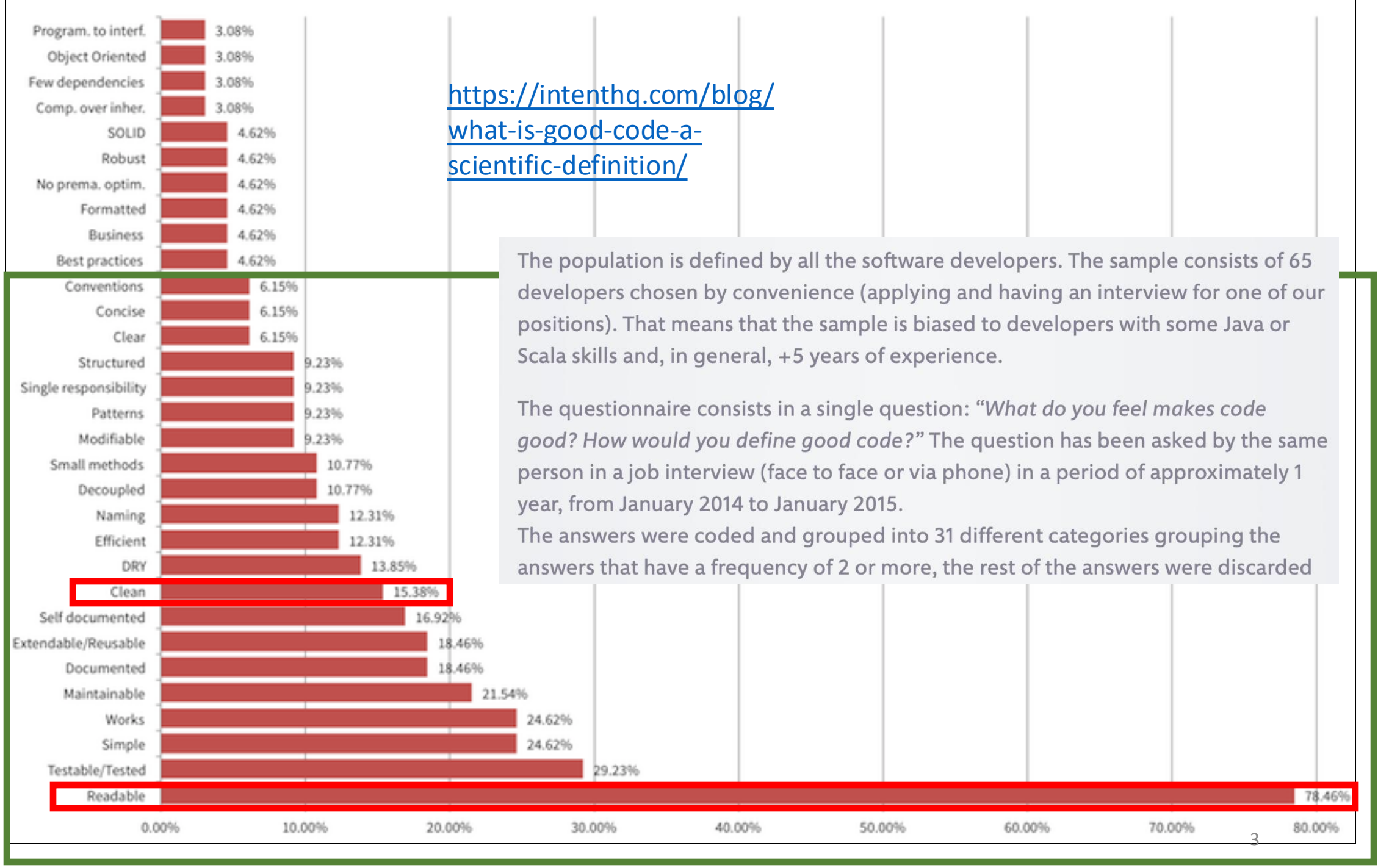
Bioinformatics Software development
Clean code

We would like to write good code

What makes code good?



https://intenthq.com/blog/what-is-good-code-a-scientific-definition/

The population is defined by all the software developers. The sample consists of 65 developers chosen by convenience (applying and having an interview for one of our positions). That means that the sample is biased to developers with some Java or Scala skills and, in general, +5 years of experience.

The questionnaire consists in a single question: *"What do you feel makes code good? How would you define good code?"* The question has been asked by the same person in a job interview (face to face or via phone) in a period of approximately 1 year, from January 2014 to January 2015.
The answers were coded and grouped into 31 different categories grouping the answers that have a frequency of 2 or more, the rest of the answers were discarded

Outline

- **Clean code**
  1. **What is clean code?**
     - **Definition & motivation**
     - **Challenges in writing clean code**
  2. **How to write clean code**
     - **Clean code principles**
     - **Tools for writing clean code**
     - **Pitfalls to avoid**
  3. **Examples from existing libraries and the class projects**
- **Happy path implementation – 10 min**
     - **Work on the happy path implementation**
       - Apply 2 clean code principles to existing code in your project

What is clean code?

- **Analogies:**
  - Clean code is code that is easy to understand and easy to change
  - Code is like humor. When you *have* to explain it, it’s bad
  - Clean code is just like a room that is well organized. You can easily find anything that you are looking for
  - Another metaphor is the newspaper. Clean code is arranged so that you can easily get a high-level idea what the code does (like reading newspaper headlines) and then decide what part you want to read about in more detail (jump to a lower-level method).

https://cvuorinen.net/2014/04/what-is-clean-code-and-why-should-you-care/

What is clean code?

"*I like my code to be elegant and efficient. The logic should be straightforward to make it hard for bugs to hide, the dependencies minimal to ease maintenance, error handling complete according to an articulated strategy, and performance close to optimal so as not to tempt people to make the code messy with unprincipled optimizations. Clean code does one thing well.*"

Bjarne Stroustrup, inventor of C++ and author of *The C++ Programming Language*

Martin, Robert C. *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.
https://www.oreilly.com/library-access/?next=/library/view/Clean-code-/9780136083238

What is clean code?

