Python Fundamentals

Modularity

Robert Smallshire
@robsmallshire
rob@sixty-north.com

Presenter

Austin Bingham
@austin_bingham
austin@sixty-north.com

pluralsight
hardcore developer training

fred

```python
def _int32_to_bytes(i):
    "Convert an integer to four bytes in little-endian format."
    return bytes( (i & 0xff,
                   i >> 8 & 0xff,
                   i >> 16 & 0xff,
                   i >> 24 & 0xff) )
```

```python
def _bytes_to_int32(b):
    "Convert a bytes object containing four bytes into an integer."
    return b[0] | (b[1] << 8) | (b[2] << 16) | (b[3] << 24)
```

jim

```python
def console_card_printer(passenger, seat, flight_number, aircraft):
    output = "| Name: {0}"     \
             "  Flight: {1}"   \
             "  Seat: {2}"     \
             "  Aircraft: {3}" \
             " |".format(passenger, flight_number, seat, aircraft)
    banner = '+' + '-' * (len(output) - 2) + '+'
    border = '|' + ' ' * (len(output) - 2) + '|'
    lines = [banner, border, output, border, banner]
    card = '\n'.join(lines)
    print(card)
    print()
```

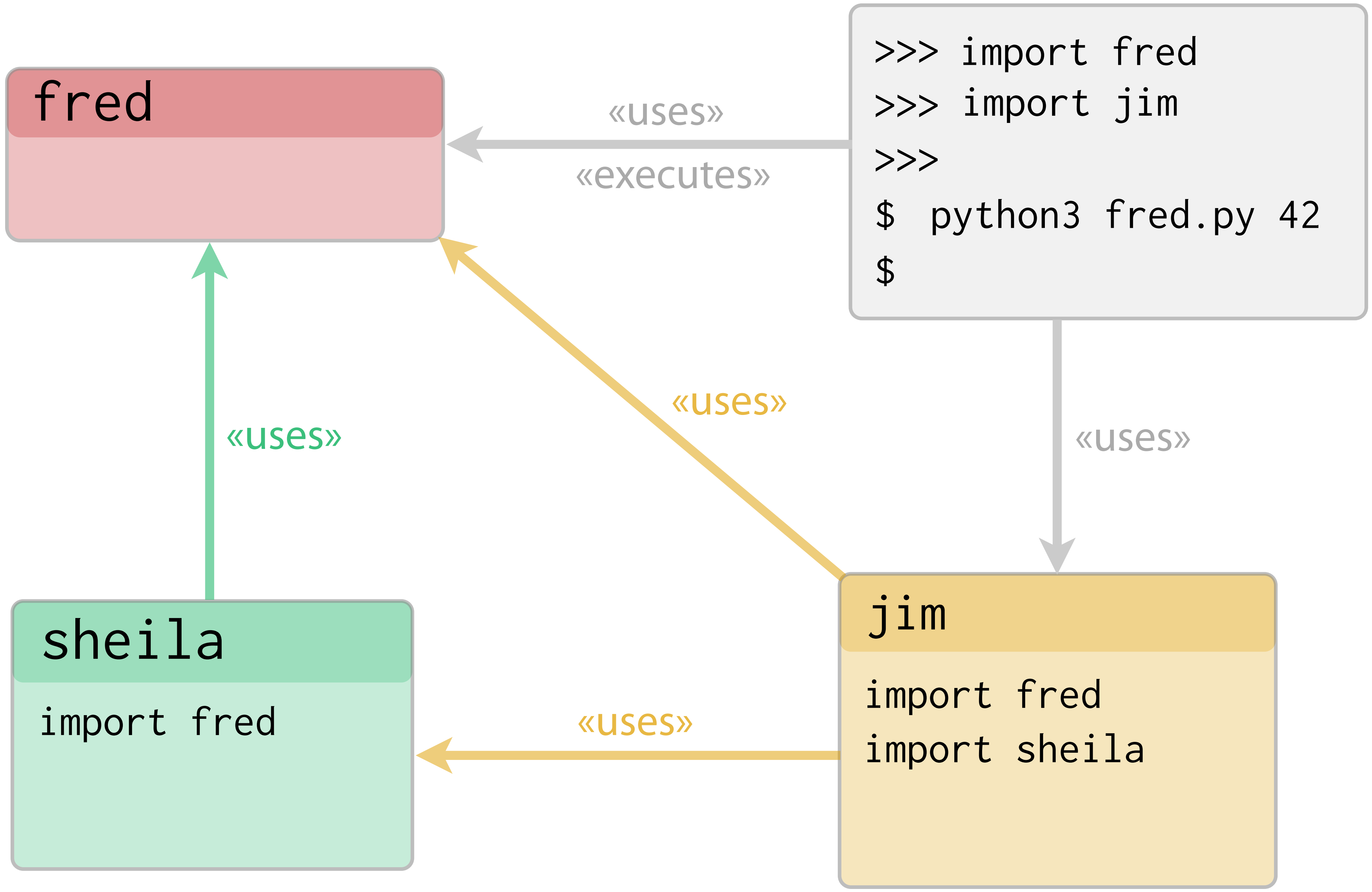
```python
def make_flight():
    f = Flight("BA758", Aircraft("G-EUPT", "Airbus A319",
                          num_rows=22, num_seats_per_row=6))
    f.allocate_seat('12A', 'Guido van Rossum')
    f.allocate_seat('15F', 'Bjarne Stroustrup')
    f.allocate_seat('15E', 'Anders Hejlsberg')
    f.allocate_seat('1C', 'John McCarthy')
    f.allocate_seat('1D', 'Richard Hickey')
    return f
```

