument).
2. The file handler reads the contents of the file. In this case, the file contents are written to the console.
3. Contents of the `diary.txt` file appear in the console.
4. Close the handler to the file. This step is important because, while the handler is open, it is using system resources. Closing the handler releases the resources back to the operating system for
5. `f2` is used to open the file `diary2.txt` in write mode (which is designated by the `w` argument).
6. A string is written to the file by calling `write()` on the `f2` file handler.
7. Close the handler. Again, this step is important. If the handler remains open it will reduce system performance by not releasing system resources back to the operating system for other processes.

Exception handling

- Exception handling is another form of flow control.
- When an error occurs, instead of failing and quitting the program, you can use a **try/except** block.
- You must specify the exception (or exceptions) that you expect might occur.



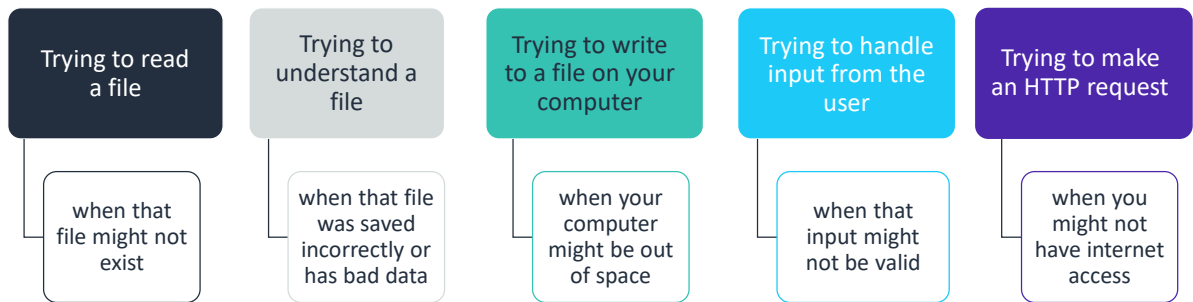
The screenshot shows a code editor window titled 'except.py'. The code defines a function 'divide_five_by' that takes a number as an argument. Inside the function, a 'try' block attempts to calculate '5 / number'. An 'except ZeroDivisionError' block catches the error and prints 'Divide by zero error'. After the exception handling, the function returns the value. The main code calls 'print(divide_five_by(2))' and 'print(divide_five_by(0))'. The output window below shows the results: '2.5' for the first call and 'Divide by zero error' followed by '1' for the second call. The time '11:25' is displayed in the top right corner of the editor window.

```
1
2 def divide_five_by(number):
3     try:
4         value = 5 / number
5     except ZeroDivisionError:
6         print('Divide by zero error')
7         value = 1
8     return value
9
10 print(divide_five_by(2))
11 print(divide_five_by(0))
```

Run C except.py

2.5
Divide by zero error
1

Examples: Exception handling





OS: Operating system module

- OS is part of the Python standard library.
- The OS module provides operating system functionality. The output of the module depends on the underlying operating system, with generally the same inputs.
- Common capabilities in the OS module are environment variable information, file manipulation, directory traversal, and process management.
- Programs that import and use the OS module are generally more portable between different platforms.

OS capabilities

- Host operating system:
 - **getlogin** – Returns the name of the logged in user
 - **getgrouplist** – Returns a list of group IDs that a user belongs to
 - **getenv** – Returns the value of the environment variable that is passed to it
 - **uname** – Returns information to identify the current OS
 - **system** – Is used to run commands in a subshell of the system
- Common functions for files:
 - **chown** – Changes the ownership of a file
 - **chmod** – Changes the access permissions of a file
 - **remove** – Removes the file at the given path
- Common functions in the os module for directories:
 - **getcwd** – Gets the current working directory
 - **listdir** – Lists the contents of the current directory
 - **makedirs** – Creates a new directory

OS: Run commands on the host system

```
>>> os.system("command on the system to run")

>>> os.system("adduser newuser")

>>> os.system("whoami")

>>> os.system("powershell.exe")
```

- Format that is used
- Linux command to add a user
- Linux or Microsoft Windows:
Displays the current user
- Generally only for Microsoft
Windows

Example: OS module – directory

```
>>> import os
>>> os.getcwd() '/home/username/folder'
>>> os.listdir() '["oldfolder"]'
>>> os.mkdir("newfolder")
>>> os.listdir() '["newfolder", "oldfolder"]'
```

The line-by-line explanation of the code explanation is as follows:

1. Import the `os` library.
2. To get the directory that the code runs in, use the `getcwd()` function from the `os` library. The output from the `getcwd()` call displays the current working directory of `‘/home/username/folder’`.
3. Calling `listdir()` lists the contents of the current directory. The output from the `listdir()` call displays the contents of the current directory. In this case it is `‘oldfolder’`.
4. Calling `mkdir()` creates a new directory. In this case, it creates a directory that is called `newfolder`.
5. Again, `listdir()` is called to list the contents of the directory. The output from `listdir()` shows the `oldfolder` entry and the new `newfolder` entry.

Demonstration: Work with the OS Module



19

Instructions

1. List your current working directory.
2. Create and then delete a new folder in your current working directory.
3. Print information about the currently logged-in user.