"*I could list all of the qualities that I notice in clean code, but there is one overarching quality that leads to all of them. Clean code always looks like it was written by someone who cares. There is nothing obvious that you can do to make it better. All of those things were thought about by the code's author, and if you try to imagine improvements, you're led back to where you are, sitting in appreciation of the code someone left for you—code left by someone who cares deeply about the craft.*"

Michael Feathers, author of Working Effectively with Legacy Code

Martin, Robert C. *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009. https://www.oreilly.com/library-access/?next=/library/view/Clean-code-/9780136083238

What is clean code?

"I like my code to be elegant and efficient. […] Clean code does one thing well."

Bjarne Stroustrup, inventor of C++

"Clean code reads like well-written prose."

Grady Booch, author of Object Oriented Analysis and Design with Applications

"Clean code can be read and enhanced by a developer other than its original author. It has unit and acceptance tests."

"Big" Dave Thomas, founder of OTI, godfather of the Eclipse strategy

"The stuff I design, if I'm successful, nobody will ever notice. Things will just work and be self-managing."

Radia Pearlman, "the Mother of the Internet," IEEE fellow, inventor of Spanning-Tree Protocol (STP), TRILL, and TORTIS, a programming language for children

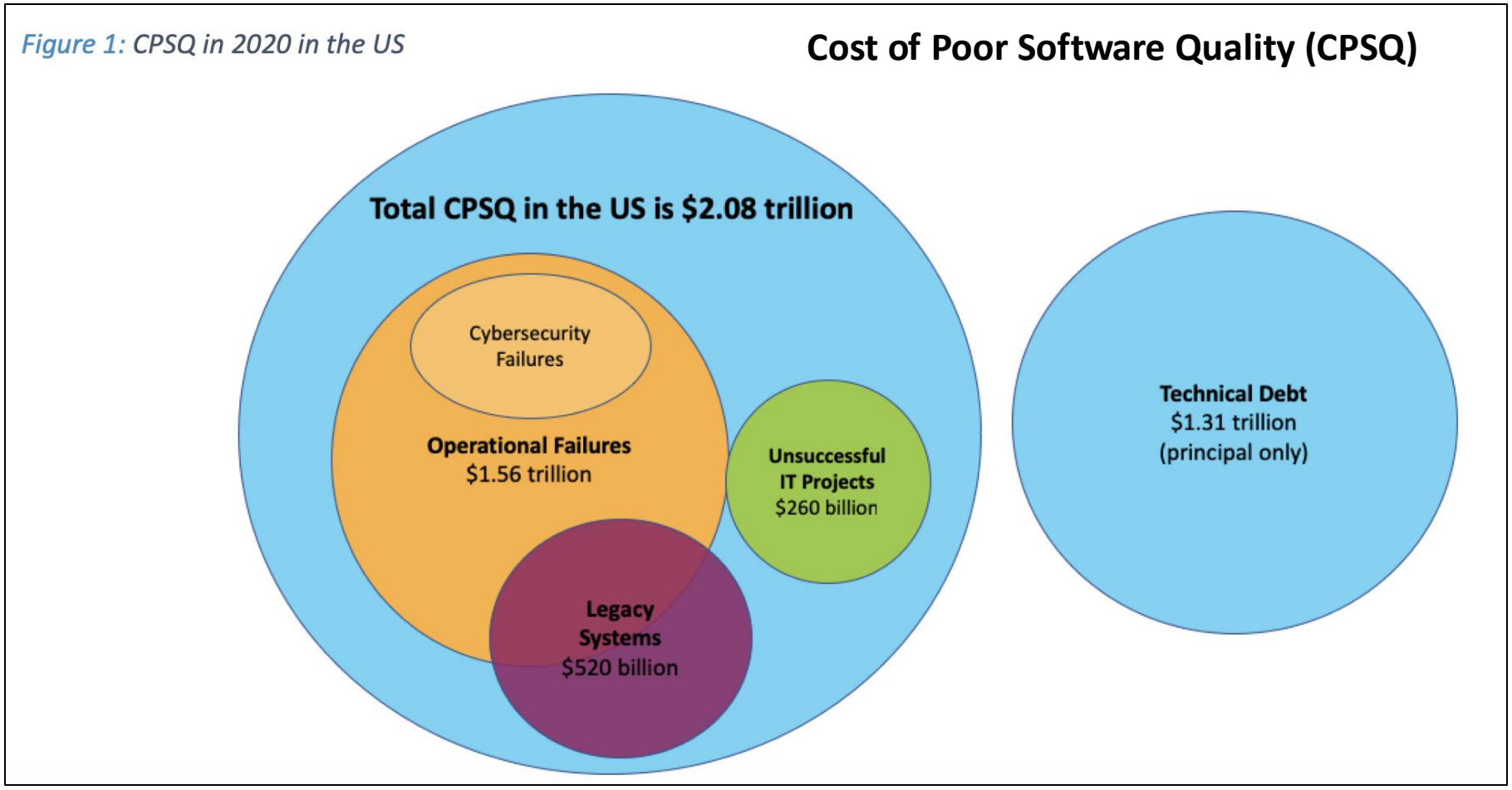
"Clean code always looks like it was written by someone who cares"

Michael Feathers, author of Working Effectively with Legacy Code

Martin, Robert C. *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.
https://www.oreilly.com/library-access/?next=/library/view/Clean-code-/9780136083238
https://lemelson.mit.edu/resources/radia-perlman
https://www.mentalfloss.com/article/53181/inspiring-quotes-10-influential-women-tech

Why is it important to write good code?

