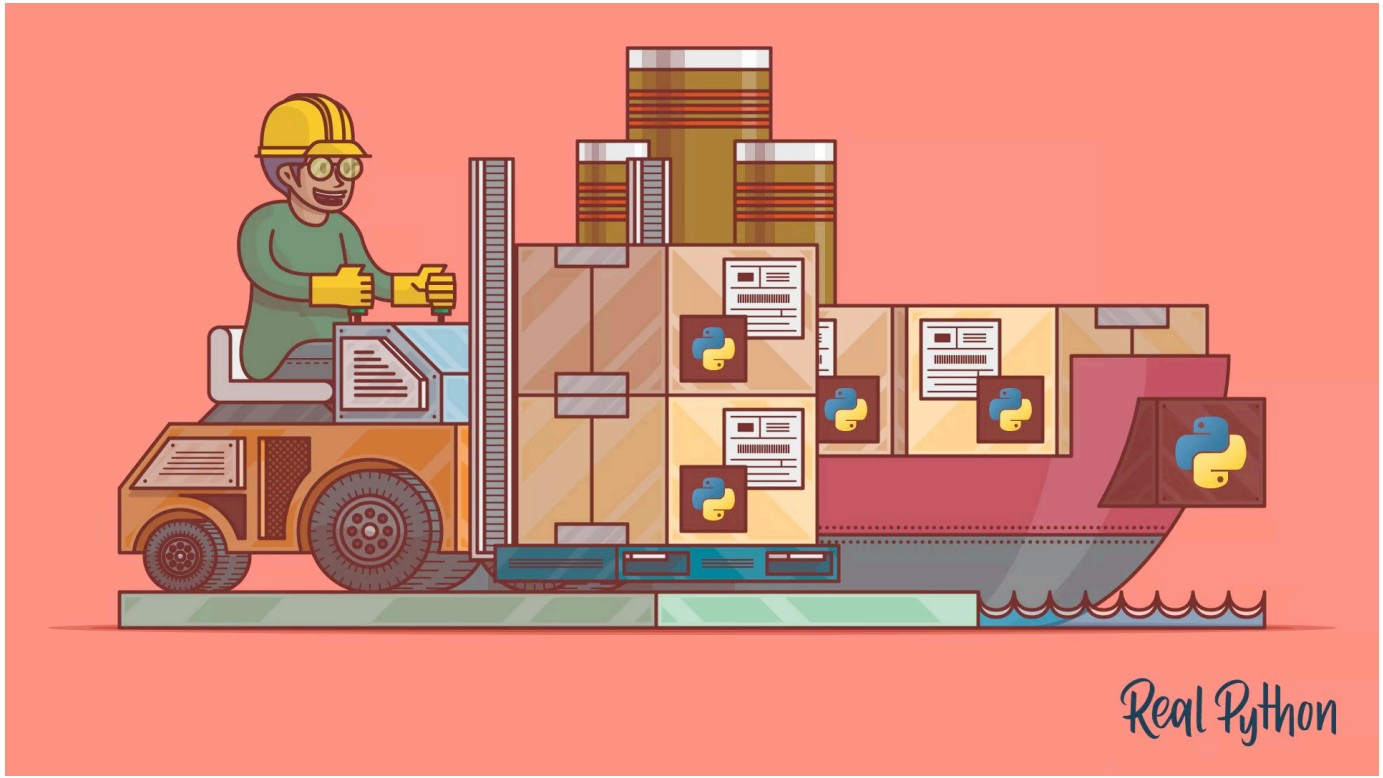
ADVANCED PYTHON IMPORT TECHNIQUES



YOU WILL LEARN ABOUT:

1. All the variations of the `import` statement
2. The difference between modules, packages, and namespaces
3. How to create installable packages
4. What the import system does when you load a module
5. The problems with circular imports
6. How to dynamically load modules
7. A way to bundle your code as a runnable zip file

VERSIONS

Note:

- Sample code tested using:
  - Python 3.12

OVERVIEW

- For anything but the smallest program you want to organize your code
- Python uses the file system to define a hierarchy of modules and packages
- The import keyword can be used in a variety of ways to load modules and objects
- Your scripts can be bundled up for re-use as installable packages or as runnable zip files
- Modules and packages can be dynamically loaded

NEXT UP...

