Best Practices in Research Software Engineering



UNIVERSITY OF MARYLAND

FEARLESSLY FORWARD

Research Software Engineering at UMD

Research Software Engineering

Research Software Engineering Consultation service catalog entry

We offer:

- Workshops
- Office hours or targeted help with research code
- Review, debugging, design, security, performance, project management, and many other aspects of software engineering
- Longer-term support for funded projects

Part 1: Data management, project planning & organization

Resources

Intermediate Research Software Development course -- Carpentries

Research Software Engineering with Python course -- The Alan Turing Institute

Research Software Engineering with Python online book -- Irving, et. al.

Data Management

UMD Guide Section 1 -- Data Management

Good Enough Practices in Scientific Computing -- Data Management

For reference:

Alan Turing Institute Section 10

Project Planning & Design

UMD Guide Section 2 -- Project Planning & Design

Carpentries Course Section 3 -- Requirements and Design

For reference:

Alan Turing Institute Section 6.6 -- Project Mangement

Alan Turing Institute Section 7 -- Design

Project Organization

UMD Guide Section 4 -- Project Organization

Research Software Engineering with Python (online book) Chapter 1 -- excellent

Setup

Follow the Installation Instructions in the Carpentries course

Follow Section 1.1 to download the software project (nice-to-have).

Follow Section 1.2 to set up a virtual environment:

(Linux:)

```
python3 -m venv venv
source venv/bin/activate
```

Exercise

Research Project: We want to explore how the popularity of baby names in the United States changed by decade.

Historical baby name data from Social Security Administration

- Download the zip file and extract it on your laptop
- StateReadMe.pdf (within the zip) explains the data structure
- Also look at some of the files

Exercise

Description of the data:

"For each of the 50 states and the District of Columbia we created a file called SC.txt, where SC is the state's postal abbreviation.

Each record in a file has the format: 2-digit state code, sex (M = male or F = female), 4-digit year of birth (starting with 1910), the 2-15 character name, and the number of occurrences of the name. Fields are delimited with a comma. Each file is sorted first on sex, then year of birth, and then on number of occurrences in descending order..."

Exercise: Data

```
VT,F,1910,Mary,45
VT,F,1910,Helen,34
VT,F,1910,Dorothy,32
VT,F,1910,Alice,25
VT,F,1910,Margaret,25
VT,F,1910,Mildred,24
VT,F,1910,Florence,23
VT,F,1910,Ruth,23
VT,F,1910,Elizabeth,22
VT,F,1910,Marion,22
VT,F,1910,Doris,18
VT,F,1910,Irene,18
VT,F,1910,Evelyn,15
VT,F,1910,Hazel,15
VT,F,1910,Gladys,14
VT,F,1910,Grace,13
VT,F,1910,Lillian,13
```

```
VT,M,1996,Drew,5
VT,M,1996,George,5
VT,M,1996,Gregory,5
VT,M,1996,Henry,5
VT,M,1996,Marcus,5
VT,M,1996,Max,5
VT,M,1996,Nicolas,5
VT,M,1996,Oliver,5
VT,M,1996,Owen,5
VT,M,1996,Parker,5
VT,M,1996,Quinn,5
VT,M,1996,Silas,5
VT,M,1996,Wyatt,5
VT,M,1997,Jacob,81
VT,M,1997,Nicholas,69
VT,M,1997,Ryan,67
VT,M,1997,Tyler,61
VT,M,1997,Matthew,60
VT,M,1997,Joshua,56
VT,M,1997,Kyle,55
VT,M,1997,Michael,52
```

Exercise: Data

Things to note:

- The data is broken down by state, but we want data pertaining to the entire U.S.
- The data contains entries for every year, but we want data for each decade.
- There are a large number of baby names each year. What might be some good ways to show changes over time?
- How should we process M and F names?
- What else?

Exercise: Requirements

Research Project: We want to explore how the popularity of baby names in the United States changed by decade.

What are the project requirements?

Exercise: Data

Does the data need to be cleaned? Is it messy?

What data validation might we need?

What manipulation of the raw data might we need or want before the program does data analysis?

Exercise: Software Design

What software components will we need?

What information will be passed between the components?

What mechanism(s) will be used to pass that information? (direct calling of functions with parameters, reading/writing to disk, sending/receiving messages, ...)

Exercise: Software Design

What if we wanted to design a front-end to allow users to access the data?

What if we wanted to add data visualization functionality?

How would these affect the design?

What other features could we add to our design?