Figure 1: CPSQ in 2020 in the US

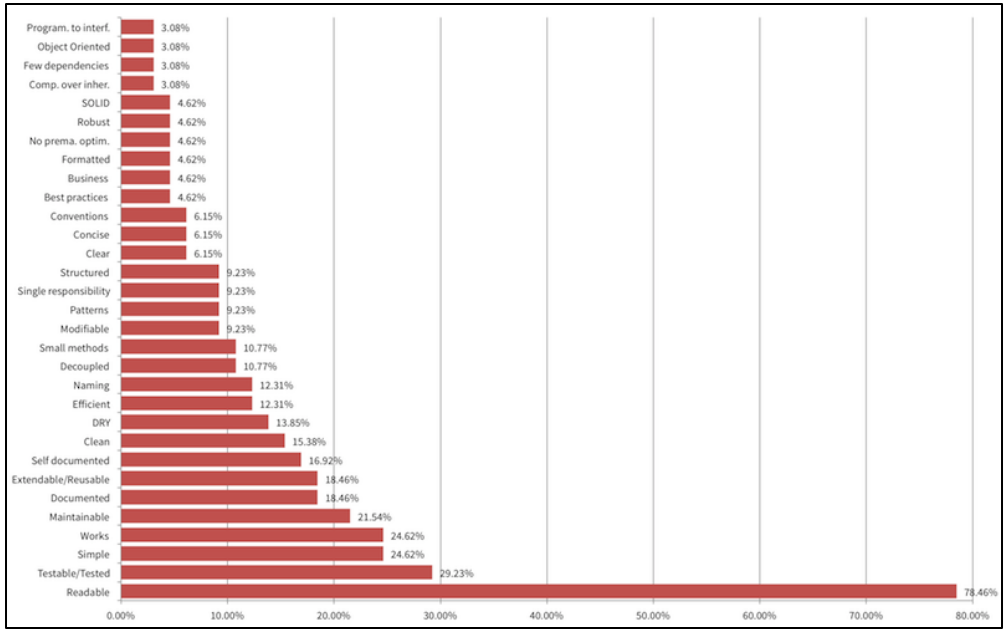


https://www.it-cisq.org/cisq-files/pdf/CPSQ-2020-report.pdf

What are the benefits of clean code?

- Analogies have been made that a programmer writing bad code is like a surgeon not washing their hands – the result has dire consequences – not always immediate though

- Writing clean code results in a software product that is more:
  - Usable
  - Readable
  - Maintainable
  - Testable
  - Robust
  - Extensible
  - Comprehensible
  - Cost efficient



- Write code that your future self would enjoy reading and changing after more than 6 months

Why doesn’t everybody write clean code?

- **Challenges in writing clean code**
  - Lack of knowledge of what clean code is
  - Longer time to finish development and get a working product
    - Time constraints
    - Priorities that do not include clean code
  - Unclear requirements
  - Not enough experience in writing clean code

https://betterprogramming.pub/the-real-reason-its-difficult-to-write-clean-code-gamedevunboxed-5ccdb06ca33f

Clean code principles

- It is easy to understand the execution flow of the entire application
- It is easy to understand how the different objects collaborate with each other
- It is easy to understand the role and responsibility of each class
- It is easy to understand what each method does
- It is easy to understand what is the purpose of each expression and variable

- Classes and methods are small and only have single responsibility
- Classes have clear and concise public APIs
- Classes and methods are predictable and work as expected

- Functions should not mix levels (high, low) of abstraction
- Functions should not have side effects
- Function output arguments are to be avoided and return values should be used instead

- The code is easily testable and has unit tests (or it is easy to write the tests)
- Tests are easy to understand and easy to change

https://cvuorinen.net/2014/04/what-is-clean-code-and-why-should-you-care/

Martin, Robert C. *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.
https://www.oreilly.com/library-access/?next=/library/view/Clean-code-/9780136083238

Clean code principles

- DRY (Don't repeat yourself)
  - Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.
- KISS (Keep it simple)
  - Most systems work best if they are kept simple, rather than made complicated.
- SoC (Separation of concerns)
  - SoC is a design principle for separating a computer program into distinct sections such that each section addresses a separate concern. A concern is a set of information that affects the code of a computer program.
- SOLID
  - SOLID is a mnemonic acronym for five design principles intended to make software designs more understandable, flexible, and maintainable.
  - The Single-responsibility principle: "A class should have one, and only one, reason to change."
  - The Open–closed principle: "Entities should be open for extension, but closed for modification."
  - The Liskov substitution principle: "Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it."
  - The Interface segregation principle: "A client should not be forced to implement an interface that it doesn't use."
  - The Dependency inversion principle: "Depend upon abstractions, not concretions."
- YAGNI (You Aren't Gonna Need It)
  - YAGNI is an Extreme Programming (XP) practice which states: "Always implement things when you actually need them, never when you just foresee that you need them."

Look at specific examples at the following links:

https://testdriven.io/blog/clean-code-python/
https://workat.tech/machine-coding/tutorial/introduction-clean-code-software-design-principles-nwu4qqc63e09

Clean code principles

- It is easy to understand the execution flow of the entire application
- It is easy to understand how the different objects collaborate with each other
- It is easy to understand the role and responsibility of each class
- It is easy to understand what each method does
- It is easy to understand what is the purpose of each expression and variable

- Classes and methods are small and only have single responsibility
- Classes have clear and concise public APIs
- Classes and methods are predictable, and work as expected

- Functions should not mix levels (high, low) of abstraction
- Functions should not have side effects
- Function output arguments are to be avoided and return values should be used instead

- The code is easily testable and has unit tests (or it is easy to write the tests)
- Tests are easy to understand and easy to change

https://cvuorinen.net/2014/04/what-is-clean-code-and-why-should-you-care/

Martin, Robert C. *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.
https://www.oreilly.com/library-access/?next=/library/view/Clean-code-/9780136083238

Tools for writing clean code – python

Write Pythonic code

Pythonic code is a set of idioms, adopted by the Python community. It simply means that you're using Python's idioms and paradigms well in order to make your code cleaner, readable, and highly performant.

