PYTHON MODULES AND PACKAGES
AN INTRODUCTION

PYTHON MODULES AND PACKAGES

Modules and Packages facilitate modular programming.

Modular programming refers to the process of breaking a large, unwieldy programming task into separate, smaller, more manageable subtasks or modules.

PYTHON MODULES AND PACKAGES

What are advantages of modularizing code in a large application?

- Simplicity
- Maintainability
- Reusability
- Scoping

TABLE OF CONTENTS

TABLE OF CONTENTS

WRITING A MODULE

Three different styles of modules in Python

- **A module written in Python itself**
- **A module written in C and loaded dynamically at run-time**
- **A built-in module is intrinsically contained in the interpreter**

TABLE OF CONTENTS

THE MODULE SEARCH PATH

Where can you `import` a module from?

- The interpreter searches for the file
  - In the current directory
  - In the PYTHONPATH environment variable list of directories
  - The directories configured as part of your Python installation

THE MODULE SEARCH PATH

Where should you put your module file?

- **To ensure you module is found place the file in:**
  - **The same directory as the input script or the current directory**
  - **Modify PYTHONPATH environment variable to contain the directory where it is located**
    - **Or in one of the directories already in the PYTHONPATH**
  - **In one of the directories configured as part of your Python installation**

THE MODULE SEARCH PATH

Where should you put your module file?

- **Or you can modify the** `sys.path` **list at run time**

```
sys.path.append(r'C:\Users\chris\ModulesDirectory')
```

TABLE OF CONTENTS

THE `import` STATEMENT

What forms can the import statement take?

- **The simplest form**

```
import <module_name>
```

- **The module contents are not *directly* accessible to the caller**
  - **A module creates a separate namespace**

THE `import` STATEMENT

What forms can the import statement take?

- **Individual objects from the module can be imported**

```
from <module_name> import <name(s)>
```

- **The individual objects are *directly* accessible to the caller**
    - **Objects are imported into the caller’s symbol table**

THE `import` STATEMENT

What forms can the import statement take?

- **It is possible to import everything from a module at once**

```
from <module_names> import *
```

- **This places all the names of objects into the local symbol table**
    - **With the exception of any that begin with an underscore**
    - **NOTE: This isn’t necessarily recommended**
    - **Unless you know all the names will not conflict and overwrite existing names**

THE import STATEMENT

What forms can the import statement take?

- **Individual objects can be imported with alternate names**

```
from <module_name> import <name> as <alt_name>
```

- **Making it possible to place names directly into the local symbol table**
    - **Avoiding conflicts with existing names**

THE `import` STATEMENT

What forms can the import statement take?

- **Import the entire module under an alternate name**

```
import <module_name> as <alt_name>
```

THE `import` STATEMENT

What forms can the import statement take?

- **Module contents can be imported from within a function definition**

TABLE OF CONTENTS

THE `dir()` FUNCTION

View the defined names in a namespace

- **The built-in function `dir()` returns a list of defined names in a namespace**
- **Without arguments, it produces an alphabetically sorted list of names in the current local symbol table**

THE `dir()` FUNCTION

How to view the defined names in a namespace

- **When given the name of a module as an argument,** `dir()` **lists the names defined in the module**

TABLE OF CONTENTS

EXECUTING A MODULE AS A SCRIPT

Is a **module** also a Python **script**?

- **Any `.py` file that contains a module is essentially also a Python script**

EXECUTING A MODULE AS A SCRIPT

What if you don’t want a module to generate output when imported?

- **When a `.py` file is imported the dunder variable `__name__` is set to the name of the module**
- **When a `.py` file is run as a standalone script, `__name__` is set to the string `'__main__'`**

TABLE OF CONTENTS

RELOADING A MODULE

A Module is only loaded once per interpreter session

- This works fine for function and class definitions
- But modules can contain executable statements as well
    - Usually for initialization
    - These statements will only be executed the first time a module is imported

RELOADING A MODULE

Is it possible to reload a module if needed?

