Intro to Python
Lab 4

1 Modules and Packages

File to use: `pricing.py`
File to use: `use-pricing.py`
File to use: `product.py`

To help maintain our code base when it becomes more complex, we can group relevant code into separate modules based on their functionality, application or intended usage. These modules can then be reused by other programs when they require that specific functionality. This promotes code reuse and also facilitates maintenance of the code base (i.e. easier to debug 20 lines of code in 10 files than 200 lines of code in 1 file).

A module is a file that contains code to perform a specific task: this may include variables, functions, classes etc. Modules have the extension `*.py`, just like scripts. The name of the module does not include its extension. The primary difference between a module and a normal Python program / script is that modules are intended to be imported for use by another script, rather than directly executed like a script.

To use the objects defined in a module from another script, you can use the `import` statement in various forms.

When you import a module, a `__pycache__` folder will be created. This directory contains the compiled bytecode of the module (*.pyc files which are generated by the interpreter to be executed by the PVM), which can be used to speed up subsequent imports of the same module (as the bytecode does not need to be regenerated again for subsequent imports)

Python will search for the module file from 3 sources:

- The current folder in which the script that imports the module is located in
- A list of folders specified in the PYTHONPATH environment variable, whose value needs to be set before attempting to execute any script with Python interpreter.

- An installation-dependent list of folders were configured Python was installed (for e.g. in `/lib/site-packages` on a Linux system)

Typically the PYTHONPATH environment variable is not set for default Python / Anaconda installations. You can provide a value for it in Windows / Linux in the same way that you provide a value for any other environment variable:
https://www.geeksforgeeks.org/pythonpath-environment-variable-in-python/

The actual complete module search path is provided in the `sys.path` variable that comes from the `sys` module.

1.1 Standard library modules

File to use: `module-math-statistics.py`

The Python standard library contains well over 200 modules. These are known as built-in modules. Some of the most frequently used modules are: `os`, `random`, `math`, `time`, `sys`, `collections`, `statistics`

The complete list of modules are available at
https://docs.python.org/3/py-modindex.html

An example of common functions available in the `math` module:

https://www.w3schools.com/python/module_math.asp

An example of common functions available in the `statistics` module:

https://www.w3schools.com/python/module_statistics.asp

There is a special module (amongst these built-in modules) which is explicitly called builtins. This module contains what are known as the built-in functions. These functions are something which we can directly access in our code without the need to first import the module.

https://www.programiz.com/python-programming/methods/built-in

Examples of the built-in functions we have been using so far up to this point are: `print`, `range`, `type`, `sorted`, `map`, `filter`, etc

1.2 Packages

Packages are essentially larger collection of modules. There are many packages that perform a wide range of functionalities / tasks developed by the Python community.
PyPi is the official public repository for all packages in the Python ecosystem developed by the Python community.

https://pypi.org/

Typically, you will search for a package that you would like to use and then install it with:

```
pip install package_name
```

pip is the most popular package manager for Python. A package manager is a tool that automates the process of installing, upgrading, configuring, and removing packages for a computer in a consistent manner.

A list of the top / most popular packages from PyPi
https://hugovk.github.io/top-pypi-packages/
https://pypistats.org/top

Anaconda (which is the distribution we are using for this workshop) uses Conda instead of pip. Conda is an open source package management system and environment management system for Python, but can be used to package and distribute software for any language

Anaconda comes preinstalled with most of the major data science / engineering packages (unlike a bare installation of Python from https://www.python.org/downloads/, where you will need to install most of the packages yourself manually). To see which packages are already pre-installed, you can check from an Anaconda prompt with:

```
conda list
```

The best place to start searching for packages is from Anaconda.org:
https://anaconda.org/

You can then install the packages using the instructions given on the package page, for e.g.

```
conda install -c conda-forge pandas
```

2 Working with files / directories for automation scripting

2.1 os and sys module

File to use: `module-os.py`
File to use: `module-sys.py`

The `os` module provides an interface with the native OS to perform many OS related functionalities such as file and directory manipulation.
A list of some other commonly used functions from this module:
https://python101.pythonlibrary.org/chapter16_os.html

You can execute the standard DOS commands
https://www.lifewire.com/dos-commands-4070427
or Linux shell commands
https://www.hostinger.my/tutorials/linux-commands
from this module

The `sys` module provides functions and variables used to manipulate different parts of the Python runtime environment. The most useful part of this module is to process the command line arguments that are passed to the script.

For e.g. open an Anaconda prompt, navigate to the directory in which `demo-sys.py` is located in and execute it by passing in 3 command line arguments:

```
python demo-sys.py cat dog mouse
```

Command line arguments are the most common ways of customizing the behaviour of automation scripts.

More documentation on common functions from this module is available at:
https://python101.pythonlibrary.org/chapter20_sys.html

2.2 Reading from / writing to text files

File to use: `read-from-text.py`
File to use: `write-to-text.py`

To read a text file in Python, follow these steps:

a) First, open a text file for reading by using the `open()` function.
b) Second, read text from the text file using the file `read()`, `readline()`, or `readlines()` method of the file object.
c) Third, close the file using the file `close()` method. This is not necessary if the file has being opened using the `with` keyword. The `with` keyword closes the file automatically and performs necessary exception handling in the event of an error during the file operation.

To write to a text file in Python, follow these steps:

a) First, open the text file for writing (or append) using the `open()` function.
b) Second, write to the text file using the `write()` or `writelines()` method.
c) Third, close the file using the `close()` method. This is not necessary if the file has being opened using the `with` keyword.

Different Modes to Open a File in Python

Mode	Description
`r`	Open a file for reading. (default)
`w`	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
`x`	Open a file for exclusive creation. If the file already exists, the operation fails.
`a`	Open a file for appending at the end of the file without truncating it. Creates a new file if it does not exist.
`t`	Open in text mode. (default)
`b`	Open in binary mode.
`+`	Open a file for updating (reading and writing)

2.3 Reading from / writing to CSV files

File to use: `read-from-csv.py`
File to use: `write-to-csv.py`

CSV files are common format for semi-structured data that are usually generated from Excel spreadsheets. They are very widely used for data analytics
Python comes with an existing `csv` module that makes it easy for you to process CSV files.

2.4 Copying, renaming, deleting and zipping files

File to use: `copy-zip-files.py`
File to use: `rename-files.py`

The `shutil` module provides some useful operations such as copying, compressing and finding files
https://www.codingninjas.com/studio/library/shutil-module-in-python

Another popular module for working with zip files (compressing / decompressing) is `zipfile`
https://www.geeksforgeeks.org/working-zip-files-python/

For deleting files, there are many possible approaches:
https://favtutor.com/blogs/delete-file-python