Pythonic code includes:

- variable tricks
- list manipulation (initialization, slicing)
- dealing with functions
- explicit code

A non-Pythonic solution would be something like this:

```
n = 10
sum_all = 0

for i in range(1, n + 1):
    sum_all = sum_all + i

print(sum_all)  # 55
```

A more Pythonic solution might look like this:

```
n = 10
sum_all = sum(range(1, n + 1))

print(sum_all)  # 55
```

https://testdriven.io/blog/clean-code-python/

Tools for writing clean code – python

Leave the code cleaner than you found it

What is a Linter?

First, let's talk about lint. Those tiny, annoying little defects that somehow get all over your clothes. Clothes look and feel much better without all that lint. Your code is no different. Little mistakes, stylistic inconsistencies, and dangerous logic don't make your code feel great.

Linters analyze code to detect various categories of lint. Those categories can be broadly defined as the following:

1. Logical Lint
    - Code errors
    - Code with potentially unintended results
    - Dangerous code patterns
2. Stylistic Lint
    - Code not conforming to defined conventions

There are also code analysis tools that provide other insights into your code. While maybe not linters by definition, these tools are usually used side-by-side with linters. They too hope to improve the quality of the code.

Finally, there are tools that automatically format code to some specification. These automated tools ensure that our inferior human minds don't mess up conventions.

https://realpython.com/python-code-quality/

Tools for writing clean code

Here are some stand-alone linters categorized with brief descriptions:

Linter	Category	Description
Pylint	Logical & Stylistic	Checks for errors, tries to enforce a coding standard, looks for code smells
PyFlakes	Logical	Analyzes programs and detects various errors
pycodestyle	Stylistic	Checks against some of the style conventions in PEP 8
pydocstyle	Stylistic	Checks compliance with Python docstring conventions
Bandit	Logical	Analyzes code to find common security issues
MyPy	Logical	Checks for optionally-enforced static types

Before delving into your options, it's important to recognize that some "linters" are just multiple linters packaged nicely together. Some popular examples of those combo-linters are the following:

Flake8: Capable of detecting both logical and stylistic lint. It adds the style and complexity checks of pycodestyle to the logical lint detection of PyFlakes. It combines the following linters:

- PyFlakes
- pycodestyle (formerly pep8)
- Mccabe

Pylama: A code audit tool composed of a large number of linters and other tools for analyzing code. It combines the following:

- pycodestyle (formerly pep8)
- pydocstyle (formerly pep257)
- PyFlakes
- Mccabe
- Pylint
- Radon
- gjslint

https://realpython.com/python-code-quality/

Tools for writing clean code

And here are some code analysis and formatting tools:

Tool	Category	Description
Mccabe	Analytical	Checks McCabe complexity
Radon	Analytical	Analyzes code for various metrics (lines of code, complexity, and so on)
Black	Formatter	Formats Python code without compromise
Isort	Formatter	Formats imports by sorting alphabetically and separating into sections

https://realpython.com/python-code-quality/

Tools for writing clean code – Example

Code to clean: `code_with_lint.py`

```
"""
code_with_lint.py
Example Code with lots of lint!
"""
import io
from math import *


from time import time

some_global_var = 'GLOBAL VAR NAMES SHOULD BE IN ALL_C

def multiply(x, y):
    """
    This returns the result of a multiplation of the i
    """
    some_global_var = 'this is actually a local variab
    result = x* y
    return result
    if result == 777:
        print("jackpot!")

```

```
def is_sum_lucky(x, y):
    """This returns a string describing whether or not
    This function first makes sure the inputs are vali
    sum. Then, it will determine a message to return b
    that sum should be considered "lucky"
    """
    if x != None:
        if y is not None:
            result = x+y;
            if result == 7:
                return 'a lucky number!'
            else:
                return( 'an unlucky number!')

            return ('just a normal number')

class SomeClass:

    def __init__(self, some_arg,  some_other_arg, verb
        self.some_other_arg  =  some_other_arg
        self.some_arg        =  some_arg
        list_comprehension = [((100/value)*pi) for val
        time = time()
        from datetime import datetime
        date_and_time = datetime.now()
        return
```

https://realpython.com/python-code-quality/

Tools for writing clean code – Example

Linter	Command	Time
Pylint	pylint code_with_lint.py	1.16s
PyFlakes	pyflakes code_with_lint.py	0.15s
pycodestyle	pycodestyle code_with_lint.py	0.14s
pydocstyle	pydocstyle code_with_lint.py	0.21s

https://realpython.com/python-code-quality/

In the terminal/gitbash/anaconda prompt:

Check tool:

type pylint
which pylint
– using one of the following commands should return a path to pylint
– if the commands above do not return anything you need to install pylint

Install tool:

pip install pylint

Run tool:

