Python Import Statements and Modules: Key Points

1. Importing Modules in Python

- Any Python file (with a `.py` extension) is a **module**.
- The file you are working with is referred to as the **main module** (e.g., `check_imports.py`).
- To import another Python file in the same directory, use:

```
import sample  # No need to include the `.py` extension
```

2. Import Limitations

- Only `.py` files can be imported as modules.
- Attempting to import non-Python files (e.g., `.txt`) results in an **ImportError**.

3. Built-in Modules

- Python includes a library of **built-in modules** that can be directly imported without installation (e.g., `json`, `os`, `math`).

```
import json  # Now you can use functions from the json module
```

4. Packages in Python

- **Packages** are collections of modules organized in directories.
- For Python to recognize a directory as a package, it must contain a special file named `__init__.py`.

5. Community-Built Packages and PyPI

- Python Package Index (**PyPI**) hosts community-built packages.
- Use `pip` or `pip3` to install packages:

```
pip install seaborn  # Installs the Seaborn package
```

- Once installed, packages can be imported like built-in modules:

```
import numpy  # Importing the NumPy package
```

6. Importing Modules from Different Directories

- To import files from directories outside the current working directory:

  i. Use the `sys` module to modify the **Python Path**:

  ```
  import sys
  sys.path.insert(0, r'path_to_directory')  # Add the directory to sys.path
  ```

  ii. Import the desired module:

  ```
  import trial  # Assuming trial.py exists in the specified path
  ```

- **Example Code**:

```
import sys

# Add a specific directory to sys.path
sys.path.insert(0, r'C:\path\to\workplace')

# Import the module from the added path
import trial

# Access variables or functions defined in trial.py
print(trial.names)  # Prints: ['Adrian', 'Maria']
```

7. Best Practices

1. **Organize your files**: Keep related modules in the same directory or package to avoid modifying `sys.path`.
2. **Import early**: Place `import` statements at the beginning of the file.
3. **Use absolute imports**: Prefer absolute paths for clarity over modifying the Python Path.

8. Key Takeaways

- Importing from the current directory is straightforward and recommended for most cases.
- Built-in and installed packages save time and effort by offering pre-built functionality.
- Adding directories to `sys.path` is an option but can introduce complexity; use it sparingly.
- Move required files into the current working directory whenever possible.