Variations on the `import` keyword

TABLE OF CONTENTS

MODULES, PACKAGES, AND NAMESPACES

- To organize your code you want to group it together
- **Namespace (general):** abstract container for holding logical groupings
  - Prevents name clashes
- Python mostly uses directory structure to define groupings
- **Module:** organizational unit in Python
  - Typically corresponds to a single Python file
- **Package:** a module with submodules or recursively sub-packages
  - Directory with a `__init__.py` file
  - Bundled for installation and re-use
- **Namespace Package (specific):** virtual package that can be split into parts

MODULES, PACKAGES, AND NAMESPACES

- Python ships with many modules and packages
- Packages can be installed as third-party containers
- The **import** keyword brings a package or module into your namespace
  - Namespace of a Python file (module)
  - Namespace of a REPL session

NEXT UP...

Packages

TABLE OF CONTENTS

PACKAGES

- A directory structure containing a `__init__.py` file
- By default submodules or sub-packages don’t get imported when the parent does
- The `__init__.py` file can contain code
  - Importing submodules or sub-packages into the parent namespace

NEXT UP...

Shadowing

TABLE OF CONTENTS

SHADOWING

- Python uses references to point at objects
- You can create a new reference with the same name and point it elsewhere
- Overrides the existing definition: “shadowing” it
- Even modules are objects, and in most cases they can be shadowed

NEXT UP...

Installable packages

TABLE OF CONTENTS

INSTALLABLE PACKAGES

- You can bundle up a package in order to share or re-use it
- Includes the Python package directory and metadata
- These same bundles are what you install from PyPI
- You can install a bundle locally
- Packaging in Python has been historically complicated

INSTALLABLE PACKAGES

- Current best practices (2024):
  - `pyproject.toml`
  - Put your modules in a `src` directory
  - Third party tools:
    - `setuptools==72.1.0`
    - `build==1.2.1`
    - `twine==5.1.1`

NEXT UP...

The problem with relative imports

TABLE OF CONTENTS

ABSOLUTE VS RELATIVE IMPORTS

- An **absolute import** use a fully qualified module name
- A **relative import** references things locally
  - Through a dot:

```
# inside americas.north
from . import canada
```

  - Through the local name:

```
# inside americas
from north import canada
```

RELATIVE IMPORTS

- Relative imports:
  - Mean less typing
  - Are problematic within packages or zip files
  - Aren’t recommended

NEXT UP...

Namespace packages

TABLE OF CONTENTS

NAMESPACE PACKAGES

- Not all programming languages use a file based packaging mechanism
- Some languages are more declarative
  - Example: all network related modules can go in a package named “network” even from different distributors
- Python 3.3 introduced **implicit namespace packages**
  - A way to group things that aren’t in the same bundle
  - No `__init__.py` file!

NEXT UP...

Module cache

TABLE OF CONTENTS

MODULE CACHE

- When you import a module, Python caches it
- Subsequent imports for the same module do nothing
- Whole module gets imported and cached even when you are adding a single item into the namespace

RELOAD

- Modules are cached
- Changes to a module won’t take effect
- You can explicitly reload a module

NEXT UP...

Circular imports

TABLE OF CONTENTS

MODULES THAT IMPORT MODULES

- Modules can import other modules
- A imports B which imports A
- Circular imports
- Mostly not a problem since modules are cached
- Corner cases can causes issues

NEXT UP...

Dynamic imports

TABLE OF CONTENTS

DYNAMIC IMPORTS

- Sometimes you want to import a module based on input from the user
  - Conditionally import based on data
  - Plugins
- `importlib.import_module()`

IMPORTING RESOURCES

- Complex programs often ship with data
  - Internal configuration files
  - Assets (sounds, images, etc)
  - Data
- You can use `__file__` to find them, or
- You can use `importlib.resources`

NEXT UP...

Python runs zips

TABLE OF CONTENTS

PYTHON RUNS ZIPS

- Python can run code inside a zip file
- The `zipapp` utility helps you create a runnable zip file
  - Automatically creates a `__main__.py` in the zip

RUNNING YOUR ZIP

- In Windows, the `.pyz` extension is already considered runnable and associated with the Python interpreter
- On *nix systems you need to associate a runnable
  - Like a shebang (#!)

```
$ python -m zipapp goodbye -m talk:main -p "/usr/bin/env python"
```

- This does not include Python in the zip!

NEXT UP...

