Python Practice Session

Ryan M. Richard

Scientist II and Adjunct Assistant Professor of Chemistry
Ames National Laboratory and Iowa State University
Ames, IA, USA

2025 SIMCODES Bootcamp

Objectives

- Understand the basics of Python software.
- Write a Python package for the Fibonacci sequence.

Notes on Writing Python

- Case-sensitive, e.g., `Print` is not the same as `print`
- Leading spaces are significant (and should always be spaces, **NOT** tabs).
  - The number of spaces is up to you, but you should be consistent.
  - e.g., if you choose to indent two spaces, you should always use two space.
- The type of brackets, braces, etc. are significant.

```
1 # This is a comment.
2 # It gives the reader context.
3 print('hello world')
4 for i in [1, 2, 3]:
5     print(i)
```

What is a Notebook?

- Examples are Colab and Jupyter.
- Notebooks are single files (`*.ipynb` extensions).
- Notebooks are designed for singe use.
  - E.g., tutorials, data analysis, rapid prototyping.
- NOT intended for reuse, or further development.
  - Can force it if you really want to...

What is a Module?

- Python software has three flavors: modules, packages, and libraries.
- A module is a single `*.py` file.
- Modules are building blocks of packages and libraries.
- All the actual effort goes into writing modules.

What is a Package?

- Packages are modules distributed together.
- Directory, `__init__.py` file, and `*.py` files.
- Name of the directory will be the name of the package.
- The `__init__.py` file is usually empty, but is essential because that's how Python determines your directory is a package.
- The `*.py` files are plain text and are where you put the code.

What is a Library?

- Libraries are packages that are distributed together.
- Libraries are usually distributed individually so the terms stop here.
- We won't worry about how to develop a library.

What is a project?

- Disclaimer, may not be a standardized term.
- Modules, packages, and libraries are "just the code."
- Good software also "infrastructure", e.g., documentation, tests, etc.
- We call everything together the "project."

Fibonacci Sequence

- 0, 1, 1, 2, 3, 5, 8, ...
- $F_i = F_{i-2} + F_{i-1}$
- Our goal: write a package for computing F_i.

Setup: Project Structure

- Let's call our project `Fibby`
- In terminal, start by making a directory called `fibby`.
  - Note it is lowercase to avoid file system gotchas.
- What is the terminal command for creating `fibby`?

Setup: Project Structure

- Let's call our project `Fibby`
- In terminal, start by making a directory called `fibby`.
  - Note it is lowercase to avoid file system gotchas.
- What is the terminal command for creating `fibby`?
  - `mkdir fibby`

Setup: Project Structure

- Let's call our project `Fibby`
- In terminal, start by making a directory called `fibby`.
  - Note it is lowercase to avoid file system gotchas.
- What is the terminal command for creating `fibby`?
  - `mkdir fibby`
- All of our project files will live inside this directory.
- Practice: add `fibby/README.md` with contents: `Welcome to Fibby!`
- N.b., paths for this tutorial are given relative to the first `fibby` directory you made.

Setup: Package Structure

- Within a project, Python modules are separated from infrastructure.
- Python modules for a package are stored in a single directory.
- Python uses the directory name as the name of the package.
- Unfortunately this leads to a very common pattern where the name of the project and the name of the package are the same.
- I.e., add a directory `fibby/fibby`

Setup: Making `fibby` a Package

- Python doesn't know `fibby/fibby` is a package.
- How do we tell Python that `fibby` is a package?

Setup: Making `fibby` a Package

- Python doesn't know `fibby/fibby` is a package.
- How do we tell Python that `fibby` is a package?
  - Add a file `__init__.py` to `fibby/fibby`.

Setup: Making the `fibby` Module

- Create a file `fibby/fibby/fibonacci.py`.
- For now, just worry about computing $F_0 = 0$ and $F_1 = 1$.
- Write a function which takes i and returns F_i.

Setup: Making the `fibby` Module

- Create a file `fibby/fibby/fibonacci.py`.
- For now, just worry about computing $F_0 = 0$ and $F_1 = 1$.
- Write a function which takes i and returns F_i.

```
1 def fibonacci(i):
2     if i == 0:
3         return 0
4     if i == 1:
5         return 1
```

Does Fibby work?

- What is the command to run Fibby?

Does Fibby work?

- What is the command to run Fibby?
  - `python3 fibonacci.py`
  - `python3 fibonacci.py 0`
  - `python3 fibonacci.py 1`

Does Fibby work?

- What is the command to run Fibby?
  - `python3 fibonacci.py`
  - `python3 fibonacci.py 0`
  - `python3 fibonacci.py 1`
- This was a trick question. We only **defined** a function `fibonacci`. We never called it.
- Very common error...

Calling Fibby.

- Add a call to `fibonacci`.
- Now how do we run it?

```
1 def fibonacci(i):
2     if i == 0:
3         return 0
4     if i == 1:
5         return 1
6
7
8 fibonacci(0)
```

Calling Fibby.

- Add a call to `fibonacci`.
- Now how do we run it?
  - `python3 fibonacci.py`

```
1 def fibonacci(i):
2     if i == 0:
3         return 0
4     if i == 1:
5         return 1
6
7
8 fibonacci(0)
```

Calling Fibby.

- Add a call to `fibonacci`.
- Now how do we run it?

```
python3 fibonacci.py
```

- This was a trick question. We called the function, but didn't **print** the value.
- Again common error. Remember computers do **exactly** what you ask them to do.

```
1 def fibonacci(i):
2     if i == 0:
3         return 0
4     if i == 1:
5         return 1
6
7
8 fibonacci(0)
```

Calling Fibby.