pylint filename.py

Tools for writing clean code – Example

```
(base) mitrea@ltbin029 ~ % pylint ~/Downloads/code_with_lint.py
************* Module code_with_lint
Downloads/code_with_lint.py:31:0: W0301: Unnecessary semicolon (unnecessary-semicolon)
Downloads/code_with_lint.py:35:0: C0325: Unnecessary parens after 'return' keyword (superfluous-parens)
Downloads/code_with_lint.py:37:0: C0325: Unnecessary parens after 'return' keyword (superfluous-parens)
Downloads/code_with_lint.py:49:0: C0305: Trailing newlines (trailing-newlines)
Downloads/code_with_lint.py:6:0: W0622: Redefining built-in 'pow' (redefined-builtin)
Downloads/code_with_lint.py:6:0: W0401: Wildcard import math (wildcard-import)
Downloads/code_with_lint.py:11:0: C0103: Constant name "some_global_var" doesn't conform to UPPER_CASE naming style (invalid-name)
Downloads/code_with_lint.py:17:4: W0621: Redefining name 'some_global_var' from outer scope (line 11) (redefined-outer-name)
Downloads/code_with_lint.py:13:0: C0103: Argument name "x" doesn't conform to snake_case naming style (invalid-name)
Downloads/code_with_lint.py:13:0: C0103: Argument name "y" doesn't conform to snake_case naming style (invalid-name)
Downloads/code_with_lint.py:20:4: W0101: Unreachable code (unreachable)
Downloads/code_with_lint.py:17:4: W0612: Unused variable 'some_global_var' (unused-variable)
Downloads/code_with_lint.py:23:0: C0103: Argument name "x" doesn't conform to snake_case naming style (invalid-name)
Downloads/code_with_lint.py:23:0: C0103: Argument name "y" doesn't conform to snake_case naming style (invalid-name)
Downloads/code_with_lint.py:29:7: C0121: Comparison 'x != None' should be 'x is not None' (singleton-comparison)
Downloads/code_with_lint.py:32:12: R1705: Unnecessary "else" after "return" (no-else-return)
Downloads/code_with_lint.py:23:0: R1710: Either all return statements in a function should return an expression, or none of them should.
(inconsistent-return-statements)
Downloads/code_with_lint.py:39:0: C0115: Missing class docstring (missing-class-docstring)
Downloads/code_with_lint.py:45:8: W0621: Redefining name 'time' from outer scope (line 9) (redefined-outer-name)
Downloads/code_with_lint.py:45:15: E0601: Using variable 'time' before assignment (used-before-assignment)
Downloads/code_with_lint.py:46:8: C0415: Import outside toplevel (datetime.datetime) (import-outside-toplevel)
Downloads/code_with_lint.py:41:4: R1711: Useless return at end of function or method (useless-return)
Downloads/code_with_lint.py:41:50: W0613: Unused argument 'verbose' (unused-argument)
Downloads/code_with_lint.py:44:8: W0612: Unused variable 'list_comprehension' (unused-variable)
Downloads/code_with_lint.py:47:8: W0612: Unused variable 'date_and_time' (unused-variable)
Downloads/code_with_lint.py:39:0: R0903: Too few public methods (0/2) (too-few-public-methods)
Downloads/code_with_lint.py:5:0: W0611: Unused import io (unused-import)
Downloads/code_with_lint.py:6:0: W0614: Unused import acos from wildcard import (unused-wildcard-import)
Downloads/code_with_lint.py:6:0: W0614: Unused import acosh from wildcard import (unused-wildcard-import)
Downloads/code_with_lint.py:6:0: W0614: Unused import asin from wildcard import (unused-wildcard-import)
Downloads/code_with_lint.py:6:0: W0614: Unused import asinh from wildcard import (unused-wildcard-import)
Downloads/code_with_lint.py:6:0: W0614: Unused import atan from wildcard import (unused-wildcard-import)
Downloads/code_with_lint.py:6:0: W0614: Unused import atan2 from wildcard import (unused-wildcard-import)
```

```
Downloads/code_with_lint.py:6:0: W0614: Unused import remainder from wildcard import (unused-wildcard-import)
Downloads/code_with_lint.py:6:0: W0614: Unused import sin from wildcard import (unused-wildcard-import)
Downloads/code_with_lint.py:6:0: W0614: Unused import sinh from wildcard import (unused-wildcard-import)
Downloads/code_with_lint.py:6:0: W0614: Unused import sqrt from wildcard import (unused-wildcard-import)
Downloads/code_with_lint.py:6:0: W0614: Unused import tan from wildcard import (unused-wildcard-import)
Downloads/code_with_lint.py:6:0: W0614: Unused import tanh from wildcard import (unused-wildcard-import)
Downloads/code_with_lint.py:6:0: W0614: Unused import tau from wildcard import (unused-wildcard-import)
Downloads/code_with_lint.py:6:0: W0614: Unused import trunc from wildcard import (unused-wildcard-import)
Downloads/code_with_lint.py:6:0: W0614: Unused import ulp from wildcard import (unused-wildcard-import)
Downloads/code_with_lint.py:9:0: W0611: Unused time imported from time (unused-import)
```

https://realpython.com/python-code-quality/

Tools for writing clean code – Example

```
(base) mitrea@ltbin029 session07_clean_code % pylint code_with_lint.py
************* Module code_with_lint
code_with_lint.py:31:0: W0301: Unnecessary semicolon (unnecessary-semicolon)
code_with_lint.py:35:0: C0325: Unnecessary parens after 'return' keyword (superfluous-parens)
code_with_lint.py:37:0: C0325: Unnecessary parens after 'return' keyword (superfluous-parens)
code_with_lint.py:49:0: C0305: Trailing newlines (trailing-newlines)
code_with_lint.py:6:0: W0622: Redefining built-in 'pow' (redefined-builtin)
```

code_with_lint.py

Users > mitrea > Documents > academic_docs > CLASSES > WINTER_2024

```
29      if x != None:
30          if y is not None:
31              result = x+y;
32              if result == 7:
33                  return 'a lucky number!'
34              else:
35                  return( 'an unlucky number!')
36
37              return ('just a normal number')
```