sheila

```python
def fetch_words():
    with urlopen('http://sixty-north.com/c/t.txt') as story:
        story_words = []
        for line in story:
            line_words = line.decode('utf8').split()
            for word in line_words:
                story_words.append(word)
    return story_words
```

```python
def print_items(items):
    for item in items:
      print(item)
```

```python
def main():
    url = sys.argv[1]
    words = fetch_words(url)
    print_items(words)
```



```
>>> import fred
>>> import jim
>>>
$  python3 fred.py 42
$
```

```
import fred
```

```
import fred
import sheila
```

Special attributes in Python are delimited by **double underscores**

`__name__`

Evaluates to "`__main__`" or the actual module name depending on how the enclosing module is being used.

The Python Execution Model

When are functions defined?

What happens when a module is imported?

python™

module,
script
or
program?

Python module

Convenient import with API

Python script

Convenient execution from command line

Python program

Perhaps composed of many modules

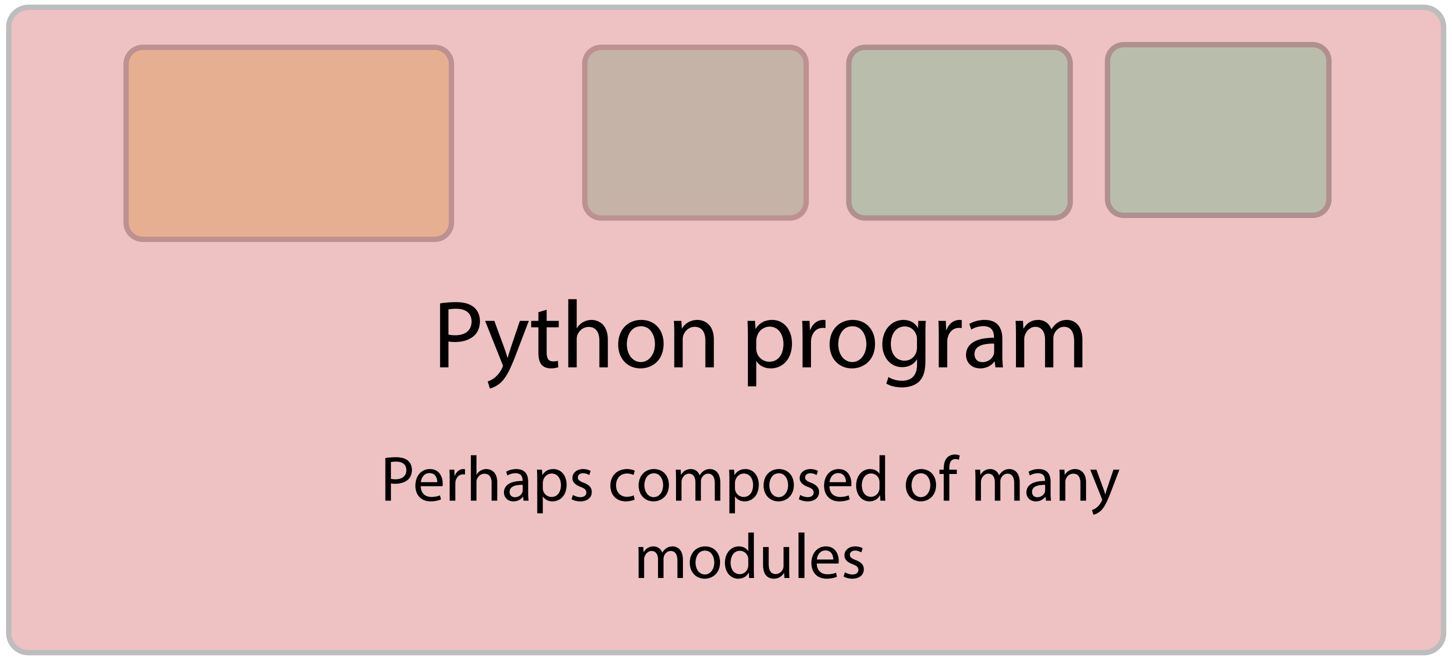
Python module
Python script

Convenient execution from command line

Convenient import with API

Python program

Perhaps composed of many modules



Python program

Perhaps composed of many modules

It doesn't have to be called this!

Setting up a `main()` function with a **command line argument**

Advanced command line argument parsing:

- Python Standard Library: `argparse`
- Many third-party options such as `docopt`

Moment of Zen

Sparse is better than dense

Two between functions
That is the number of lines
PEP eight recommends

