INF200_H19_J07

June 12, 2020

1 INF200 Lecture No J07

1.0.1 Hans Ekkehard Plesser / NMBU

1.0.2 11 June 2020

1.1 Today's topics

- Status
- Priorities for remaining work
- Exam information
- Schedule for remainder of block
- Distribution packages
- Profiling and optimization

1.2 Status

- By today, all groups should have
  - herbivores and carnivores behaving correctly in a single lowland cell as in my examples
- By tomorrow afternoon, all groups should have
  - a draft version of migration

1.3 Priorities for remaining work

1. Migration
2. Visualization (use RandVis as a starting point!)
3. Make sure your code passes `test_biosim_interface.py` test and that `check_sim.py` works with your code
4. Documentation with Sphinx
5. Packaging
6. Optimization

1.4 Exam information

1.4.1 Time and Place

- **Ca 09.00-17.00, Monday 22 June and Tuesday 23 June 2020**
- You can find your specific exam time on Canvas and Teams (coming soon)
- Room: Zoom room for INF200, details will follow

1.4.2 Format

- Both students in a group are examined together
- Short presentation of your results (**five minutes**)
  - You may use one PDF file (a few slides) and one short animation (MP4 or GIF)
  - These must be submitted by Saturday, 20 June, 12.00
- Discussion with examiners (20 minutes, Plesser and Jochen Eppler, Research Centre Jülich)
  - Your code will available on screen during discussion

1.4.3 Content of presentation/discussion

- You are a software developer presenting your product to your client
- Explain to the client/examiner how you have solved the task
  - Overall structure of the code
  - Examples of how you solved specific aspects/problems
- Persuade the client/examiner that your code is trustworthy
  - How did you ensure quality?
  - Is code in maintainable shape (tests/documentation etc)
  - Do you as a developer know your stuff?
- Pursuade the client that your code is productive
  - Ease of use. Is it easy to use and manipulate the code you have written?
  - Performance: Is the code fast?
- Show some interesting results
- Remember, 5 minutes is not very long!
  - Choose in advance what you want to spend your time on
  - Do not make too many or too verbose slides

1.4.4 Evaluation

- Code evaluation jointly for both team members:
  - 0-70 points
- Individual evaluation of performance during exams:
  - 0-30 points
- Final grade set according to total number of points
  - A-F

1.4.5 Criteria for code

- Completeness
  - Did you provide everything customer asked for?
  - Did you do any further development?
- Code Quality
  - Correctness of implementation
  - Were specifications followed?
  - Suitable coding constructs vs unnecessarily complex code
  - Readability of code
  - Pythonic coding style
  - Tidiness (PEP8, maximum line length 100)
- Quality control/Testing

- Documentation (docstrings and sphinx) (PEP257)

1.4.6 Criteria for presentation

- Your ability to present and explain your code
  - Overall structure
  - What solutions you chose and why
  - Why your code is trustworthy
- Your ability to discuss your code
  - Do you understand your own code?
  - Are you aware of its limitations?
  - What would you improve/extend if you had better time?

1.4.7 Submission of material

Project Code

- **Deadline: Thursday, 18 June, 12.00**
- Submit your code as follows, working in your team repository (video will follow):
  1. Commit your final changes
  2. Make sure all your code is in your **`master` branch**
  3. In your Git program, add the tag `BiosimSubmission` as tag (no spaces)
  4. Push tag to Github
  5. **Confirm that tag is visible on Github!**
- The tagged commit must be dated no later than 18 June 2020, 12.00 CEST.
- Test creating tags before next Thursday!