Exercise: Set up Your Project

Now we have the raw data, requirements, and design. Create a new project with:

- Organized directory structure. Consider using the project from the Carpentries course as a template for your project.
- Raw data saved within the project
- Skeleton code with functions/objects that will implement the design.
  - Add commentary to each module/function/object to explain what it will do, as well as any other important details
  - Specify all inputs and outputs
- **BONUS**: Write code to read in the data, validate it, pre-process it, and store the pre-processed data.

Part 2: Code Quality, Documentation & Logging

Historical Baby Name Analysis

Baby Name Data

Historical baby name data from Social Security Administration

51 CSV files, corresponding to data from each state and the District of Columbia.

Baby Name Data

```
VT,F,1910,Mary,45
VT,F,1910,Helen,34
VT,F,1910,Dorothy,32
VT,F,1910,Alice,25
VT,F,1910,Margaret,25
VT,F,1910,Mildred,24
VT,F,1910,Florence,23
VT,F,1910,Ruth,23
VT,F,1910,Elizabeth,22
VT,F,1910,Marion,22
VT,F,1910,Doris,18
VT,F,1910,Irene,18
VT,F,1910,Evelyn,15
VT,F,1910,Hazel,15
VT,F,1910,Gladys,14
VT,F,1910,Grace,13
VT,F,1910,Lillian,13
```

```
VT,M,1996,Drew,5
VT,M,1996,George,5
VT,M,1996,Gregory,5
VT,M,1996,Henry,5
VT,M,1996,Marcus,5
VT,M,1996,Max,5
VT,M,1996,Nicolas,5
VT,M,1996,Oliver,5
VT,M,1996,Owen,5
VT,M,1996,Parker,5
VT,M,1996,Quinn,5
VT,M,1996,Silas,5
VT,M,1996,Wyatt,5
VT,M,1997,Jacob,81
VT,M,1997,Nicholas,69
VT,M,1997,Ryan,67
VT,M,1997,Tyler,61
VT,M,1997,Matthew,60
VT,M,1997,Joshua,56
VT,M,1997,Kyle,55
VT,M,1997,Michael,52
```

Baby Name Analysis

Objective: We want to analyze the frequency of baby names historically for the entire U.S. (all 50 states + D.C.). For this workshop, each time the program runs, it will analyze the baby names for one year.

The All-in-one-file Program

All-in-one-file: main_one_file.py

```
a = sys.argv[1:]
b = None
c = None
d = 0
while d < len(a):
    if a[d] in ["-n", "--name"]:
        if d + 1 < len(a):
            b = a[d + 1]
        d += 2
    elif a[d] in ["-y", "--year"]:
        if d + 1 < len(a):
            c = a[d + 1]
        d += 2
    else:
        d += 1
if b is None or c is None:
    raise SystemExit("Invalid argument(s).")
try:
    c = int(c)
except:
    raise SystemExit("Invalid year.")
```

All-in-one-file: main_one_file.py

```
e = []
f = os.path.join(os.path.dirname(os.path.dirname(__file__)), "data")
g = sorted(os.listdir(f))
h = 0
while h < len(g):
    i = g[h]
    if i.endswith(".TXT"):
        j = open(os.path.join(f, i), "r", encoding="utf-8")
        try:
            for k in j:
                k = k.strip()
                if k != "":
                    l = k.split(",")
                    if len(l) == 5:
                        e.append(l)
        finally:
            j.close()
    h += 1
```

