# Introducing Python packages

In this lesson, we'll investigate what we mean when we refer to Python packages. We'll consider how we use built-in Python packages, as well as mention some standard Python libraries and their applications.

## Learning objectives

In this train, we will learn how to:

- Define a package and its hierarchical organisation.

## What are packages in Python?

For obvious reasons, we can't really store all of our files on our computer in the same location or folder. It would be nearly impossible to find the relevant file we're looking for! Therefore, we make use of well-organised directory structures for easier accessibility.

A specific directory is designated to files that share similarities, for example, we may keep all the photos in the "Pictures" directory. In the same way, **Python packages can be seen as structures similar to directories** with **modules playing a role similar to files**.

As our program grows larger in size with an increased number of modules, we can cluster similar modules in one package and other clusters of similar modules in different packages. This, in turn, will allow for the efficient management of our project (program), making it conceptually clear. Similarly, as a directory can contain subdirectories and files, a Python package can also contain sub-packages and modules.

This organisation of modules and code allows us to define a **Python package (also referred to as a library) as a collection of hierarchically structured directories of Python code, consisting of sub-packages and modules.**

In order for a directory to be considered as a package by Python, it must contain a file named `__init__.py`. This file can be left empty but the initialisation code for that package is generally placed in this file.

The figure below presents a possible organisation of packages and modules present if we were developing a game:

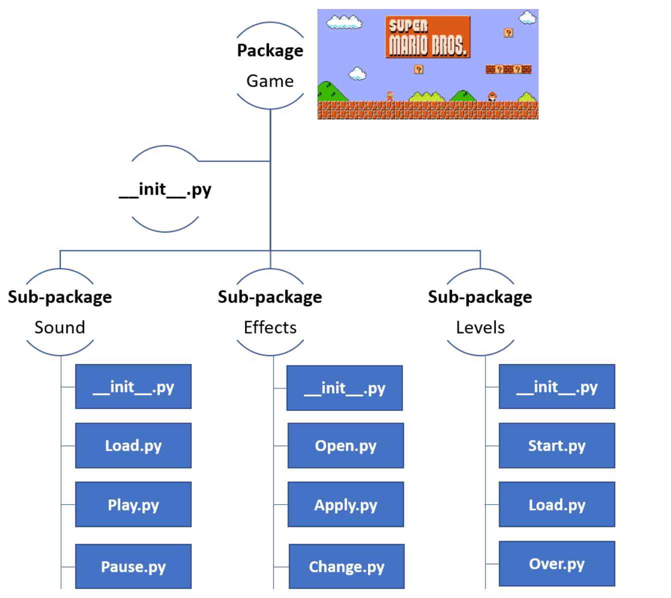

Figure 1: In this figure, we use a game to represent the package structure used in Python, allowing us to logically structure our code as it grows in complexity.

## Standard Python packages

Python distributions are shipped with a standard list of libraries/packages. Some of these include:

*It's not crucial to know what all these packages do right now, but it is useful to know that they exist, so feel free to read up on them in more detail later.*

**Text processing services**

- string â€“ Common string operations
- re â€“ Regular expression operations
- unicodedata â€“ Unicode Database

**Data types**

- datetime â€“ Basic date and time types
- calendar â€“ General calendar-related functions
- array â€“ Efficient arrays of numeric values
- copy â€“ Shallow and deep copy operations
- pprint â€“ Data pretty printer

**Numeric and mathematical modules**

- numbers â€“ Numeric abstract base classes
- math â€“ Mathematical functions
- cmath â€“ Mathematical functions for complex numbers
- decimal â€“ Decimal fixed point and floating point arithmetic
- fractions â€“ Rational numbers

- random â€“ Generate pseudo-random numbers
- statistics â€“ Mathematical statistics functions

**File and directory access**

- pathlib â€“ Object-oriented filesystem paths
- fileinput â€“ Iterate over lines from multiple input streams
- stat â€“ Interpreting stat() results
- filecmp â€“ File and directory comparisons
- tempfile â€“ Generate temporary files and directories
- shutil â€“ High-level file operations

**Data persistence**

- pickle â€“ Python object serialisation
- copyreg â€“ Register pickle support functions
- shelve â€“ Python object persistence
- marshal â€“ Internal Python object serialisation
- dbm â€“ Interfaces to Unix â€œdatabasesâ€
- sqlite3 â€“ DB-API 2.0 interface for SQLite databases

**Data compression and archiving**

- zlib â€“ Compression compatible with gzip
- gzip â€“ Support for gzip files
- bz2 â€“ Support for bzip2 compression
- lzma â€“ Compression using the LZMA algorithm
- zipfile â€“ Work with ZIP archives
- tarfile â€“ Read and write tar archive files

**File formats**

- csv â€“ CSV file reading and writing
- configparser â€“ Configuration file parser
- netrc â€“ netrc file processing
- xdrlib â€“ Encode and decode XDR data
- plistlib â€“ Generate and parse Mac OS X .plist files

**Cryptographic services**

- hashlib â€“ Secure hashes and message digests
- hmac â€“ Keyed hashing for message authentication
- secrets â€“ Generate secure random numbers for managing secrets

**Generic operating system services**

- os â€“ Miscellaneous operating system interfaces
- io â€“ Core tools for working with streams
- time â€“ Time access and conversions
- errno â€“ Standard errno system symbols
- ctypes â€“ A foreign function library for Python

**Concurrent execution**

- threading â€“ Thread-based parallelism
- multiprocessing â€“ Process-based parallelism
- subprocess â€“ Subprocess management

**Networking and interprocess communication**

- asyncio â€“ Asynchronous I/O
- socket â€“ Low-level networking interface
- ssl â€“ TLS/SSL wrapper for socket objects
- signal â€“ Set handlers for asynchronous events
- mmap â€“ Memory-mapped file support

**Internet data handling**

- email â€“ An email and MIME handling package
- json â€“ JSON encoder and decoder
- mailcap â€“ Mailcap file handling
- mailbox â€“ Manipulate mailboxes in various formats
- Graphical User Interfaces with Tk
- tkinter â€“ Python interface to Tcl/Tk

Python packages can also be installed from local or online repositories such as the**Package Index (PyPI)**, a repository of software for the Python programming language.

PyPI helps you find and install software developed and shared by the Python community. For specific applications such as scientific computing, packages can be installed using package managers such as Anaconda.

## Conclusion

In this train, we looked at Python packages; their creation and hierarchical organisation. There are endless uses of modules and packages, and each can be tailored to an area of expertise and shared with other programmers.

We can apply the skills we have gained from this train to create our own custom-built modules and packages. We can even create a package that may assist with data analysis!

## Appendix

Below are additional useful resources to help you further under Python modules and packages:

- Official Python Tutorial for Modules