- Add a call to `fibonacci`.
- Now how do we run it?

```
python3 fibonacci.py
```

- This was a trick question. We called the function, but didn't **print** the value.
- Again common error. Remember computers do **exactly** what you ask them to do.
- Correct code.
- How would we print F_1?

```
1 def fibonacci(i):
2     if i == 0:
3         return 0
4     if i == 1:
5         return 1
6
7
8 print(fibonacci(0))
```

Development Notes

- When you develop software you typically don't call your function, class, etc. in the module you define it in.
- We did this just as a "sanity check."
- Want to "separate concerns."
- (Go ahead and delete the `print(fibonacci(0))` line).
- Modern software development is test-driven. So lets write a test.

```
1 def fibonacci(i):
2     if i == 0:
3         return 0
4     if i == 1:
5         return 1
```

Testing Fibby

- Create `fibby/tests`
- Create `fibby/tests/__init__.py`
- Create `fibby/tests/test_fibonacci.py`
- Run: `pytest` in `fibby` directory.
- At this point

```
1 from fibby.fibonacci import fibonacci
2
3 assert fibonacci(0) == 0
4 assert fibonacci(1) == 1
```

Notes on Structure

- We now have a bare-bones Python project.
- There's actually theory behind why projects are set up this way.
- If you try to get creative you will likely run into import errors.
- Until you are a Python expert, just stick with this pattern.

```
# Project Structure
# project_name/
# |--- package_name/
# |     |---__init__.py
# |     |---module1_name.py
# |     |---module2_name.py
# |--- tests/
# |     |--- __init__.py
# |     |--- test_module1_name.py
# |     |--- test_module2_name.py
```

Towards Arbitrary F_i

- Our code only works for $i \leq 1$.
- Let's generalize it.
- Many possible implementations.
- General tip: get something that works first.
- Easy implementation: fill a list.

Towards Arbitrary F_i

- Our code only works for $i \leq 1$.
- Let's generalize it.
- Many possible implementations.
- General tip: get something that works first.
- Easy implementation: fill a list.

```
1 def fibonacci(i):
2     seq = [0, 1]
3
4     while len(seq) <= i:
5         seq.append(seq[-2] + seq[-1])
6
7     return seq[i]
```

Performance

- Our implementation is not optimal.
- How bad do you think it is?
- Try asking for $F_{10} = 55$

Performance

- Our implementation is not optimal.
- How bad do you think it is?
- Try asking for $F_{10} = 55$
- $F_{54} = 86267571272$

Performance

- Our implementation is not optimal.
- How bad do you think it is?
- Try asking for $F_{10} = 55$
- $F_{54} = 86267571272$
- $F_{502} =$ 365014740723634211012237077 906479355996081581501455497 852747829366800199361550174 096573645929019489792751

Performance

- Our implementation is not optimal.
- How bad do you think it is?
- Try asking for $F_{10} = 55$
- $F_{54} = 86267571272$
- $F_{502} = 365014740723634211012237077906479355996081581501455497852747829366800199361550174096573645929019489792751$
- $F_{5008} = 182\cdots$

Performance

- Our implementation is not optimal.
- How bad do you think it is?
- Try asking for $F_{10} = 55$
- $F_{54} = 86267571272$
- $F_{502} = 365014740723634211012237077906479355996081581501455497852747829366800199361550174096573645929019489792751$
- $F_{5008} = 182\cdots$
- $F_{25000} =$??? (string conversion error)

Performance: Lesson

- Pro tip: do NOT pre-maturely optimize your code.
- Our naive Fibonacci code can compute F_{25000} in under a second!
- Code readability/maintainability should be the priority for the first pass.
- Only start optimizing code when it becomes a bottleneck.

Documentation

- I recommend adding documentation as the last step before a PR.
  - If you add it too early, you'll often find that you need to change it because you added/removed a parameter, changed the type of a parameter, etc.
- In Python we document functions with "docstrings".
- Docstrings are free-form, but in science we usually follow "numpydoc" style.
  - https://www.geeksforgeeks.org/python-docstrings/

```
def fibonacci(i):
    '''
    This is a summary line. It should be
    about a sentence long.

    This is the extended description.
    Note that the entire docstring is
    indented as far as the quotes we used.

    Parameters
    ----------
    i : int
        Which fibonacci number you want.
        i == 0 is 0. i == 1 is 1.
        i == 2 is 1. i == 3 is 2. Etc.

    Returns
    -------
    int
        The requested fibonacci number
    '''

    # Code would go here
```

Generalizing

- While real-world problems are harder, they're tackled basically the same way.
- Write code units (e.g., function, class, libraries) for each task.
- Test the code units as you write them.
- If code unit gets too big, split it.
- Document before committing.

Misc.

- Software engineering is ultimately a soft science.
- Should this be a function? A class? A library? Subjective.
- Ultimately, any solution that gets the right answer (for the right reasons) is "correct".
- Easier to test small code units. I recommend erring on the side of too small, rather than too large.
- A lot like "showing your work" in math. Each code unit shows a step.
- As you get more comfortable you can start "skipping steps."

Debugging

- In practice, most of coding is debugging.
- Something you get better at with practice.
- Need to trace the logic of the program and see where if fails.
- Usually introduced through `print` statements.
  - Basically print values until you find the first one that is wrong.
- Tools called "debuggers" that can do this (and more) for you.

Practice

- Fibonacci sequence prototypical example of recursion.
  - Recursion: Function `fibonacci` calls itself.
- Goal: write the call for F_i in terms of the calls for F_{i-2} and F_{i-1}.
- Hint: in terminal `ctrl + c` kills the current running process.
  - Common recursion problem is running forever because function just keeps calling itself.

Acknowledgements

- NSF for funding the REU.
- ISU and Ames National Lab