All-in-one-file: main_one_file.py

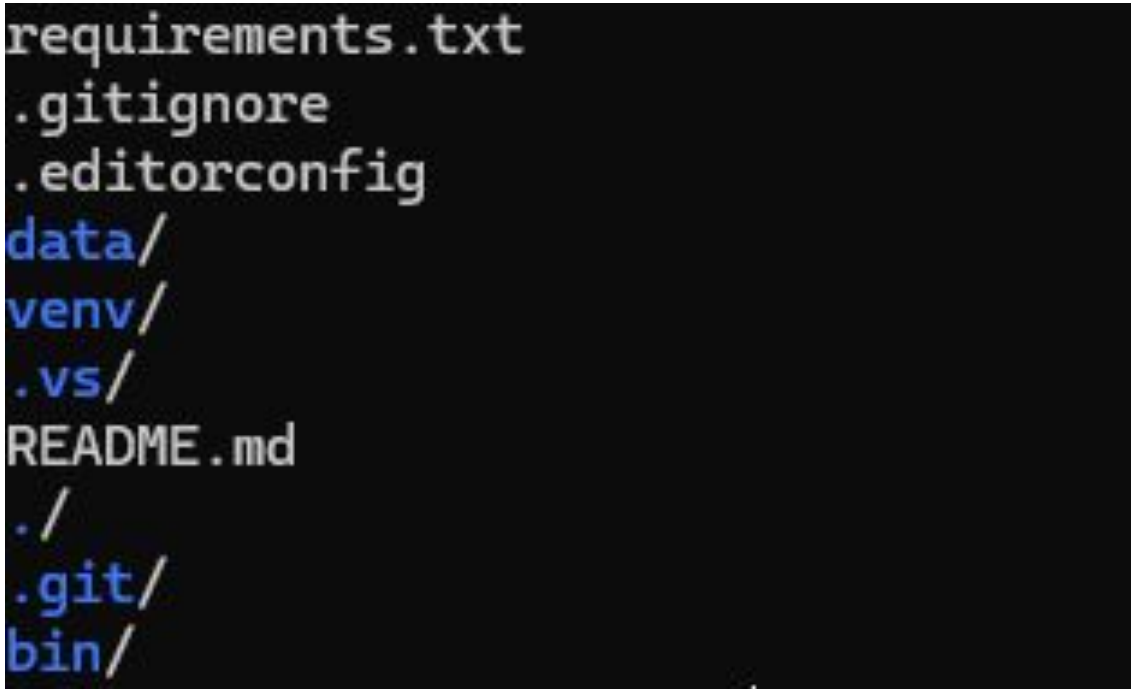
```
m = []
n = 0
while n < len(e):
    o = e[n]
    if o[3] == b and int(o[2]) == c:
        m.append(o)
    n += 1
print("State Count")
p = 0
while p < len(m):
    q = m[p]
    print(q[0] + "    " + q[4])
    p += 1
```

Discussion: All-in-one-file

- What do you notice about this code?
- Imagine you started working in a lab and you were given this code. You were asked to learn how to run it and add new functionality to it. What challenges would you face?
- How could the code be improved to make it easier to understand, maintain, and expand on?

A Different Approach

Project Structure



Files/Classes

- main.py: The entry point. Handles overall program flow.
- data_reader.py (DataReader class): Reads in the data, stores it, and returns the data in a structured format.
- data_analyzer.py: provides data analysis functionality.
- name_record.py (NameRecord class): a structured way of storing one row of CSV data.

main.py

```python
def main(
    name: str,
    year: int
):
    reader = DataReader("data")

    print("Getting and filtering records.")
    all_records = reader.get_all_records()
    filtered_records = filter_records(all_records, name=name, year=year)
    print("\nState Count")
    for record in filtered_records:
        print(f"{record.state}    {record.count}")

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

data_reader.py

```python
class DataReader:
    """
    Reads baby-name records from CSV-formatted .TXT files.

    Expected row format:
        AK,F,1910,Mary,14

    Fields:
        state, sex, year, name, count
    """

    def __init__(self, directory: str | Path, pattern: str = "*.TXT") -> None:
        self.directory = Path(directory)
        self.pattern = pattern

        if not self.directory.exists():
            raise FileNotFoundError(f"Directory does not exist: {self.directory}")
        if not self.directory.is_dir():
            raise NotADirectoryError(f"Not a directory: {self.directory}")

        print(f"Loading data from CSV files.")
        self.records = list(self._iter_records())
        print(f"Loaded {len(self.records)} records.")
```

data_analyzer.py

```
"""
Provides analysis methods for name record data.
"""

def filter_records(
    name_records: List[NameRecord],
    *,
    state: Optional[str] = None,
    sex: Optional[str] = None,
    year: Optional[int] = None,
    name: Optional[str] = None,
) -> List[NameRecord]:
    """
    Return records matching optional filters.

    Args:
        name_records (List[NameRecord]): Name records apply filtering to.
        state: Optional[str]: Filter for state.
        sex: Optional[str]: Filter for sex.
        year: Optional[int]: Filter for year.
        name: Optional[str]: Filter for name.
    """
    results: List[NameRecord] = []
```

Discussion: The second approach

- What do you notice about the code for the second approach?
- Imagine you started working in a lab and you were given this code. You were asked to learn how to run it and add new functionality to it. Would you prefer getting this code over the all-in-one-file code?
- What factors make the code for the second approach easier to understand, maintain, and expand on?

Code Quality



Code Quality

In this section, we'll be following The Alan Turing Institute's *Research Software Engineering with Python* Chapter 7.

Code Quality

Software Construction (Section 7.0):

"For a given piece of code, there exist several different ways one could write it:

- Choice of variable names
- Choice of comments
- Choice of layout"

"Remember you're programming for humans as well as computers"

Code Quality

Comments (Section 7.1)

"Why comment?

You’re writing code for people, as well as computers."

Code Quality

Coding Conventions (Section 7.2):

Layout

Naming Conventions

Newlines

Syntax Choices

Code Quality: Coding Conventions