Tips and tricks

TABLE OF CONTENTS

HANDLING PACKAGES ACROSS VERSIONS

- Some built-in packages have been back-ported as third party libraries
- Depending on what Python version a user is running they may need different modules

```
try:
    from importlib import resources
except ModuleNotFoundError:
    import importlib_resources as resources
```

```
import sys
if sys.version_info >= (3, 7):
    from importlib import resources
else:
    import importlib_resources as resources
```

PACKAGE ALTERNATIVES

- Some packages are drop-in replacements for built-in packages

```
try:
    import ujson as json
except ModuleNotFoundError:
    import json
```

- Use a `mock` instead

SCRIPTS VS LIBRARIES

- Loading a module runs any bare code in the module
- Scripts are modules

```
def main():
    print("Do something!")

if __name__ == "__main__":
    main()
```

PROFILING IMPORTS

- The CPython interpreter will give you performance information about import load times
- Use `-X importtime`

```
$ python -X importtime -c "import math"
import time: self [us] | cumulative | imported package
import time:       223 |        223 |   _io
import time:        28 |         28 |   marshal
import time:       188 |        188 |   posix
import time:       250 |        687 | _frozen_importlib_external
...
import time:       564 |        564 | math
```

IMPORTS STYLE GUIDE

- PEP 8 has recommendations about imports:
  - Keep imports at the top of the file
  - Write imports on separate lines
  - Organize imports into groups:
    - first standard library imports, then
    - third-party imports, and
    - finally local application or library imports
  - Order imports alphabetically within each group
  - Prefer absolute imports over relative imports
  - Avoid wildcard imports like from module `import *`
- Some linters apply these style guidelines

IMPORTS STYLE GUIDE

```
# Standard library imports
import sys
from typing import Dict, List

# Third party imports
import feedparser
import html2text

# Reader imports
from reader import URL
```

NEXT UP...

Summary

TABLE OF CONTENTS

SUMMARY

- You organize your code in Python into modules and packages
- The namespace is where everything currently defined lives
- To use a module you invoke the `import` keyword bringing it into the namespace
- A package is a directory with a `__init__.py` file
- A `__init__.py` file is still Python
    - Can be used to import other modules or objects into the namespace

SUMMARY

- You can replace an item in the namespace with another using the same name
  - Shadow built-ins
- You can name a module the same as a built-in and use it instead
- Packages can be bundled for re-use
  - Provide metadata using a `pyproject.toml` file
  - Install locally using `pip install -e`

SUMMARY

- Relative importing is connected to your current location
  - Problematic with bundled packages or `pyz` files
- Imports are cached:
  - Importing something twice does nothing
  - Changing the code and re-importing it doesn't work
  - You can reload a module using tools in `importlib`
- Modules and resources can be loaded dynamically
- Python can run code inside a zip file

MORE FROM REAL PYTHON

- Python import: Advanced Techniques and Tips

  https://realpython.com/python-import/ (tutorial)

  - Examples:
    - Using namespace packages as interfaces
    - Reading resources (x2)
    - Factory methods for plugins
    - Using modules as singletons
  - Finders and loaders

MORE FROM REAL PYTHON

- How to Publish an Open-Source Python Package to PyPI
  - https://realpython.com/courses/pypi-publish-python-package/ (course)
  - https://realpython.com/pypi-publish-python-package/ (tutorial)
- What's a Python Namespace Package, and What's It For?
  - https://realpython.com/python-namespace-package/ (tutorial)
- Python 3.8: Cool New Features for You to Try
  - https://realpython.com/python38-new-features/#importlibmetadata (tutorial)
  - https://realpython.com/lessons/importlibmetadata/ (course)
- Python Zip Imports: Distribute Modules and Packages Quickly
  - https://realpython.com/python-zip-import/ (tutorial)

Thanks!

Dankie ju faleminderit faleminderit شكرا Grazias Շնորհակալություն Sağ ol eskerrik asko Дзякуй তোমাকে ধন্যবাদ hvala trugéré
благодаря Akeva Chezu ba gràcies Salamat zikomo 谢谢 hvala děkuji Tak dank u Dankon aitäh takk fyri salamat kiitos Merci
Grazas დიდი მადლობა Danke σας ευχαριστώ આભાર Mèsi poutèt ou Na gode Mahalo תודה Dhanyawaad köszönöm þakka þér
Daalụ terima kasih Go raibh maith agat grazie ありがとう matur nuwun ಧನ್ಯವಾದಗಳು សូមអរគុណអ្នក Kamsahamnida ຂໍຂອບໃຈທ່ານ
Ngiyabonga paldies ačiū ви благодариме misaotra Terima kasih Grazzi Mauruuru Dhanyawaadh Welálin баярлалаа barka
Ahéhee' Dhanyabaad miigwetch manana تشکر از شما dziękuję obrigado ਤੁਹਾਡਾ ਧੰਨਵਾਦ mulțumesc спасибо tapadh leibh хвала
ďakujem hvala Waad ku mahadsan tahay Gracias Asante Tack Salamat rahmat நன்றி ధన్యవాదాలు ขอบคุณ tualumba teşekkür
ederim Спасибі آپ کا شکریہ rahmat cảm ơn bạn Diolch yn fawr אַ דאַנק Balika o ṣeun