Presentation files

- All presentation files must be handed in, and will be available on the examiners machine during the presentation, this is to save time on setup between students.
- **Deadline: Saturday, 20 June, 12.00**
- Submit your material (one PDF, one mp4 or GIF) as follows, working in your **team repository**:
  1. At the top level in the repository, create a folder `Exam`
  2. Put your files into the `Exam` folder, on branch `master`
  3. Commit and Push
  4. Add a tag `INF200Exam` to the final commit of the material
  5. Make sure material and tag are visible on Github
- Files and tag must be in place by the deadline.
- I will contact you by mail if there is any issues getting hold of your pdf/mp4/gif so **please check your mail during Saturday and Sunday!**

1.5 Schedule for remainder of block

- Lecture Thursday 18 June, 13.30-15.00 on C++
- For the remainder of the course, we will have morning meetings (from 09.00) and afternoon meetings (from 14.30) on Zoom to discuss status and provide advice, but no more lectures
  - Mandatory attendance will be checked
- Friday 19 June

  - Individual work on your presentations
  - *No* mandatory attendance
  - Amir and Bishnu will be available for some time for questions
    * Precise times will be posted later
- Saturday 20 June
  - Presentation deadline 12.00 noon CET
- Monday/Tuesday 22/23 June
  - Exam

1.6 Distribution Packages

Let us say you have spent the last year creating some really great Python code, and now you want to share it with others. What do we need to do? - Need to put "everything together" into a nice "parcel" - Need to handle *dependencies* (e.g., that our code needs NumPy) - Need to "spread the word (code)"

Python solution: *Packaging*

Packages *vs* Packages You might have noted that we now have to different things called *packages*, they are either - Collections of modules (import packages) - A collection of code neatly packaged for sharing with others (distribution packages)

Yes, having the same name for two different things is confusing. Programmers are horrible at naming conventions, we just have to deal with that

The Python Packaging User Guide Glossary defines a Distribution Package as

"A versioned archive file that contains Python packages, modules, and other resource files that are used to distribute a Release. The archive file is what an end-user will download from the internet and install."

1.6.1 Where to share distribution packages?

You have now created a nice distribution package of your code (we will check out the details soon), how do you share it? - If it is only with a few people, email, direct transfer, etc is fine - If you want to keep the code open for everyone to see, github/bitbucket is a nice way to do it - Alternatively, you can use the Python Package Index (PyPI), aka the "CheeseShop" - If you want to make it easily available for Conda users, consider creating a Conda package as well - Discussion of Conda vs PIP by Jake Vanderplas

Quiz: Why is the Python programming language called Python?

Guido van Rossum, Pythons original creator and Benevolent Dictator For Life (retired summer 2018), is a huge fan of Monty Python's Flying Circus.

Due to this, you will often find references to Monty Python if you look through Python's surrounding community and documentation. For example with PyPI, which is referred to as the Python Cheese Shop

1.7 How do we create Distribution Packages?

We wil only skim over the subject briefly, for details see for example - Python Packaging User Guide - PyPA Sample Project on Github - https://docs.python.org/3.8/distributing - https://docs.python.org/3.8/installing

Key idea of a distribution package We want to make sharing Python-based projects easy - Collect - Source code: Python modules, import packages, tests - Example scripts - Documentation - ...

- Provide *metadata* about the code, e.g.,
  - Purpose, Dependencies, Author information
  - License information, Version information, ...
- Provide a *build archive*
- Support easy installation to predefined locations

Example: Typical distribution package directory layout

```
chutes_project
   chutes
      __init__.py
      board.py
      player.py
   examples
      chutes_demo.py
   tests
      test_board.py
   MANIFEST.in
   README.rst
   requirements.txt
   setup.py
```

In our example, `chutes` is an import package included in our distribution, it is the source code. In this example `tests` is placed next to the source code package.

In addition to the `chutes` package we have a folder called `examples`, with some scripts the user can look at to see how the `chutes` packaged can be used. Note that `example` is *not* a package, as it does not have an `__init__.py` file, it is just a regular folder. If you have a notebook with examples, they could also be placed here.

Other files - A `README.rst` contains a description of the distribution package, and usually contain some information to the user about how to install it and where to look for examples/documentation. The file type is flexible, but it should be a reStructured Text file for compatibility with PyPI - `MANIFEST.in` describes which files to include in addition to packages (e.g., example scripts) - `setup.py` is a Python script using distutils or setuptools to define the distribution package including metadata - `requirements.txt` lists Python packages that are required for this software to work (see Pip documentation)

When somebody downloads your distribution package, they will install your package by running the `setup.py` script. This can be done in several different ways.