```python
def calculate_area(width, height):
    """Return area of a rectangle."""
    return width * height
```

```python
def calculateArea(width,height): return width*height
```

Code Quality: Coding Conventions

Coding Conventions (Section 7.2):

Aim for consistency and readability.

There are published standards for various programming languages.

Consider adopting one.

Code Quality: More Resources

https://third-bit.com/py-rse/style.html

Documentation

Documentation: READMEs

READMEs: In this section, we'll be following Cornell Data Service's Writing READMEs for Research Code & Software.

A README will be the first place a person goes to learn about the project. It should have what they need to run and/or start working on the project.

Documentation: READMEs

A README should give:

- Project overview / purpose
- Installation instructions
- Usage instructions
- License
- Contact info
- Acknowledgements & references
- Scientific background (references, notes about data, etc.) if applicable

Documentation: READMEs

A README will be the first place a person goes to learn about the project. It should have what they need to run and/or start working on the project.

A README should give:

- Project overview / purpose
- Installation instructions
- Usage instructions
- License
- Contact info
- Acknowledgements & references

Example README

A README from a real github project (DualSPHysics):

Instructions for regular users

If you only want a copy of DualSPHysics to create and run cases in your system, you probably want the full DualSPHysics package from the official website. There you will find documentation and packages of different versions for different Operating Systems.

Example README (continued)

Instructions for developers

If you are a developer and want to use this code check the following guides.

Take into account that for pre- and post-processing you will need to use GenCase and the different post-processing tools included in the main DualSPHysics package, here. If you compile your own DualSPHyiscs version just overwrite the one in the package.

Developing a modified version to fit your own needs.

You can fork this repository and change or add anything you want. Keep in mind that your changes will not be taken into account into the main versions. If your objective is to implement your changes/improvements to the main code, check the next section.

Example README (continued)

Building the project

Microsoft Windows

This application is being developed in Visual Studio Community 2022 since it is free and compatible with CUDA 12.3 (download web). The repository contains project files.

Make sure that you install the CUDA SDK beforehand if you want to compile the GPU version, and configure the Visual Studio project to point to the CUDA libraries directory to compile (now prepared for CUDA 12.3).

You can also use the Makefile. It is possible that you'll need to edit it. Check the GNU/Linux guide on how to compile if you're using the makefile, as it is mostly the same, just installing the things manually by yourself.

GNU/Linux

Documentation: Docstrings

Docstrings: In this section, we'll be following the online book *Research Software Engineering with Python* Appendix G.

Documentation: Docstrings

Docstring (documentation string) -- a string at the beginning of a module, function, class, or method that describes what that object does.

Good docstrings include:

"The name and purpose of every public class, function, and constant in our code.
The name, purpose, and default value (if any) of every parameter to every function.
Any side effects the function has.
The type of value returned by every function.
What exceptions those functions can raise and when."

Documentation: Docstrings

Docstring example - Google style

```
def calculate_age(birthday):
    """Find current age from birth date.

    Args:
        birthday (str): birth date.

    Returns:
        Age in years.

    Raises:
        ValueError: If birthday not in YYYY-MM-DD format.
    """
```

Logging & Error Handling

Logging & Error Handling

We will be following this resource for the next few slides:

Research Software Engineering with Python online book,

Handling Errors chapter.

Logging & Error Handling

Modern languages raise exceptions when something goes wrong. You can raise exceptions in your code for error cases. The libraries you call can also raise exceptions. Use try/except statements to handle exceptions.

From the book:

```
for fname in args.infiles:
    try:
        with open(fname, 'r') as reader:
            update_counts(reader, word_counts)
    except ValueError as e:
        print(f'{fname} is not a CSV file.')
        print(f'ValueError: {e}')
```

Logging & Error Handling

Try to make your error messages helpful.

OSError: Unsupported file type.

is better than

OSError: Something went wrong, try again.

Logging & Error Handling

A logging framework is useful for printing messages at runtime.

Messages can be logged at different levels depending on their importance:

- DEBUG: very detailed information used for localizing errors.
- INFO: confirmation that things are working as expected.
- WARNING: something unexpected happened, but the program will keep going.
- ERROR: something has gone badly wrong, but the program hasn't hurt anything.
- CRITICAL: potential loss of data, security breach, etc.

Logging & Error Handling

Basic logging setup for python:

