Practical 11
Projects in Python

Learning Objectives

1. Access and assess packages in the Python Package Index
2. Evaluate risks in using packages and code from other developers
3. Know some useful tools to support your software development

Overview

In this practical you will use tools that can help when developing code

Tasks

1. Finding packages in PyPI

In your browser, go to the Python Package Index at https://pypi.org/. In the search box, enter “funniest” to find the example package shown in the lecture. Note that there are now many variations of this package as people all over the world work through the example and publish their version of the package.

Go back to the PyPI home page and select “Browse Projects”. On the left, open the “By Topic” filter and choose a topic/subtopic. For example, “Astronomy” will give you 451 projects – packages developed that can be used for astronomy applications.

Remove the astronomy filter and add a new one for “text processing, markup, XML”. These packages can help translate XML code into python data structures. Have a look at “xml2data 0.1.0”. Looking at the project, what can you see that would indicate it may not be a good package to use in terms of suitability and risk? Another option is beautifulsoup4. How does it compare? Which would you choose and why?

2. Getting to know Git

Git is a program for keeping track of files and versions, helping you to manage projects over time and among project teams. Create the Prac11 directory and change into it. To check the current contents of the directory, type “`ls -la`”. You should only see . and .., representing the current and the parent directories.

To initialise a new git repository, type “`git init`”. You should see a message to say the empty repository has been initialised.

Type "`ls -la`" again to see what has changed. You should see a ".git" directory. Typing ls will show an empty directory as these files are "hidden" - they start with ".".

Type "`ls -Ra .git`" to see what the git initialisation created. You don't need to know what goes on in these directories – that's git's business, but I think it helps to know and respect the magic behind the scenes.

To set up git, we should let it know our name and email address. Modify the following with your own details to set up git:

```
$ git config --global user.name "Vlad Dracula"
$ git config --global user.email "vlad@tran.sylvan.ia"
```

We now need some code to work with. Download accounts.py and testAccounts.py into your Practical11 directory.

Type "`git status`" to see the current state of the repository. There should be no files waiting to be committed.

Type "`git add accounts.py`" and then "`git status`" - and you should have new file accounts.py tracked and waiting to be committed.

Add testAccounts.py to track it in the staging area. You can continue to update tracked files in the staging area. Once they are at a point that you want to put them into the repository, we can commit them

To commit the files, type:

```
git commit -m "Initial versions of files"
```

The -m indicates a commit message and you need to put a meaningful message in with each commit.

Check `git status` to see what has changed.

3. PEP8 tool

PEP8 is a guideline for the style of our code. There is a pep8 utility that will check code against the guidelines. Run **pep8** against accounts.py and testAccounts.py. Look at the feedback and make changes until it is accepted by pep8. Once you have completed the updates, you can stage and commit the new versions of the programs. Git can tell you if there have been untracked changes:

```
git status
```

… will indicate that your Python files have been modified but not staged.

```
git add accounts.py testAccounts.py
git commit -m "PEP8 compliant versions of files"
```

4. Pyflakes tool

Pyflakes checks our code for correctness and gives more useful feedback than just using python3. Run pyflakes against accounts.py, testAccounts.py and dodgy.py. Look at the feedback and make changes until it is accepted. Once you have completed the updates, you can stage and commit the new versions of the programs:

```
git add accounts.py testAccounts.py dodgy.py
git commit -m "pyflakes compliant versions of files"
```

5. Pylint tool

Pylint is another code checker – looking for bugs and assessing the quality of your code. Run pylint against accounts.py, testAccounts.py and dodgy.py. Look at the feedback and make changes to see how your assessment improves. Note: you may not be able to get a perfect result. Once you have completed the updates, you can stage and commit the new versions of the programs:

```
git add accounts.py testAccounts.py dodgy.py
git commit -m "pylint compliant versions of files"
```

6. Spyder environment

Spyder is an integrated development environment for developing code. If you were doing a lot of coding, this would be the recommended environment to work in. Download the dodgy.py code again but rename it as dodgySpyder.py. Open the file in spyder. You can run the application, or run it from the command line as spyder& (the & makes it run in the background so you get your command prompt back).

Spyder also does code checking. Press F8 to check the code and run through all your changes to fix the dodgy program. Spyder produces an “output” file including the feedback – click on “output” on the right to see the feedback. Once the syntax errors are fixed, the code checker gives more useful feedback in the static code analysis tab on the top right. You’ll also notice the code is decorated with symbols marking any issues. Hopefully you’ll recognise that Spyder is giving the output of all of the previous tools.

Make the required changes to fix your code. When the code does run, you will see that the variables are in the window on the right hand side (top) and the output is on the bottom right. Once you have completed the updates, you can stage and commit the new versions of the programs:

```
git add dodgySpyder.py
git commit -m "Spyder update of files"
```

7. Git outta here!

In each task, we've committed new versions of our files. To see the history of your work in Practical 11, type "git log". In there, you can see the commit identifier, which lets you get back to any version of your files. There's not enough time to go through all of git, but hopefully you've seen how it fits into a workflow.

For more about git, see https://swcarpentry.github.io/git-novice/ for an excellent tutorial.

And that's it for the Fundamentals of Programming practicals!

Submission

Create a README file for Prac11. Include the names and descriptions of all of your Python programs from today. All of your work for this week's practical should be submitted via Blackboard using the link in the assessments area. This should be done as a single "zipped" file.

Reflection

1. **Knowledge**:
2. **Comprehension**:
3. **Application**:
4. **Analysis**:
5. **Synthesis**:
6. **Evaluation**:

Challenge

- To come