```
"""Documenting your code.

Using docstrings.
"""
```

Docstring conventions

- PEP 257 – not widely adopted
- reStructuredText / Sphinx
- Google Python Style Guide

↖ Google it!

# Comments

Windows python™

PyLauncher

- executable is `py.exe` and is on the `PATH`
- associated with `*.py` files
- parses Unix-style shebangs to locate the correct Python interpreter version
- `#!/usr/bin/env python3` works on Windows

Available from Python 3.3

Modularity – Summary

- **Python code is placed in** `*.py` **files called "modules"**
- **Modules can be executed directly with**

```
python module_name.py
```

- **Brought into the REPL or other modules with**

```
import module_name
```

- **Named functions defined with the** `def` **keyword**

```
def function_name(arg1, argn):
```

- **Return from functions using** `return` **keyword with optional parameter**
- **Omitted** `return` **parameter or implicit return at end returns** `None`
- **Use** `__name__` **to determine how the module is being used**
- **If** `__name__ == "__main__"` **the module is being executed**
- **Module code is executed exactly once, on first import**
- `def` **is a *statement* which binds a function definition to a name**

Modularity – Summary

- Command line arguments are accessible through `sys.argv`
- The script filename is in `sys.argv[0]`
- Docstrings are a standalone literal string as the first statement of a function or module
- Docstrings are delimited by triple quotes
- Docstrings provide `help()`
- Comments begin with # and run to the end of the line
- A special comment on the first line beginning #! controls module execution by the program loader