Exercises, day 1, morning

Best practices in scientific programming

Software Carpentry, Part I

Valentin Hänel* Rike-Benjamin Schuppner†

Python Winterschool Warsaw, Feb 2010

EXERCISE 1 CREATE YOUR SPACE ON THE COURSE REPOSITORY

Learn how to obtain a copy of an existing repository, and add new files to the repository.

i) Check out a copy of the SVN repository you will be using in this course:

```
svn co --username=your_username ↩
    https://escher.fuw.edu.pl/svn/python-winterschool/public winterschool
```

(all on one line). This command will create a local copy of the repository under the directory winterschool. This is the place where you are going submit your exercises and work on the various projects with the other students. Notice the option `--username`, which needs to be used here, as you'll be sharing the repository with your programming partner for the morning.

ii) Create a personal directory:

```
cd winterschool/students
mkdir yourname
```

iii) In the new directory, create a text file called `README`, and write your name and your email address into the file.

iv) Add the new directory and the file to the repository, and then commit them. Remember to provide a meaningful comment when committing the file!

EXERCISE 2 COLLABORATING WITH SVN

Write a story using the SVN techniques. Learn to follow the basic SVN cycle; use SVN to collaboratively work on a set of files; resolve conflicts that arise from simultaneous changes.

i) Enter the SVN repository at `winterschool/day1/morning/exercises/stories`. In a text file called `metasyntactic_variable.txt`[1], write a short 2-sentences story. For example,

> Tiziano bought a brand new laptop.
> He immediately installed Python.

*valentin.haenel@bccn-berlin.de
†rike.schuppner@bccn-berlin.de

[1] Please choose whatever comes to your mind.

Put each sentence on a new line.

ii) Add and commit the new file to the repository. Wait a moment for everybody else to finish, then update the repository to get their stories.

iii) Pick a file at random, and expand one of the sentences in two new sub-sentences that give more details about the story. For example, the new text in `has_no_title.txt` could read

> Tiziano spilled coffee on his laptop.
> He had to replace it with a new one.
> He immediately installed Python.

Remember to use one line per sentence.

iv) Update the repository and solve the possible conflicts (choose one of the two conflicting versions, or manually merge the two versions into a new one), then commit the file.

v) Pick another story and start again from iii). Continue until somebody stops you. Make sure you stick to the SVN basic work cycle.

vi) Have a look at the stories, and use the commands `svn blame` and `svn log` to find out who is responsible for editing them.

EXERCISE 3 BUG HUNT

Isolate a bug in existing code; debug using agile development practices.

i) Enter the course repository at `winterschool/day1/morning/exercises/debug`. Copy the files in that directory into your personal directory (create a new sub-directory for this exercise) and commit them.

ii) Have a look at the files `working.data` and `not_working.data`:

```
cat working.data
cat not_working.data
```

The program `convert_to_dict.py` reads these files line by line and converts each line into a dictionary of numbers (look at the docstrings in the code for details). Run the program on the two data files

```
python convert_to_dict.py working.data
python convert_to_dict.py not_working.data
```

and observe how it fails to convert the data in the second case.

iii) Use `pdb` to inspect the code and isolate the bug. *Hint: Use breakpoints to jump out of uninteresting loops; look at the local variables with* `locals()`.

iv) In the file `tests.py` there is a simple unit test with some code left out. Change the code according to the comments.

v) Add more test cases to `tests.py`, which reproduce the error. Run the test and verify that it fails.

vi) Correct the bug and verify that the new code passes the tests.

vii) Commit everything to the repository. Remember to set a meaningful message.

EXERCISE 4 SUDOKU SOLVER (BONUS EXERCISE)

Practice test-driven development; use Python tools to optimize, debug, and document the code.

i) Enter the course repository at `winterschool/day1/morning/exercises/sudoku`. Copy the files into your personal directory and commit them.

ii) Look at the test cases in `test_sudoku.py`. Write a module `sudoku.py` that makes the tests pass (this is equivalent to writing a Sudoku solution verifier and a Sudoku solver). Some notes:

- The file `problems.py` contains two dictionaries with Sudoku boards and their solutions. Each board is represented as a 2D list. Write three helper functions, `get_row(grid, nr)`, `get_column(grid, nr)`, and `get_box(grid, nr)`, that return the `nr`-th row, column or box of the Sudoku grid. These will come very handy. Make sure you write tests for the new functions!
- Start by working on the Sudoku verifier, `sudoku.is_solution`.
- Use a brute-force approach to solve the Sudoku board in `sudoku.solve_sudoku`:

> Briefly, a brute force program would solve a puzzle by placing the digit '1' in the first cell and checking if it is allowed to be there. If there are no violations (checking row, column, and box constraints) then the algorithm advances to the next cell, and places a '1' in that cell. When checking for violations, it is discovered that the '1' is not allowed, so the value is advanced to a '2'. If a cell is discovered where none of the 9 digits is allowed, then the algorithm leaves that cell blank and moves back to the previous cell. The value in that cell is then incremented by one. The algorithm is repeated until the allowed value in the 81st cell is discovered. The construction of 81 numbers is parsed to form the 9×9 solution matrix. (Excerpt from Wikipedia[2])

iii) Profile your code on the `'hard2'` problem. Save the profile results in `sudoku.profile`. Examine the results and discuss what could be optimized and how (write the response in `optimize.txt`).

iv) Check that your code adheres to Python standards using pylint: `pylint sudoku.py` Improve your code until the overall pylint score is greater than 7.0.

v) Create documentation for your Sudoku solver: `pydoc -w sudoku`

5		1	2	8				
8						7		2
2						1	8	5
	1	4	7			5		
			4				2	
	2	6						
1				3	6			
4							5	1
6				4	1			

[2] `http://en.wikipedia.org/wiki/Algorithmics_of_sudoku`