https://realpython.com/python-code-quality/

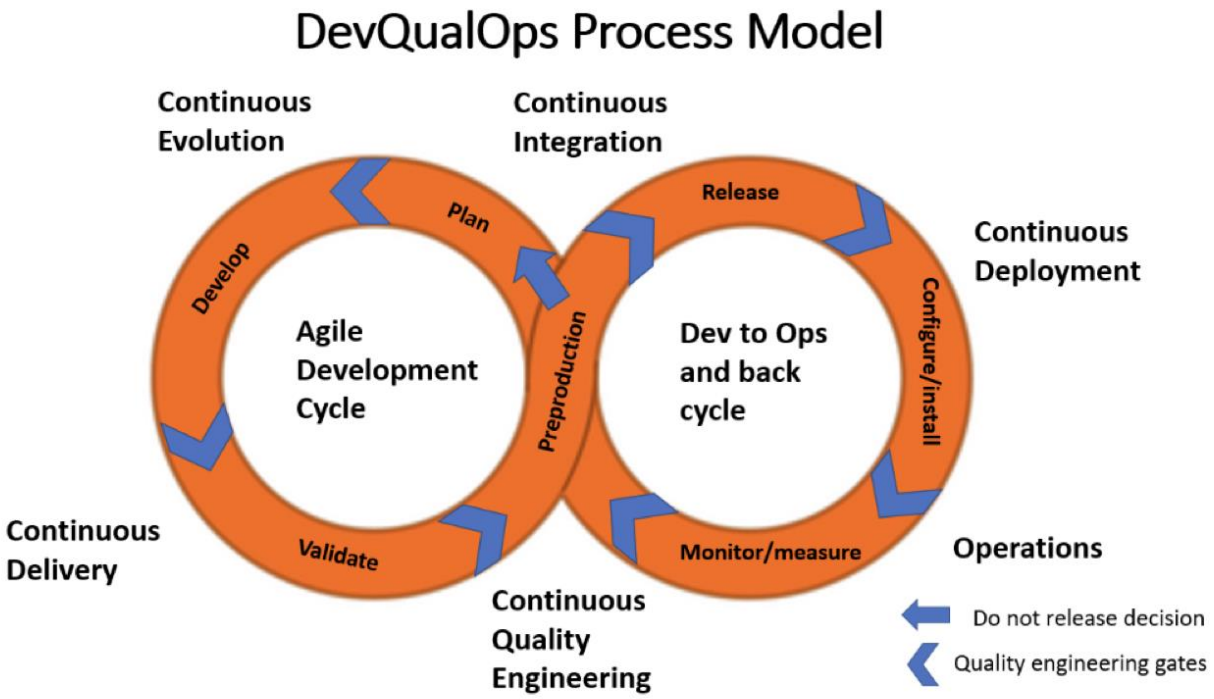
Pitfalls to avoid in order to write clean code

- **Clean code is most at risk when:**
  - There is a deadline pressure – which in academia is always
    - Try to prioritize and practice writing clean code
  - Looking for perfection – nothing is perfect
    - Try to achieve a very good level of clean code
      - If you spend 10 minutes looking for a way to improve a specific code block following clean code principles and do not find an issue to fix, then it is good enough
  - There is unnecessary complexity – simplify, simplify
  - The solution is too general – be specific

Examples of mistakes and how to fix them:

https://crocoblock.com/blog/writing-clean-code-tips/

Figure 8: The New DevQualOps Process Model



In this new model, we have added the following features:

- Defined quality objectives, many of which are measurable
- The traditional QA/testing role is shifted left to become a broader and more strategic team role in software quality engineering (SQE)
- Identified quality characteristic champions are on the team

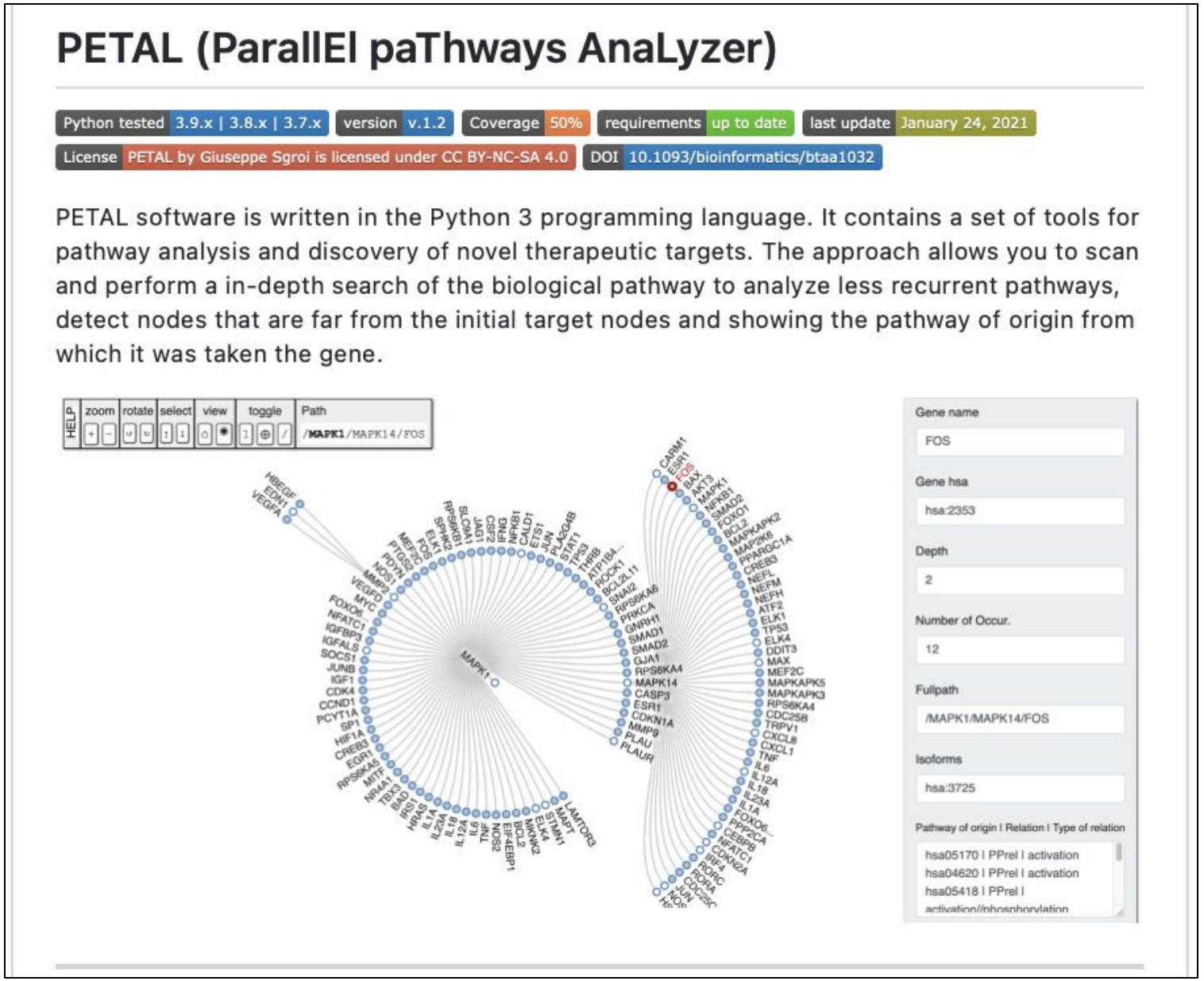
https://www.it-cisq.org/cisq-files/pdf/CPSQ-2020-report.pdf

