INFO1112 A2 marking guide

This assignment will be marked out of 20. Each component is weighted as follows.

- Auto-test: 75% (15 marks)
- Manual inspection of code: 10% (2 marks)
- Manual inspection of self-testing: 15% (3 marks)
- Breaking restriction (4 marks deducted)

MARKING BUDGET

18 mins per student.

FEEDBACK TEMPLATE

- Tutors may use the `feedback_template.txt` template when writing feedback on Ed.

AUTOMATED MARKING

(tutors can ignore this section)

Marks are evenly distributed for test cases dedicated for each task, including public and private cases.

MANUAL MARKING

Manual inspection of code

A manual mark will be given for overall style, quality, readability, etc (2 marks).

1. Briefly open 1 or 2 of the Python programs submitted by the student, and quickly scroll through and see the most common naming scheme that is used (`snake_case`, `camelCase`, `PascalCase`) in functions and variables. This allows for some flexibility in styling when linting their code (15-30 seconds).
2. Copy the `pyproject.toml` and `check_code_style.sh` file included within this folder into the student's submission folder on Ed.
3. Install Pylint in the student's submission's Ed environment by running:

   ```
   pip install --break-system-packages pylint
   ```

   Make *sure* this is run on Ed, as the `--break-system-packages` option for `pip` can cause issues if you use this on a local machine.
4. Execute the `check_code_style.sh` script, and provide a single argument for the student's coding style determined in step 1, which should be:
   - `sc` for snake case
   - `cc` for camel case
   - `pc` for Pascal case

   Example of running script:

```
bash check_code_style.sh sc
```

5. Pylint will give a score out of 10. Award marks based on the score `x` divided by 10 (`x/10`). Then, round this to the closest value from 0.00 / 0.25 / 0.50 / 0.75 / 1.00.
6. Pick two programs (from `server.py`, `recursor.py`, `launcher.py`, `verifier.py`) that passes the most test cases on Ed. Take **6-8** mins reading through the code and make inline comments (see the video `howto-make-inline-comment.mp4`). Award
    - 1.00 marks for clear code with just enough comments/docstrings.
    - 0.75 marks for unclear code or insufficient comments/docstrings.
    - 0.50 marks for unclear code AND insufficient comments/docstrings.
    - 0.25 marks for highly incomplete code.
    - 0.00 marks for zero submission.
7. Press "Save" and refresh the webpage.

Manual inspection of self-testing

The instructions given to students in the specs for designing test cases is as follows:

> The examiner will change the working directory to the root of your git repository and execute bash tests/run.sh to run ALL your test cases.
>
> You are expected to write a number of test cases for all implemented programs, however ONLY the testing for the server program will be assessed. You are expected to test as many execution paths as possible for your code.
>
> We will provide you with one sample test case, but it does not test all the functionality described in the assignment. It is important that you thoroughly test your code by writing your own tests.
>
> Unit tests are not required and not assessable.

The general instructions for marking self-testing are as follows:

1. If the test script `tests/run.sh` cannot be found, deduct 3 marks.
2. If any (unexpected) error message is presented or any case fails, deduct 0.5 mark.
    - Run `bash tests/run.sh`.
      Do not deduct marks if Ed complains about `nc` not existing
    - In case the student has the `nc`-not-found issue, refresh the submission page to clear the cache and reset the submission's workspace. Replace `nc` with `ncat` by using the script below:

      ```
      # this may destroy students' cases if `nc ` is a part of their test
      script or test file name
      find . -type f -name "*.sh" -exec sed -i 's/nc /ncat /g' {} +
      ```

      or make use of the bash script below to conduct the replacement:

      ```
      # this works but with a limited scope
      nc () {
          ncat $@
      }
      ```

      and rerun `bash tests/run.sh`.

3. Check the coverage rate from the "All" row of the coverage report.
   Deduct at most 1 mark by the following rules:
   - If the coverage report is never shown, deduct 1 mark.
   - If the coverage rate is under 40%, deduct 1 mark.
   - If the coverage rate is above 80%, no deduction applies.
   - Otherwise, deduct `2 - (coverage_rate / 40%)` marks.
4. Browser the test script and check if functionalities of the server are properly tested.
   This should take 3-4 minutes. Deduct at most 0.5 marks by the following rules:
   - If invalid configuration is never tested, deduct 0.25 marks.
   - If the standard output is never captured (by file redirection, pipe or a program like `tee`) or compared with the expected output, deduct 0.25 marks.
   - If the server's response is never captured (by the client) or compared with the expected response, deduct 0.25 marks.
5. Press "Save" and refresh the webpage.

Restrictions

Make sure they only use the following libraries, otherwise deduct 20% of A2 (4 marks). If they import an invalid library without actually using it, do not deduct marks. Ask on Discord if you find any "reasonable" breaches that may be eligible for exemption.

```
pathlib
random
signal
sys.argv
sys.exit
socket
time
typing
```

Upload the `check_restrictions.sh` script on Ed to the student's last submission, and execute it.

If the script reports a restricted module or import is used, do a quick manual check to see if the module or import is used within the code. For example, if the student has the line `from invalid_lib import x, y, z`, manually check (e.g. with Ctrl+F) if `x`, `y`, or `z` is being used within their code.

The End

The examiner should fill in the total manual marks (`-4<=x<=5`) in the grade box on Ed. Note that if any deduction applies, the grade may be negative.

Remember to fill in the feedback text box and make inline comments.

Press "Save", refresh the webpage and review.