Configuring programs

Data Sciences Institute
University of Toronto

Simeon Wong

Course objective

How to write robust software in a team that we, our colleagues, and the public can trust and use with confidence.

Course overview

1. ☑ Version control with Git
2. **Command-line Python & Configuration**
3. Raising errors
4. Testing software
5. Application programming interfaces
6. Modules and packages

Today’s learning outcomes

- I can write Python scripts
- I can interpret and write simple YAML files
- I can load YAML files into Python
- I can read command line arguments from Python

Command-line Python

Notebooks vs. Command line

- So far, we've been using notebooks
  - Great for exploratory analyses, generating basic reports
  - Work that benefits from in-line figures and feedback
- Python can be used from the command line
  - Code that is well-established and stable
  - Great for automation (batch-processing)
  - Deployment on servers without graphical interfaces

Command-line Python

$> Interactive live coding

1. Create a Python script named `hello_world.py`

```
print('Hello world!')
```

Python CLI Considerations

- Errors will cause Python to quit immediately
  - Variables will be lost
  - Won't allow execution of sections of code multiple times that result in unexpected results
- Data and visualizations should be saved to disk
  - `pd.save_csv(`*`filename`*`)`
  - `plt.save_fig(`*`filename`*`)`

Command-line Python

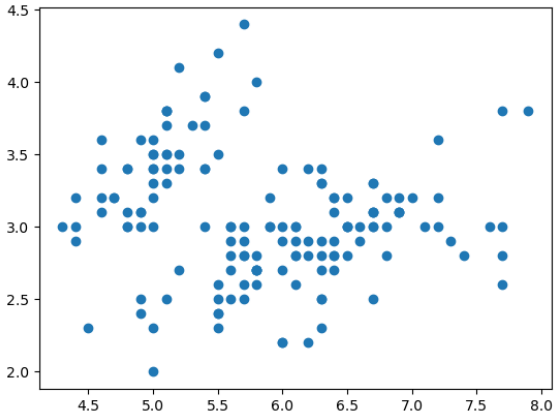
$> Interactive live coding

1. Create a Python notebook with a cell that plots the UCI Iris dataset

```
import matplotlib.pyplot as plt
import pandas as pd

iris_data = pd.read_csv(
    'https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv')

# Create a scatter plot
plt.scatter(iris_data['sepal_length'], iris_data['sepal_width'])
```



Command-line Python

$> Interactive live coding

1. Copy the code into `plot_data.py`
2. Run in the terminal

   ```
   python plot_data.py
   ```

3. What happens?

Command-line Python

$> Interactive live coding

1. Modify `plot_data.py` to save the figure

```
plt.savefig('iris_scatter.png')
```

VSCode Interactive View for debugging

- Allows you to test out your code by running specific lines from your Python script like the notebook interface
- Alternatively, test your code in a notebook first, then convert to a script

Command-line Python

$> Interactive live coding

1. Run lines from `plot_data.py` in VSCode Interactive View

Configuring programs

Why configure?

- Programs are only useful if they can be **controlled**.
- Work inside programs is **reproducible** only if controls are **explicit** (documented and understandable) and **shareable**.
- Programs may need to work differently in different contexts:
  - Operating systems
  - Data directories
  - Localization: language, region settings
  - Analytical methods or outputs (e.g. linear vs quadratic regression)

Data sources

- Data that controls how a program functions can come from multiple sources:
  - Context
    - Current operating system
    - System language
  - Files
    - Configuration files or job description files
  - Dynamic sources
    - Network server, Secrets manager, Azure/AWS/GCP configuration server, etc...

Layers of configurations

1. Default program configuration
2. System-wide configuration
   - General settings that impact all users (and multiple programs)
3. User-specific configuration
   - Personal preferences
4. Job-specific configuration
   - Run-specific information
5. Command-line options
   - Frequently changed, or on-the-fly configuration

Default program configuration

- Sensible defaults included with the program
  - Designed to be generally useful for most people who run the program
  - e.g. matplotlib default colour is blue
- Consider different types of users
- Be careful of unexpected behaviour if left unconfigured
  - e.g. Program that saves data to `financials.xlsx` by default
    - Might unintentionally overwrite users' files
  - Some defaults should not be set to force users to specify

System-wide configuration file

Purpose: To provide default settings applicable to all users of a system.

Impact: Changes affect every user and application on the system.

Modification and Risks: Editing system-wide files requires caution as incorrect settings can impact system stability.

e.g: Modifying a system-wide configuration file in an operating system to change the default network timeout settings. This change will affect all network-related operations for all users, even those using other programs.

User-Specific Configuration

Scope: Settings that apply only to a single user's environment.

Flexibility: Allows personalization without affecting other users.