How does this work in practice?

Examples from existing libraries

PETAL (ParallEl paThways AnaLyzer)

https://github.com/Pex2892/PETAL



Examples from existing libraries

PETAL (ParallEl paThways AnaLyzer)

https://github.com/Pex2892/PETAL

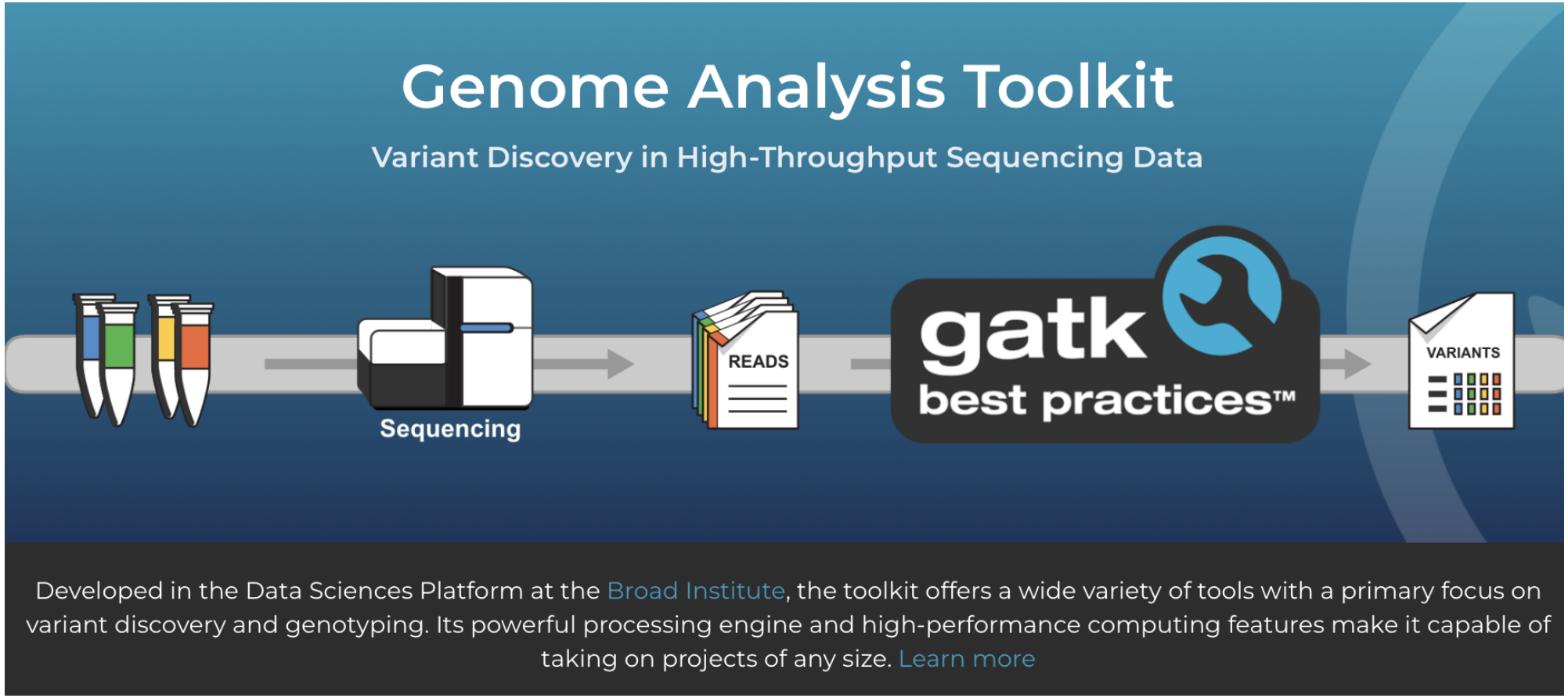
https://github.com/Pex2892/PETAL/blob/master/analysis.py

```
10  def run_analysis(starting_depth):
11      for deep in range(starting_depth, gl.deep_input + 1):
12          if deep == 1:
13              gene_info = get_gene_info_from_name(gl.gene_input, gl.CSV_GENE_HSA)
14              if gene_info is not None:
15                  # set globals variables
16                  gl.gene_input = check_gene_and_alias(gl.gene_input, gene_info[1])
17                  gl.gene_input_hsa = gene_info[0]
18                  gl.gene_input_url = gene_info[2]
19
20                  # read initial pathway, create and add genes to csv
21                  list_rows_df_returned = read_kgml(deep, gl.pathway_input, gl.gene_input,
22                                                    gl.gene_input_hsa, gl.gene_input, 1)
23
24                  # add n genes found to the dataframe
25                  unified([list_rows_df_returned])
26
27                  # retrive other list pathways in reference to initial pathway
28                  list_pathways_this_gene = read_gene_txt(gl.gene_input_hsa)
29
30                  # The pathway set as input from the config file is removed
31                  if gl.pathway_input in list_pathways_this_gene:
32                      list_pathways_this_gene.remove(gl.pathway_input)
33
34                  if len(list_pathways_this_gene) > 0:
35                      # process single gene on each CPUs available
36                      list_rows_df_returned = Parallel(n_jobs=gl.num_cores_input)(delayed(analysis_deep_n)(
37                          deep, gl.gene_input, gl.gene_input_hsa, pathway_this_gene, gl.gene_input, 1)
38                                                                                  for pathway_this_gene in set_progress_bar(
39                          f'[Deep: {deep}]', str(len(list_pathways_this_gene)))(list_pathways_this_gene))
40
41                      unified(list_rows_df_returned)
42                  else:
43                      print('[Deep: 1] Only directly connected genes were found')
44              else:
45                  print('The starting gene was not found in the chosen pathway! Try to verify manually.')
46                  exit(1)
47          else:
48              # Retrieve the genes found at depth-1, avoiding the input gene
49              df_genes_resulted = (
```

