Accessing Python Modules: Key Points

Types of Python Modules

1. **Built-in Modules**
   - Pre-installed in Python, providing ready-to-use functionality.
   - Example: `sys`, `calendar`, `math`.
2. **User-Defined Modules**
   - Python files created by users to define specific functionality.
   - Example: A file named `my_module.py` with custom functions.

Module Search Path

When importing a module, Python searches for it in the following order:

1. **Current Directory**: The folder where the script is being executed.
2. **Built-in Modules Directory**: Pre-installed modules provided by Python.
3. **Python Path Environment Variable**: A list of directories specified in the `PYTHONPATH` environment variable.
4. **Installation-Dependent Default Directory**: The directory where Python is installed.

Accessing Built-in Modules

- Built-in modules must be explicitly imported before use.
- **Example: Accessing the `sys` module and its search paths**

```
import sys

# Get a list of directories where Python searches for modules
locations = sys.path
print("Python module search paths:")
for location in locations:
    print(location)
```

Using the `calendar` Module

- **Importing the `calendar` Module**

```
import calendar
```

- **Functions in the `calendar` Module**

  i. **`calendar.leapdays(year1, year2)`**

  - Returns the number of leap days between two years (exclusive).
  - Example:

```
leap_days = calendar.leapdays(2000, 2050)
print(f"Leap days between 2000 and 2050: {leap_days}")
# Output: Leap days between 2000 and 2050: 13
```

  ii. **`calendar.isleap(year)`**

  - Returns `True` if the specified year is a leap year, otherwise `False`.
  - Example:

```
is_leap = calendar.isleap(2036)
print(f"Is 2036 a leap year? {is_leap}")
# Output: Is 2036 a leap year? True
```

Best Practices for Using Modules

1. **Import Modules at the Start of the Code**
   - While modules can be imported anywhere, importing them at the beginning is considered good practice for clarity.
2. **Explore Module Documentation**
   - Hover over a module (e.g., in Visual Studio Code) while pressing the `Ctrl` (Windows) or `Command` (Mac) key to view its definitions and location.
3. **Check Module Dependencies**
   - Built-in modules like `calendar` may internally import other modules, making them highly reusable.

Exploring Module Locations

- **Find the location of a built-in module**:

```
import calendar
```

```
print(calendar.__file__)  # Prints the file path of the module
```

- Example output:

```
/usr/lib/python3.9/calendar.py
```

Key Takeaways

- Built-in modules and user-defined modules streamline coding by providing pre-built functionality.
- Python searches for modules in a specific order, starting with the current directory.
- Built-in modules like `sys` and `calendar` are efficient and widely used for common tasks.
- Leveraging modules helps in creating clean, modular, and reusable code.