Storage: Typically stored in user's home directory or specified user profile sections.

e.g: A user creates a configuration file in their document editor to set a default font size and page layout, different from the system-wide defaults.

Job-Specific Configuration

Application: Used for settings that apply to a specific task or project.

Priority: Overrides system and user-specific settings for the job's duration.

Creation and Use: Crafted for individual projects or tasks, often located within the project directory.

e.g: Setting up a configuration file in a data analysis project to specify data sources and output formats unique to that project.

Command-line options

Use: For temporary adjustments or one-off changes.

Flexibility: Allows quick, on-the-fly changes without altering permanent configurations.

Best Practices: Use for frequent or minor changes; avoid for complex configurations.

e.g: Running a file compression tool with command-line options to set a high compression level for a specific large file, overriding the default compression setting.

Layers of configurations

1. Default program configuration
2. System-wide configuration
   - General settings that impact all users (and multiple programs)
3. User-specific configuration
   - Personal preferences
4. Job-specific configuration
   - Run-specific information
5. Command-line options
   - Frequently changed, or on-the-fly configuration

Overlay/override system

- The same option is valid in multiple configuration layers
- Options read from more specific layers override settings configured in more general ones

e.g: Plot colours for an analysis program

- Program's default colour is blue (matplotlib default)
- Your organization's brand colour is green (system-wide)
- For this specific analysis, need to use orange (job-specific)

Questions?

Command-line arguments in Python

- The **argparse** library built into Python is used to interpret command-line arguments
- Common types of command-line arguments
  - Positional argument (e.g. `mv oldpath newpath`)
  - Flags (true/false values) (e.g. `ls -l`)
  - Options with values (e.g. `head -n 5`)

$> Interactive live coding

1. Create a Python script
2. Make it print “Hello World!”
3. Referring to the **argparse** documentation:
    1. Initialize **argparse**
    2. Write a short description
    3. Add a **positional argument** for greeting target
    4. Add an **option** for number of repeats
    5. Add a **flag** for saying goodbye

Configuring programs

$> Interactive live coding

```
import argparse

parser = argparse.ArgumentParser(description='Say hello (or goodbye!)')
parser.add_argument('name', help='Name of person to greet')
parser.add_argument('--goodbye', action='store_true', help='Say goodbye instead of hello')
parser.add_argument('--repeat', '-r', type=int, default=1, help='Number of times to greet')
args = parser.parse_args()

if args.goodbye:
    message = 'Goodbye'
else:
    message = 'Hello'

for _ in range(args.repeat):
    print(f'{message} {args.name}!')
```

Questions?

Configuration file formats

Popular Formats:

- INI – older format for simple, structured data
  - Commonly found in compiled Windows / Linux programs
- JSON – older standard, flexible, can be difficult to type by hand
- YAML – new standard, readable and flexible

Due to its flexibility and readability, especially for complex configurations, YAML is recommended.

Configuring programs

The YAML format

```
key: value

dict:
  key1: value1
  key2: value2

list:
  - value1
  - value2

# comment
```

“YAML is Another Markup Language”

Online tools for browsing YAML:
https://codebeautify.org/yaml-parser-online

$> Interactive live coding

1. Create a YAML file
2. Write Python code to load and consolidate YAML files
3. **Exercise:** Write your own configuration files
   - For an analysis where a dataset is loaded and two columns from the dataset are plotted on the x and y axes in your favourite colour
   - Variables: dataset_url (str), cols_to_plot (dict with x and y str), color (str)
   - Which configuration levels do these variables go into?

Configuring programs

$> Interactive live coding

```
import yaml

config_paths = ['user_config.yml']
config_paths += args.config

config = {}
for path in config_paths:
    with open(path, 'r') as f:
        this_config = yaml.safe_load(f)
        config.update(this_config)
```

Environment variables

- A standard mechanism to provide user-specific, system-wide, or contextual configuration to applications
- Environment variables can be:
  - readable by all programs running on a system
  - specific to a terminal session or shell script
  - specific to a particular command

Configuring programs

$> Interactive live coding

1. Read environment variables in the terminal with `env`
2. Read environment variables in Python

```
import os
print(os.environ)
```

Today’s learning outcomes

- I can interpret and write simple YAML files
- I can load YAML files into Python
- I can read command line arguments from Python

Course objective

How to write robust software in a team that we, our colleagues, and the public can trust and use with confidence.

Homework

- Convert your Python Assignment 2 into a script
- Add argument parsing and configuration files

- Ungraded. Submit before Saturday, Feb 10 at noon for feedback.

References

- Research Software Engineering with Python by Damien Irving, Kate Hertweck, Luke Johnston, Joel Ostblom, Charlotte Wickham, and Greg Wilson (https://merely-useful.tech/py-rse/config.html)