- **You can restart the interpreter**
- **Or use a function called** `reload()` **from the module** `importlib`

TABLE OF CONTENTS

PYTHON PACKAGES

How to keep track of a growing number of modules

- **Packages allow for a hierarchical structuring of the module namespace using dot notation**
- **Example package structure:**

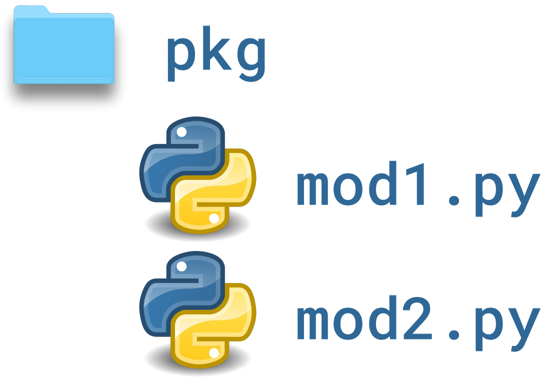
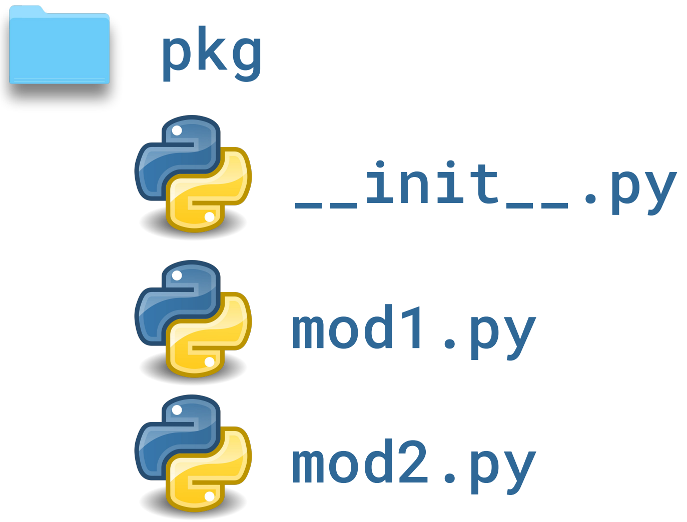


TABLE OF CONTENTS

PACKAGE INITIALIZATION

The `__init__.py` file

- **If a file named __init__.py is present in a package directory, it is invoked when the package or a module in the package is imported**



PACKAGE INITIALIZATION

The `__init__.py` file

- **__init__.py can also be used to automatically import the modules from a package**

TABLE OF CONTENTS

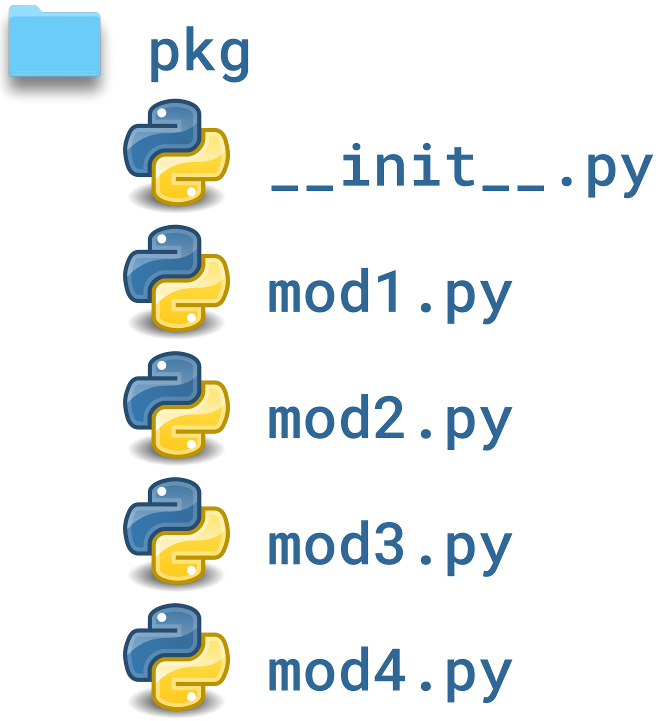
IMPORTING * FROM A PACKAGE

Expanding the current package

- pkg
  - mod1.py
  - mod2.py
  - mod3.py
  - mod4.py

IMPORTING * FROM A PACKAGE

Add an `__init__.py` file with a list named `__all__`



IMPORTING * FROM A PACKAGE

The `__all__` list controls what is imported when `import *` is specified

- **For a package, when `__all__` is not defined, `import *` does not import anything**
- **For a module, when `__all__` is not defined, `import *` imports everything (except names starting with an underscore)**

TABLE OF CONTENTS

SUBPACKAGES

Packages can contain nested subpackages to arbitrary depth

```
pkg
    sub_pkg1
        mod1.py
        mod2.py
    sub_pkg2
        mod3.py
        mod4.py
```

SUBPACKAGES

It is possible to use a relative import

- `..` **evaluates to the parent package**
- `..sub_pkg` **evaluates to the subpackage of the parent package**

CONGRATULATIONS

YOU’VE COMPLETED THE COURSE!

PYTHON MODULES AND PACKAGES

AN INTRODUCTION

TABLE OF CONTENTS

THANK YOU!

PRACTICE WITH
WHAT YOU HAVE LEARNED