

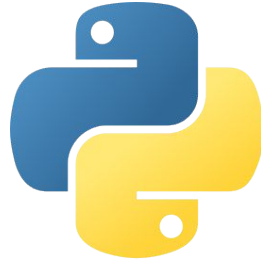
Software Development

Object-Oriented Programming with Python

Module 22



Python Built-In Modules



What are built-in modules?





Built-in modules: A set of libraries that come pre-installed with the Python installation.



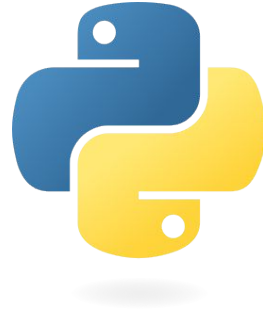
List the Built-In Modules

- To see all the built-in modules Python provides, activate Python in the command line, then type and run the following.
- The output will list about 83 built-in modules.

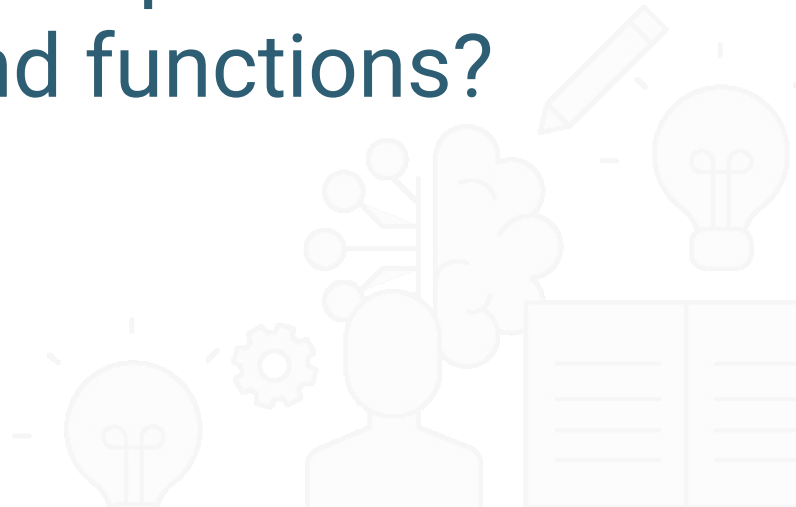
```
>>> help('modules')
```



Importing and Using **Modules** **and Functions**



How do you import and use
modules and functions?



Importing Modules and Functions

Module contents are made available to the caller with the import statement. The import statement takes many different forms:

- The statement `import <module_name>` only allows the module to be imported to the programming file. All the functions and methods in the module can be accessed.
- An alternate form of the import statement allows individual objects from the module to be imported directly into the programming file.

Python

```
import <module_name>
```

Python

```
from <module_name> import <function_name>  
(or <method_name>)
```


Using the Imported Function

The Python math module contains many methods and constants that can be used to perform mathematical tasks.

```
# Import the sqrt function from the math module.  
from math import sqrt  
  
# Calculate the square root of a number.  
number = 16  
result = sqrt(number)  
print(f"The square root of {number} is {result}")
```

The output from running the code is:

The square root of 16 is 4.0



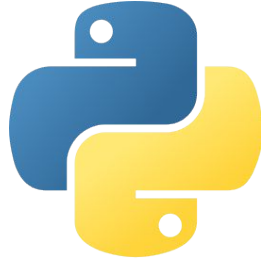
Pro Tip

The import statements are placed at the top of the file.





Modular **Programming**



What is modular programming?



Modular Programming

Modular programming involves breaking down large, complex programming tasks using smaller “building blocks,” called modules.

The advantages of using modules include:

Reusability

Modules are able to be called and used over and over again without the need to duplicate the code.

Simplicity and organization

Modules help to organize cumbersome code into smaller blocks, each of which accomplishes a specific task. Simplicity makes debugging errors easier too.

Maintainability

It is best practice for programmers to write modules with as few dependencies on other modules as possible. This reduces the risk that a change to one module will have a knock-on effect on other modules. This is especially important when collaborating on code.

Scoping

Remember LEGB scoping rules. Programmers have to consider which modules will have access to variables in the code. By using modules, any variables defined inside those modules are only accessible locally, which reduces the risk of naming conflict in the larger program.



Python File **Input Operations**



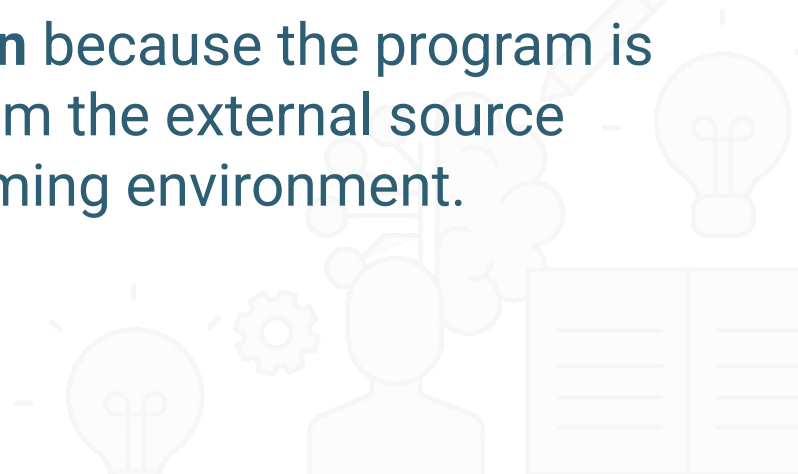
What are input operations?