```
import logging
logging.basicConfig(
    level=logging.DEBUG,
    filename='logging.log',
    format="%(asctime)s %(levelname)s: %(message)s"
)

logging.debug('This is for debugging.')
logging.info('This is just for information.')
```

Exercise

Setup

- Follow these instructions to install python3 if you don't have it already.
- Download and unzip rse-workshop-baby-names.zip.
- Under rse-workshop-baby-names/, open README.md and follow the instructions to set up your virtual environment and run the program.

Exercise

Add a new function to data_analyzer.py. The new function should accept as an input the aggregated records (U.S.-wide for one year) and return the top 10 baby names and the count for each name.
Use code style consistent with existing functions.
Add docstring consistent with other functions.
Add logging.

Bonus: Add the code in main.py to call the new function and display the results.

Part 3: Code Review, Testing & Debugging

Introduction

Most Common Baby Names: Code

```
def most_common_names(
    input_data: Dict[str, int],
    n: int
) -> Dict[str, int]:
    """
    Return top n names in a set of data.

    Args:
        input_data (Dict[str, int]): Input data for one or more years.
            key (str): Name
            value (int): number of counts
        n (str): Number of names to return

    Return a dictionary containing the n number of entries with the highest counts.
            key (str): Name
            value (int): number of counts
    """
    result = sorted(input_data.items(), key=lambda item: item[1], reverse=True)[1:n+1]
    return dict(result)
```

Most Common Baby Names: Output

```
Getting and logging 10 most common names for year 1999.
Michael      33990
Matthew      30431
Joshua      27280
Emily      26542
Nicholas      25644
Christopher      25621
Andrew      23873
Tyler      23425
Joseph      23239
Daniel      22740
```

Most Common Baby Names

Are the code and output data correct?

How can we tell?

Most Common Baby Names: Correct Output

```
Getting and logging 10 most common names for year 1999.
Jacob      35395
Michael      33990
Matthew      30431
Joshua      27280
Emily      26542
Nicholas      25644
Christopher      25621
Andrew      23873
Tyler      23425
Joseph      23239
```

Most Common Baby Names: Code Bug

```
def most_common_names(
    input_data: Dict[str, int],
    n: int
) -> Dict[str, int]:
    """
    Return top n names in a set of data.

    Args:
        input_data (Dict[str, int]): Input data for one or more years.
            key (str): Name
            value (int): number of counts
        n (str): Number of names to return