First, make sure that you are in a Terminal/Anaconda Prompt window in the directory containing the `setup.py` file for the package you want to install.

Installing in an existing conda environment

1. Make sure you have activated the right conda environment.
2. Run `python setup.py install`
3. Once complete, you should see a line like

```
Installed <PATH TO HOME>/miniconda3/envs/<ENV NAME>/lib/python3.8/site-packages/<PACKAGE NAME E
```

where the parts in `<>` depend on your details.

The package is installed in the environment and only available if the environment is active.

To avoid cluttering your base Python installation, this approach is recommended if you use conda environments (which you should).

Installing for you as user

1. Run `python setup.py install --user`
2. Once complete, you should see a line like

```
Installed <PATH TO HOME>/.local/lib/python3.8/site-packages/<PACKAGE NAME ETC>.egg
```

The package is then available every time you run Python 3.8 (or whichever Python version you used to run `setup.py`.

Uninstalling There is no automated way to uninstall packages installed as described here. You need to go to the corresponding `site-packages` directory and remove the directory for the package you want to uninstall.

Therefore, it is smart to use conda environments: You avoid clutter and can just delete an environment when you don't need it anymore.

1.7.1 A typical setup.py for a simple project

```
from setuptools import setup
import codecs
import os

def read_readme():
    here = os.path.abspath(os.path.dirname(__file__))
    with codecs.open(os.path.join(here, 'README.rst'), encoding='utf-8') as f:
        long_description = f.read()
    return long_description

setup(
      # Basic information
      name='chutes',
      version='0.1.0',
```

```
      # Packages to include
      packages=['chutes'],

      # Required packages not included in Python standard library
      requires=['pytest'],

      # Metadata
      description='A Chutes & Ladders Simulation',
      long_description=read_readme(),
      author='Hans Ekkehard Plesser, NMBU',
      author_email='hans.ekkehard.plesser@nmbu.no',
      url='https://github.com/heplesser/nmbu_inf200_june2020',
      keywords='simulation game',
      license='MIT License',
      classifiers=[
        'Development Status :: 3 - Alpha',
        'Intended Audience :: Developers',
        'Topic :: Science :: Stochastic processes',
        'License :: OSI Approved :: MIT License',
        'Programming Language :: Python :: 3.8',
        ]
)
```

A typical `MANIFEST.in` file

```
recursive-include examples *.py *.ipynb
recursive-include tests *.py
```

This tells `setup.py` to include recursively all `*.py` and `*.ipynb` files from `examples` and all `*.py` files from `tests`.

Creating packages In addition to being used for installation, the `setup.py` script can also be used to create a nice *archive* of your package for easy transmission as a single file - Run - `python setup.py sdist`

- To create a zip-archive (easier on Windows), do - python setup.py sdist –formats=gztar,zip

The archives are placed in directory `dist`

You can then share the compressed archive file (zip, tgz, …)

Automatic installation of packages Many popular packages are available through Anaconda, you can simply write - `conda install <package name>` In your terminal (requires you to have Anaconda installed). This automatically downloads and installs the package, if it is available

Not all packages are avilable through conda, an alternative is `pip`, which is a program to automate - `pip install <package name>`

1.8 Optimization

1. Profile to understand where you program uses time

- Disable all graphics and file output for profiling
- Profile in PyCharm using `Profile` from the `Run` menu
  - See call statistics and call graph
  - Graph can also be stored as figure

2. Optimize the parts that require most time first
   - See if you can find "low hanging fruits": individual functions taking a lot of time
3. How to optimize
   - Reduce number of function calls required
   - Make functions run faster
4. Techniques
   - Lazy evaluation
     - If a computation is costly, perform it only when really necessary
     - Mark value as invalid if changes occur that will require recomputation at some point
     - When a value is marked invalid when requested, recompute and store
   - Implementation of lazy evaluation
     - Use *properties* in Python
   - Just in time compilation using Numba
     - See discussion in lectures in the fall term
     - Use only for compact functions or methods performing mathematical operations
   - Coding critical parts with Cython (or even in C++, but that is a whole new story …)

[ ]: