GEP Coding challenge week 2 extended

Scoring

As stated in the presentation of Phil, this is the scoring table:

Criterium	points
Does it work	1
Readability (names, structure, linting)	3
Handle change / single responsibility	2
Testing	3
Deep Source	1
total	10

Some Recommendations

- Add a README.md with your final answer, and possibly additional comments
- Use comments wisely
- Give variables and functions meaningful names
- Give your methods (functions) just one responsibility
- Create tests. Check at least the example given by Euler.
- Carefully read the presentation of Phil Rice.

A word on linting

Linting is the process of running a program that will analyze code for potential errors.

Linting is important to reduce errors and improve the overall quality of your code. Using lint tools can help you accelerate development and reduce costs by finding errors earlier.

If you use an IDE like PyCharm you get linting for "free".

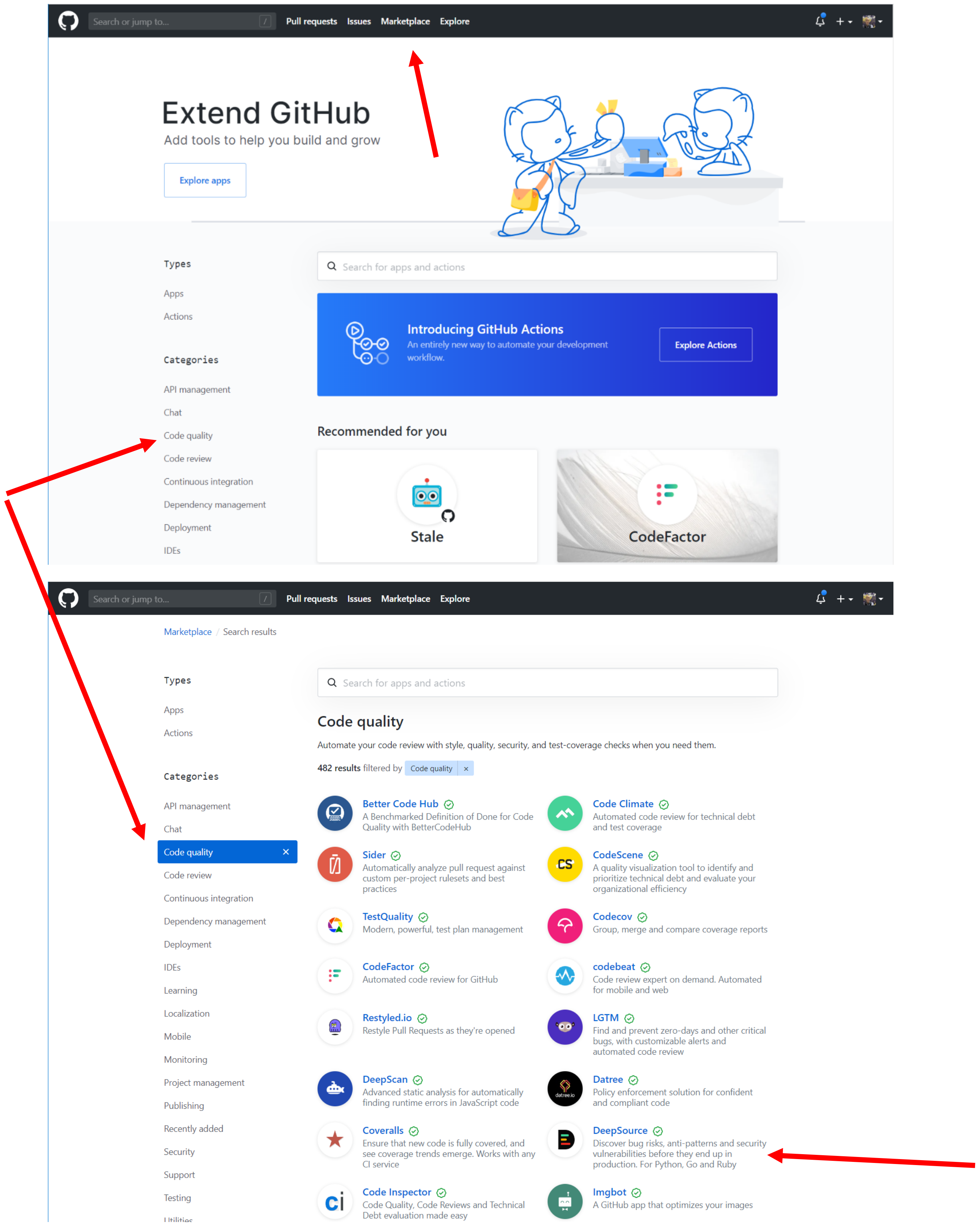
But you can also use a tool like Flake8 from the command-line.

https://pypi.org/project/flake8/ or https://flake8.pycqa.org/en/latest/

A word on Deepsource

DeepSource builds source code analyzers -- to find and fix issues like bug risks, anti-patterns, performance optimizations and security vulnerabilities. Once integrated, DeepSource runs on every commit and pull/merge request.

You can connect Deepsource directly to your GitHub repository.



An entirely new way to automate your development workflow.

Code quality

Automate your code review with style, quality, security, and test-coverage checks when you need them.

482 results filtered by Code quality ×

Better Code Hub
A Benchmarked Definition of Done for Code Quality with BetterCodeHub

Code Climate
Automated code review for technical debt and test coverage

Sider
Automatically analyze pull request against custom per-project rulesets and best practices

CodeScene
A quality visualization tool to identify and prioritize technical debt and evaluate your organizational efficiency

TestQuality
Modern, powerful, test plan management

Codecov
Group, merge and compare coverage reports

CodeFactor
Automated code review for GitHub

codebeat
Code review expert on demand. Automated for mobile and web

Restyled.io
Restyle Pull Requests as they're opened

LGTM
Find and prevent zero-days and other critical bugs, with customizable alerts and automated code review

DeepScan
Advanced static analysis for automatically finding runtime errors in JavaScript code

Datree
Policy enforcement solution for confident and compliant code

Coveralls
Ensure that new code is fully covered, and see coverage trends emerge. Works with any CI service

DeepSource
Discover bug risks, anti-patterns and security vulnerabilities before they end up in production. For Python, Go and Ruby

Code Inspector
Code Quality, Code Reviews and Technical Debt evaluation made easy

Imgbot
A GitHub app that optimizes your images

Deepsource findings from week 1

Legend:

White:	recommendation, but no penalty
Yellow:	-0.1 points penalty per finding
Orange:	-0.5 points penalty per finding
Red:	No points for DeepSource!

anti-patterns	**explanation**
Module imported but unused	
Unused variable found	
Function contains unused argument	
File opened without the `with` statement	`exec(open(filename).read())`
Re-defined variable from outer scope	6 times an iterator like x was used multiple times. 1 time a variable from a function was re-used outside the function
Consider using enumerate for iteration	A lot of times when dealing with iterators, we also get a need to keep a count of iterations. Python eases the programmers' task by providing a built-in function enumerate() for this task. Enumerate() method adds a counter to an iterable and returns it in a form of enumerate object. This enumerate object can then be used directly in for loops or be converted into a list of tuples using list() method.

Re-definition found for builtin function	sum = 0
	list = []
	sum and list are standard Python functions

Unnecessary `else` / `elif`	if len(character_list) == len(number_as_string):
	    return True
	  else:
	    return False

red = 0 (zero), orange = 0.5 each and yellow = 0.1 each	
Bug risk	
Assert statement used outside of tests	Unit test not recognized as such by Python
invalid syntax	SyntaxError: invalid character in identifier
Unexpected indentation	
Indentation contains mixed spaces and tabs	
Performance	
Unnecessary comprehension	Unnecessary comprehension - \`sum\` and 'max 'can take a generator print(sum([i for i in range(1000) if (i % 3 == 0 or i % 5 == 0)]))
Equality comparison detected with singleton object	comparison to None should be 'if cond is None:' if (None == number or number < 2):
Expression not assigned	Expression "[print_and_stop_after_first_divisor_limit_found(triangle_nr, 500) for triangle_nr in create_triangle_number_list(1000000)]" is assigned to nothing
Security	
Audit required: Use of `exec`	`exec(open(filename).read())`

Documentation	
Use of `FIXME`/`XXX`/`TODO` encountered	
One-line docstring should fit on one line with quotes	
Missing docstring in public function	
Use of single quote detected in docstring	Use """triple double quotes"""
Docstring is over-indented	

The following are all Linting issues, which have already been scored at “Readability”.

So here no (additional) penalties.

Style	
expected 2 blank lines	
Trailing whitespace detected	
At least two spaces before inline comment	
Expected 2 blank lines after end of function or class	
No newline at end of file	
Multiple blank lines detected at end of the file	
Missing whitespace after ‘,’, ‘;’, or ‘:’	
Blank line contains whitespace	
Missing whitespace around operator	
Whitespace before opening parenthesis	
Missing whitespace around modulo operator	
Too many blank lines found	
Unexpected spaces around keyword / parameter equals	
Whitespace before colon	
Multiple spaces found before operator	
Multiple spaces found after operator	
Whitespace after opening parenthesis detected	
Whitespace before closing parenthesis	
Doc line too long	
Block comment should start with #	
Inline comment should start with #	
Indentation is not a multiple of four	
Indentation is not a multiple of four in comments	
Bad indentation detected	
Unexpected indentation in comments	
Indentation contains tabs	
Misplaced comparison constant	
Module level import not at the top of the file	
Unnecessary semicolon	
Statement ends with a semicolon	
Multiple imports on one line	
Inconsistent return statements	
Unnecessary parentheses after keyword	
line too long (> 88 characters)	