    Return a dictionary containing the n number of entries with the highest counts.
            key (str): Name
            value (int): number of counts
    """
    result = sorted(input_data.items(), key=lambda item: item[1], reverse=True)[1:n+1]
    return dict(result)
```

Should be *[:n]*

Most Common Baby Names: Code Bug

What can we do to try to prevent bugs like this from going undetected?

Code Review

Code Review

In this section, we will draw from The Carpentries' *Intermediate Research Software Developement* Section 4.1.

Code Review: What is it?

"Code review is a team software quality assurance practice where team members look at parts of the codebase in order to improve their code's readability, understandability, quality and maintainability."

Code Review: Benefits

- Catch mistakes early. Before they make it to out into the world.
- Knowledge sharing. Allows others on the team to learn about the code.
- "Having to explain code to someone else clarifies your rationale and design decisions in your mind too."

Code Review: My Personal Experience

- Someone may catch errors you missed.
- Others may have a better idea for a more efficient or more maintainable software design.
- Opportunity to learn and teach coding practices and subject matter.
- Knowledge sharing. The more everyone knows about the code, the better it is for the team.

Code Review: Tool-assisted code review



Code Review: Defining a Review Process

"It is important to agree on a set of best practices and establish a code review process in a team to help to sustain a good, stable and maintainable code for many years."

Code Review: Defining a Review Process

Decisions the team should make:

- "Whether all changes need to go through code review
- What technologies you are going to use to manage the review process
- How quickly you expect someone to review the code once a PR has been raised
- How long should be spent reviewing code
- What kind of issues are (and are not) appropriate to raise in a PR
- How will someone know when they are expected to take action (e.g. review a PR)."

Code Review: Defining a Review Process

Recommendations:

- Establish a process that you think will work for your team
- Use it consistently from the outset of the project

Code Review: More Resources

https://third-bit.com/py-rse/style.html#style-review

Testing

Testing

In this section, we are going to make use of The Alan Turing Institute's *Research Software Engineering with Python* Section 5.

Testing

Reasons to test:

- "peace of mind -- tests (should) ensure code is correct
- runnable specification -- best way to let others know what a function should do and not do
- code structure / modularity -- since the code is designed for at least two situations
- easier to modify since results can be tested"

Testing: Levels of Testing

Unit testing - tests the "smallest logical block of work." For example, a python module or class.

They should be "re-run regularly (eg every git commit)." In order to be run idependently, the tests have to be written so that they do not rely on output from other software units.

Testing: Levels of Testing

Integration Testing or Functional Testing - tests "all components together / whole program."

These "can be run less often."

Testing: Writing Tests

"When writing new code... always start by creating tests at the smallest scale (unit tests).

"If a unit test is too complicated to write, then consider adjusting your production code (possibly by breaking it down into smaller, individually testable functions). Ensuring that your production code is easy to test is a healthy habit."

Testing: Writing Tests

"Very difficult to create unit-tests for existing code. Instead we make a regression test" to test functionality. Very similar to an integration test or functional test.

Testing: Writing Tests

We cannot test all possible inputs/scenarios. How to choose test cases?

- "Positive tests: code should give correct answer with various inputs
- Negative tests: code should behave appropriately given invalid inputs"
- Boundary cases or edge cases

Testing: Writing Tests

List of test cases and test code can also be reviewed to get input from the team. Helps ensure test coverage is sufficient and that the tests do what they intend to do.

Testing: Testing Frameworks

Testing frameworks exist in most languages.

You can leverage these frameworks to avoid writing your own test driver code.

"pytest is a recommended python testing framework."

Testing: More Resources

https://third-bit.com/py-rse/testing.html

Debugging



Debugging

In the Logging & Error Handling section of our workshops, we covered:

Research Software Engineering with Python online book,

Handling Errors chapter

Debugging

For logging and error handling, we talked about:

- Logging during normal program flow
- Logging error conditions
- Throwing and catching exceptions
- Logging tracebacks

These are extremely useful for debugging problems.

Baby Names Project: Invalid CSV Data



Baby_names project: Invalid CSV data

```
2026-04-23 13:30:12,427 INFO: Loading data from files.
2026-04-23 13:30:12,428 INFO: Loading data from CSV files.
2026-04-23 13:30:14,683 ERROR: Exception occurred while loading data: Invalid row in data/FL.TXT at line 1: expected 5 fields,
 got 4 -> ['FL', 'F', '1910', '']
Traceback (most recent call last):
  File "/home/wolkm/rse_workshops/rse-workshop-baby-names/bin/main.py", line 32, in main
    reader = DataReader("data")
             ^^^^^^^^^^^^^^^^^^
  File "/home/wolkm/rse_workshops/rse-workshop-baby-names/bin/data_reader.py", line 31, in __init__
    self.records = list(self._iter_records())
                   ^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/wolkm/rse_workshops/rse-workshop-baby-names/bin/data_reader.py", line 46, in _iter_records
    yield from self._read_file(file_path)
  File "/home/wolkm/rse_workshops/rse-workshop-baby-names/bin/data_reader.py", line 66, in _read_file
    raise ValueError(
ValueError: Invalid row in data/FL.TXT at line 1: expected 5 fields, got 4 -> ['FL', 'F', '1910', '']
~
~
```

Baby_names project: Invalid CSV data

What if we didn't have such detailed logging in our code?

What would we have to do to track down the source of the problem?

Remember the problem is caused by one malformed line in one of 51 CSV data files.

Using a Debugger

The Alan Turing Institute's Research Software Engineering with Python Section 5.5 describes debuggers and how to use them.

Using a Debugger

"Debuggers are programs that can be used to test other programs. They allow programmers to suspend execution of the target program and inspect variables at that point."

Using a Debugger

With a debugger, you can:

- Set a breakpoint
- Step line by line through the code
- Examine variable values, function parameters, etc.
- Examine the state of the program when a crash occurred

Python Debugger

python3 **-m pdb** my_program.py

Python Debugger

```
-> from ast import Try
(Pdb) b data_analyzer.py:102
Breakpoint 1 at /home/wolkm/rse_workshops/rse-workshop-baby-names/bin/data_analyzer.py:102
(Pdb) c
> /home/wolkm/rse_workshops/rse-workshop-baby-names/bin/data_analyzer.py(102)most_common_names()
-> return dict(result)
(Pdb) result[0]
('Jacob', 35395)
(Pdb) result[1]
('Michael', 33990)
(Pdb) result[2]
('Matthew', 30431)
(Pdb)
```

Back to Our Example: Baby Names

Most Common Baby Names: Code Bug

```
def most_common_names(
    input_data: Dict[str, int],
    n: int
) -> Dict[str, int]:
    """
    Return top n names in a set of data.