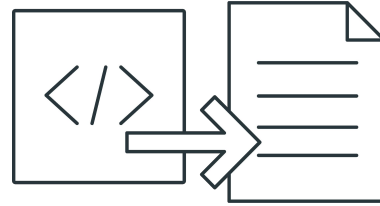




File input operations are actions that allow a program to read and receive data or information from external sources, such as reading data from a file.

Reading data from an external source file is an **input operation** because the program is receiving data from the external source into the programming environment.





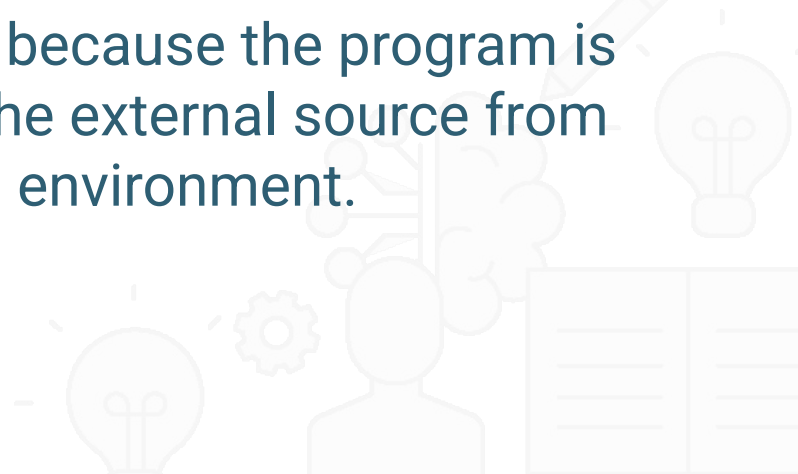
What are output operations?





File output operations are actions that allow a program to send data or information to external destinations, such as writing data to a file.

Writing data to an external source file is an **output operation** because the program is sending data to the external source from the programming environment.