1. From the command line, start a Python console by entering *python* and pressing ENTER.
2. Run the following commands:

```
import os
os.listdir()
os.mkdir("newFolder")
os.listdir()
os.rmdir("newFolder")
os.listdir()
os.getlogin()
```

JSON

- JSON stands for JavaScript Object Notation.
- JSON is a standard file format that transmits data objects. It is language independent. It was originally derived from JavaScript, but now most modern languages include the ability to generate and parse JSON (including Python).
- JSON is used to *serialize* objects, which means that it turns code into a string that can be transmitted over a channel.

```
{
  "users": [
    {
      "name": "John Doe",
      "age": 25
    },
    {
      "name": "Li Juan",
      "age": 29
    },
    {
      "name": "Sofía Martínez",
      "age": 22
    }
  ],
  "dataTitle": "JSON Tutorial!",
  "swiftVersion": 2.1
}
```

JSON, continued

- Four useful functions in the JSON module are:
 - **dump**
 - **dumps**
 - **load**
 - **loads**
- **dump** and **dumps**: Turn various kinds of structured data into a string, which can be written to a file.
- **load** and **loads**: Turn a string back into structured data.
- **dump** and **load** work directly with files.
- **dumps** and **loads** work with strings.
 - The s at the end of the name is for *string*.

Uses for JSON

As mentioned earlier, JSON is used to make large amounts of data into strings and then put them back into their correct data types.

JSON takes a float and turns it into a string so that the information can be sent easily. Then, it turns the string back into a float when the information is received.

This technique is especially useful when you have more than one data type in a dataset. You can translate your Boolean, Integer, Float, and String to a single string.

You can then store the information in a file or a webpage.

When the data is retrieved and turned back, they are still the Boolean, Integer, Float, and String data types.

Example: JSON module

```
>>> import json
>>> json.dumps(5)
'5'
>>> json.dumps([1, "string", 3])
'[1, "string", 3]'
>>> json.loads('[1, "string", 3]')
[1, "string", 3]
```

The line-by-line code explanation is as follows:

1. Import the JSON library.
2. The `dumps()` call converts the indicated value into JSON.
3. The output from the conversion of 5 to JSON displays. In this case, it is the number 5.
4. The integers 1 and 3 are converted, along with the string *string*.
5. The output of the conversion displays.
6. The `loads()` function from the Python JSON library converts a JSON string into a Python dictionary.
7. The output of the `loads()` command displays.

Demonstration: Recognize and Greet People by Name



24

Instructions

1. Use the JSON module to save and recall the history of past runs.
 - a. Create **helloworld.py**.
 - b. Import the JSON module.
 - c. Write data to a file.
 - d. Examine the file.
2. Use the if statement for flow control.
 - a. Update helloworld.py to read from the file.
 - b. If the file has data, use it in the greeting. Otherwise, request the user's name.
3. Try/except blocks are used to handle exceptions.
 - a. Update the program again to use a try/catch block to handle file errors



The following code covers all the requirements in the instructions:

```
import json

filename = 'userName.json'
name = ''

# Check for a history file
try:
    with open(filename, 'r') as r:
        # Load the user's name from the history file
        name = json.load(r)
except IOError:
    print("First-time login")

# If the user was found in the history file, welcome them back
if name != "":
    print("Welcome back, " + name + "!")
else:
    # If the history file doesn't exist, ask the user for their name
    name = input("Hello! what's your name? ")
    print("Welcome, " + name + "!")
```



```
# Save the user's name to the history file
try:
    with open(filename, 'w') as f:
        json.dump(name, f)
except IOError:
    print("There was a problem writing to the history file.")
```

What is pip?

pip is the package manager for Python, and it is similar to apt in Linux.

- It is used to install third-party packages. A package holds one or more Python modules that you can use in your code.
- It is installed along with Python.

It is not called from within Python—pip is called from the command line, like Python itself is.

Demonstration: Install Requests with pip



26

Instructions

1. Test to see whether *Requests* is already installed.
 - To test it, in the command line, enter:
pip show requests
2. Review the results.
3. Install requests with pip.
 - Enter the command: ***pip install requests***
4. Review the results.
5. Test whether Requests is installed.
 - Use the same command that you used previously.
6. Review the results.

- For steps 1 and 2, run *pip show requests* and review the output.
- For steps 3 and 4, run *pip install requests* and review the output.
- Rerun *pip show requests* and review the output.

Checkpoint questions

What is the relationship between a function, a module, and a library?

What is the purpose of file handlers?

Can Python be used to run systems commands?

What are exceptions used for in Python?

True or False: When used, the JSON library automatically creates the correct data type for data.

What is pip?

Why use pip?

Answers:

1. Functions are blocks of code that can be wrapped in a module with other functions. Multiple modules can then be wrapped into a single library.
2. File handlers enable Python to read and write to files.
3. Yes, by calling functions like `system()` from the `os` library.
4. When an error occurs in a Python application, an exception is raised. The exception can be caught and handled to enable the application to elegantly handle the occurrence of errors.
5. True.
6. `pip` is the standard package manager for Python. You can use it to install and manage additional packages that are not part of the Python standard library.
7. `pip` makes it easier to manage packages in Python than it would be to install, update, and remove them manually.

Key takeaways



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

28

- Python uses modules and libraries to store code that is used often or that is distributed to many systems.
- Modules like *os* enable developers to run operating system commands.
- The module *JSON* can create, read, and write to JSON files.
- *pip* can be used to quickly and easily manage Linux packages that Python uses.

aws re/start

Some key takeaways from this lesson include:

- Python uses modules and libraries to store code that is used often or is distributed to many systems.
- Modules like *os* enable developers to run operating system commands.
- The *JSON* module can create, read, and write to JSON files.
- *Pip* can be used to quickly and easily manage Linux packages that Python uses.