    Args:
        input_data (Dict[str, int]): Input data for one or more years.
            key (str): Name
            value (int): number of counts
        n (str): Number of names to return

    Return a dictionary containing the n number of entries with the highest counts.
            key (str): Name
            value (int): number of counts
    """
    result = sorted(input_data.items(), key=lambda item: item[1], reverse=True)[1:n+1]
    return dict(result)
```

Should be *[:n]*

Most Common Baby Names: Output

```
Getting and logging 10 most common names for year 1999.
Michael      33990
Matthew      30431
Joshua      27280
Emily      26542
Nicholas      25644
Christopher      25621
Andrew      23873
Tyler      23425
Joseph      23239
Daniel      22740
```

Most Common Baby Names: Code Bug

Could any or all of the techniques we talked about help to ensure that this bug is caught before the code is used for data analysis?

- Code review?
- Unit testing?
- Functional or integration testing?
- Debugging?

Part 4: Reproducibility & Keeping Track of Changes

Reproducibility

Automating Analysis

For this topic, we will follow the online book *Research Software Engineering with Python*, Chapter 9.

Automating Analysis

"It’s easy to run one program to process a single data file, but what happens when our analysis depends on many files, or when we need to re-do the analysis every time new data arrives? What should we do if the analysis has several steps that we have to do in a particular order?

"If we try to keep track of this ourselves, we will inevitably forget some crucial steps and it will be hard for other people to pick up our work. Instead, we should use a build manager to keep track of what depends on what and run our analysis programs automatically. These tools were invented to help programmers compile complex software, but can be used to automate any workflow."

Automating Analysis with Make

"When the user runs Make, the program checks all of the rules in the Makefile and runs the commands needed to update any that are out of date."

Filesystem timestamps are used by Make to determine if targets are out-of-date.

Automating Analysis with Make

Example (simple) Makefile:

```
# Regenerate results for "Moby Dick"
results/moby_dick.csv : data/moby_dick.txt
    python bin/countwords.py \
        data/moby_dick.txt > results/moby_dick.csv
```

Automating Analysis with Make

```
# Regenerate all results.
all : results/moby_dick.csv results/jane_eyre.csv

# Regenerate results for "Moby Dick"
results/moby_dick.csv : data/moby_dick.txt $(COUNT)
    $(RUN_COUNT) data/moby_dick.txt > results/moby_dick.csv

# Regenerate results for "Jane Eyre"
results/jane_eyre.csv : data/jane_eyre.txt $(COUNT)
    $(RUN_COUNT) data/jane_eyre.txt > results/jane_eyre.csv

# Remove all generated files.
clean :
    rm -f results/*.csv
```

Automating Analysis with Make

To run Make,

- change to directory containing Makefile
- Execute command: **make** or **make all**
- Execuate command: **make clean**, for example, to run a specific target in the Makefile

Automating Analysis: Other Tools

There are many alternative build tools. One example is **Snakemake** for Python.

Managing Dependencies

For this section, we will mostly follow The Alan Turing Institute's *Research Software Engineering with Python*, Section 6.2.

Managing Dependencies

"To install dependencies from a requirements.txt file do the following:

pip install -r requirements.txt"

Managing Dependencies: Pinning

Example requirements.txt:

geopy==2.2.0

imageio==2.19.3

matplotlib==3.5.2

numpy==1.23.0

requests==2.28.1

Managing Dependencies

"It is good practice to specify the versions of our dependencies and even better to specify version ranges. For example, if we have only tested our package on pandas version 1.1.2, we could put pandas==1.1.2 or pandas>=1.1.2 instead of just pandas in the list argument passed to the install_requires parameter."

From the online book *Research Software Engineering with Python* Section 14.3.

Managing Dependencies

- Why is specifying versions important for reproducibility?
- Why is it usually better to specify a range of versions rather than one specific version?

Managing Dependencies: Version Ranges

"You don’t have to specify an exact version, you can also use comparisons like <=, !=, and >= to give ranges of package versions that are compatible with your code (see here)."

Managing Dependencies: Versions

Not specifying a version:

Dependencies are likely to introduce breaking changes in the future that will cause your code to fail or give different results.

Pinning an exact version:

Specific versions may not be available on all platforms. You (or new users of your code) won't get bug and security fixes in new versions.

Conda

conda is a virtual environment, dependency, and package manager for multiple languages. There are multiple distributions including Anaconda, which comes with many common data science libraries pre-installed, and Miniconda, which is conda without pre-installed dependencies.

Containerization: Docker

Docker - creates isolated “containers” which are whole virtual systems, allowing you to configure everything including the operating system to use. This is a “maximally reproducible” solution that ensures future users of your code get a complete and identical environment from the ground up.

Other Tools

pyenv - install and manage different versions of Python

Poetry - create virtual environments and Python packages, and improved dependency management

setuptools - for creating Python packages (we'll be looking at this later)

The table at the end of The Alan Turing Institute's *Research Software Engineering with Python*, Section 6.2 gives a easy-to-read comparison.

Packaging

For this section, we will follow The Alan Turing Institute's *Research Software Engineering with Python*, Section 6.4

Packaging

Project layout is important. This was covered in "Project Planning," earlier in these slides.

Packaging for Python

"The three most common package config files are:

pyproject.toml (preferred)

setup.py (may be needed for packages with complex build requirements)"

Packaging for Python

"There are multiple 'build systems' that can interpret pyproject.toml files and build your package.'

Some options: setuptools, Poetry, Flit and Hatch.

"The structure of pyproject.toml will differ depending on the tool you're using."

Packaging for Python: pyproject.toml

You specify:

- Build system - [build-system]
- Project metadata - [project]
- Dependencies
- Python version
- Optional dependencies - [project.optional-dependencies]

Packaging for Python: pyproject.toml