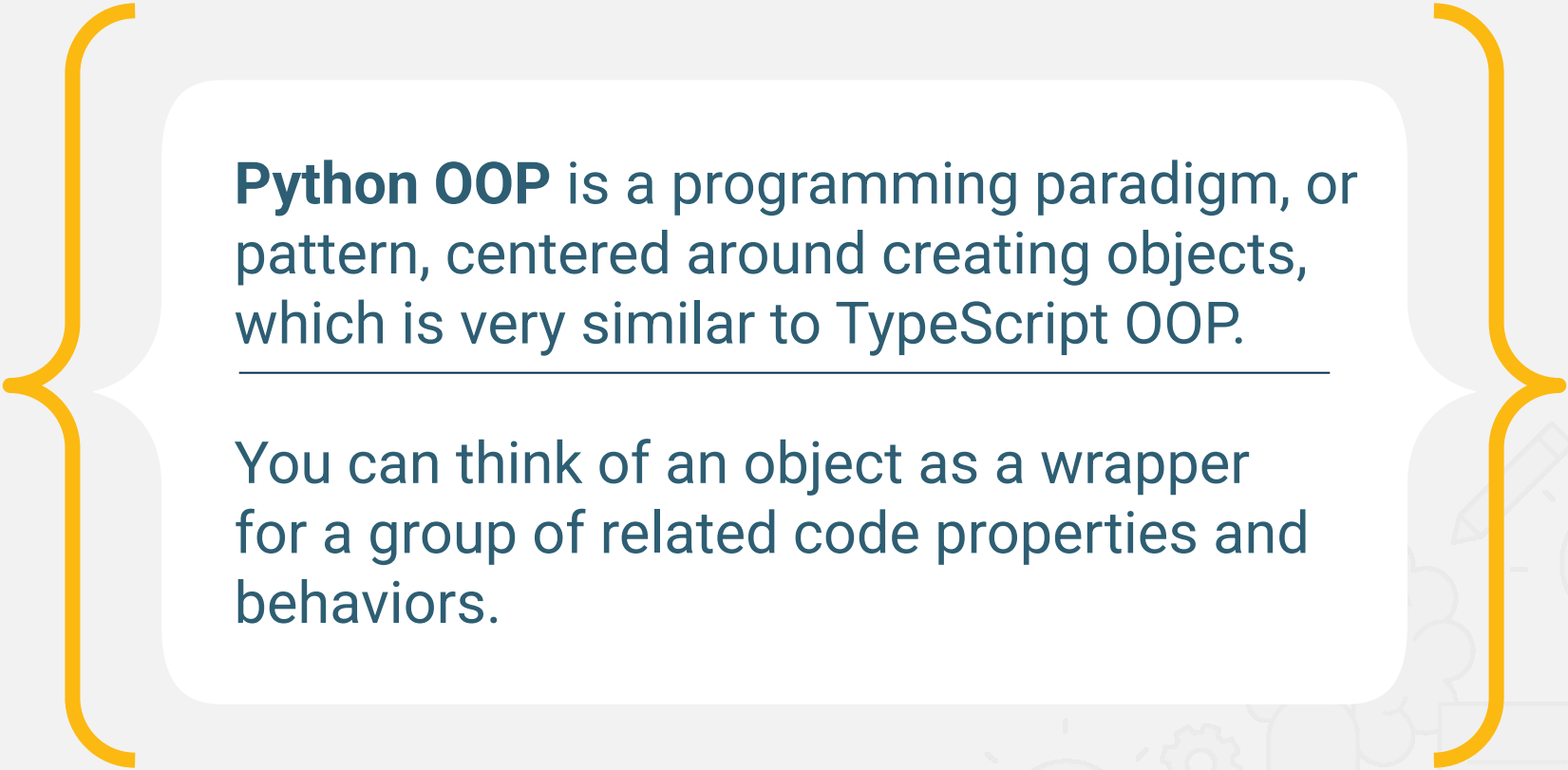


Python Object-Oriented Programming (OOP)



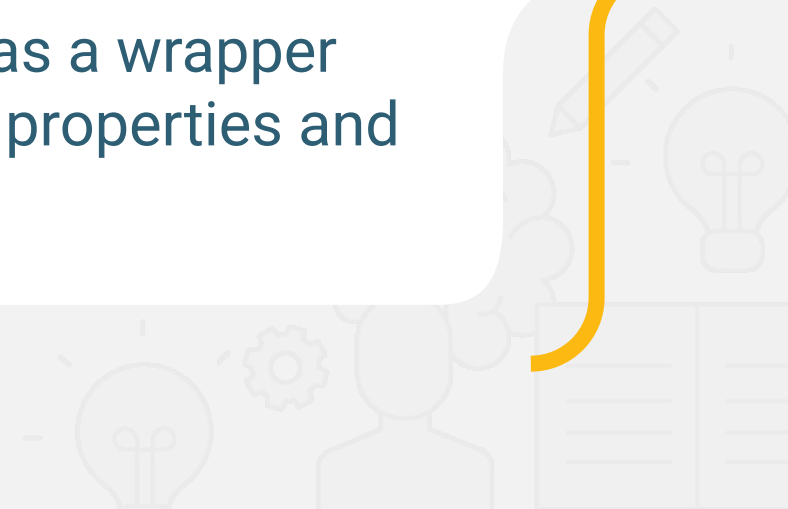
What is Python OOP?





Python OOP is a programming paradigm, or pattern, centered around creating objects, which is very similar to TypeScript OOP.


You can think of an object as a wrapper for a group of related code properties and behaviors.




Python OOP

Objects allow us to model things from the real world, as well as any relationships or interactions they have with other objects.

An object is a software entity that contains both data and procedures.

A dark blue circle with a white center, connected to a horizontal line extending to the left.

The **data** contained in an object is known as the object's **attributes**. You can think of these as the object's properties. For example, if you were creating an object to represent a restaurant ordering system, the object's attributes might be the menu, tables, and order.

A dark blue circle with a white center, connected to a horizontal line extending to the left.

The **procedures** that an object performs are known as **methods**. **Methods** are **functions** that are defined inside of an object and perform operations on the object's attributes. The methods are responsible for the object's behavior. In our restaurant ordering system, the methods would be an item's price and description, updating table status during reservations, and updating the order during preparation.

OOP

Objects' ability to communicate with each other makes them well-suited to address complex problems. The following concepts you learned with TypeScript OOP are the foundations of the OOP paradigm:

1

Encapsulation: Bundles the data (attributes) and the methods (functions) that operate on the data into a single unit, typically a class. This restricts access to some components to prevent accidental data modification.

2

Inheritance: Allows a new class to inherit properties and methods from existing classes. The new class, known as a subclass or derived class, extends the functionality of the parent class (or base class).

3

Polymorphism: Enables different classes to be treated as instances of a common superclass, allowing for methods to be defined in a base class and overridden in derived classes, enabling a single function to operate in different ways.

OOP

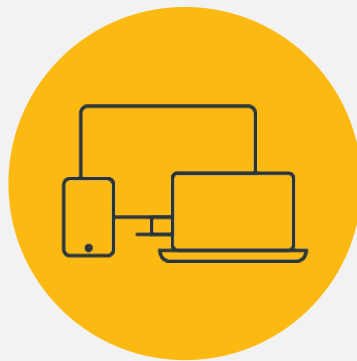
Objects' ability to communicate with each other makes them well-suited to address complex problems. The following concepts you learned with TypeScript OOP are the foundations of the OOP paradigm:

1

Composition: A design principle where a class is composed of one or more objects from other classes, forming a "has-a" relationship. Instead of inheriting behavior, a class can reuse functionality by containing instances of other classes.

2

Abstraction: Hides complex implementation details and only shows the most essential information and functionalities to the users.



Instructor **Demonstration**

Mini Project



Questions?





The End