```
[build-system]
requires = ["setuptools"]
build-backend = "setuptools.build_meta"

[project]
name = "greetings"
version = "0.0.1"
requires-python = ">=3.6"
dependencies = ["colorama ~= 0.4.4"]

[project.optional-dependencies]
dev = ["pytest ~= 7.1.2"]

[project.scripts]
greet = "greetings.command:process"
```

Packaging

- Write a script to do the packaging
- You can create the package and install it locally or share it with others

Reproducibility: Other Resources

https://carpentries-incubator.github.io/python-intermediate-development/42-software-reuse.html

Keeping Track of Changes

Keeping Track of Changes

In this section, we will follow the article "Good enough practices in scientific computing", Keeping track of changes section.

Keeping Track of Changes

"Keeping track of changes that you or your collaborators make to data and software is a critical part of research. Being able to reference or retrieve a specific version of the entire project aids in reproducibility for you leading up to publication, when responding to reviewer comments, and when providing supporting information for reviewers, editors, and readers."

Keeping Track of Changes

Version Control systems, such as Git, Mercurial, and Subversion, are the best way to track changes. However they have a learning curve. The authors suggest two possible ways to track changes:

(1) a "systematic manual approach", and

(2) a version control system

Keeping Track of Changes

Guidelines:

- Back up (almost) everything created by a human being as soon as it is created

- Keep changes small

- Share changes frequently

Keeping Track of Changes

Version control systems work best for text data because you can see a diff to pinpoint exactly what change and because file sizes are not too large.

Data files are usually not good candidates to be placed in version control because of file sizes and because the files are not likely to change.

Binary files can be placed in version control if they are an important part of the project, but diffs will not be meaningful.

Keeping Track of Changes: Manuscripts

The authors offer two different recommendations:

- *Write manuscripts using online tools with rich formatting, change tracking, and reference management..., such as Google Docs. With the document online, everyone's changes are in one place and hence don't need to be merged manually.*
- *Write the manuscript in a plain text format that permits version control..., such as LaTeX or Markdown, and then convert them to other formats, such as PDF, as needed using scriptable tools like Pandoc.*

With the second option, the text files can be placed in version control. This is useful for "concurrent changes. It also provides a convenient platform for making comments and performing review."

Have a Definitive Copy of Everything

If you have a definitive copy of:
- Data
- Configuration files
- Code for all stages of processing
- Setup files for virtual environments
- Setup files for dependency management
- Setup files for containerization or packaging, if applicable
- Build/run files
- Documentation with instructions for the entire workflow

...the processing should be fully reproducible. Version control is ideal.

Other Resources

The Alan Turing Institute's *Research Software Engineering with Python*, Chapter 4

Wrapping Up

Feedback

Comments or suggestions related to this workshop?

Topics you'd like to have covered in the future?

Would you like to be notified about future workshops, office hours, etc.?

THANK YOU

UNIVERSITY OF MARYLAND

FEARLESSLY FORWARD