Code with issues

Long function
Complex function
Commented code
Side effects
Untestable code

Examples from existing libraries



https://gatk.broadinstitute.org/hc/en-us
https://github.com/broadinstitute/gatk

Examples from existing libraries

gatk (Genome analysis toolkit)

https://github.com/broadinstitute/gatk

https://github.com/broadinstitute/gatk/blob/master/src/main/python/org/broadinstitute/hellbender/vqsr_cnn/vqsr_cnn/plots.py

master / gatk / src / main / python / org / broadinstitute / hellbender / vqsr_cnn / vqsr_cnn / **plots.py** / Jump to — Go to file

lucidtronix Add CNNScoreVariants with 2D model, CNNVariantTrain, CNNVariantWriteT... — Latest commit 6979f37 on Mar 27, 2018 — History

1 contributor

180 lines (147 sloc) | 6.67 KB — Raw | Blame

```
# plots.py
#
# Plotting code for Variant Filtration with Neural Nets
# This includes evaluation plots like Precision and Recall curves,
# various flavors of Receiver Operating Characteristic (ROC curves),
# As well as graphs of the metrics that are watched during neural net training.
#
# December 2016
# Sam Friedman
# sam@broadinstitute.org

# Imports
import os
import math
import matplotlib
import numpy as np
matplotlib.use('Agg') # Need this to write images from the GSA servers.  Order matters:
import matplotlib.pyplot as plt # First import matplotlib, then use Agg, then import plt
from sklearn.metrics import roc_curve, auc, roc_auc_score, precision_recall_curve, average_precision_score

image_ext = '.png'

color_array = ['red', 'indigo', 'cyan', 'pink', 'purple']
key_colors = {
    'Neural Net':'green', 'CNN_SCORE':'green', 'CNN_2D':'green',
    'Heng Li Hard Filters':'lightblue',
    'GATK Hard Filters':'orange','GATK Signed Distance':'darksalmon',
    'VQSR gnomAD':'cornflowerblue', 'VQSR Single Sample':'blue', 'VQSLOD':'cornflowerblue',
    'Deep Variant':'magenta', 'QUAL':'magenta', 'DEEP_VARIANT_QUAL':'magenta',
    'Random Forest':'darkorange',
```

Good code

Readable function
Nice organization
Good comments
No surprises – relevant comment

Examples from existing libraries

gatk (Genome analysis toolkit)

https://github.com/broadinstitute/gatk

https://github.com/broadinstitute/gatk/blob/master/src/main/python/org/broadinstitute/hellbender/vqsr_cnn/vqsr_cnn/arguments.py

master gatk / src / main / python / org / broadinstitute / hellbender / vqsr_cnn / vqsr_cnn / **arguments.py** / Jump to Go to file

lucidtronix type hints in inference (#5548) Latest commit 19ddf36 on Jan 29, 2019 History

1 contributor

213 lines (182 sloc) | 12.2 KB Raw Blame

```python
import argparse
import numpy as np

import keras.backend as K

from . import defines


def parse_args():
    """Parse command line arguments.

    The args namespace is used promiscuously in this module.
    Its fields control the tensor definition, dataset generation, training, file I/O and evaluation.
    Some of the fields are typically dicts or lists that are not actually set on the command line,
    but via a companion argument also in the namespace.
    For example, input_symbols is set via the input_symbol_set string
    and, annotations is set via the annotation_set string.
    Here we also seed the random number generator.
    The keras image data format is set here as well via the channels_last or channels_first arguments.

    Returns:
        namespace: The args namespace that is used throughout this module.
    """
    parser = argparse.ArgumentParser()

    # Tensor defining arguments
    parser.add_argument('--tensor_name', default='read_tensor', choices=defines.TENSOR_MAPS_1D+defines.TENSOR_MAPS_2D,
                        help='String key which identifies the map from tensor channels to their meaning.')
    parser.add_argument('--labels', default=defines.SNP_INDEL_LABELS,
                        help='Dict mapping label names to their index within label tensors.')
```

Good code but some issues

Long function due to the number of arguments
Code is too general/flexible

Examples from existing libraries

gatk (Genome analysis toolkit)

https://github.com/broadinstitute/gatk

https://github.com/broadinstitute/gatk/blob/master/src/main/python/org/broadinstitute/hellbender/vqsr_cnn/vqsr_cnn/inference.py

```
56  # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
57  # ~~~~~~~ Inference ~~~~~~~~~~~~~~~
58  # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
59  def score_and_write_batch(model: keras.Model,
60                            file_out: TextIO,
61                            batch_size: int,
62                            python_batch_size: int,
63                            tensor_type: str,
64                            annotation_set: str,
65                            window_size: int,
66                            read_limit: int,
67                            tensor_dir: str = '') -> None:
68      """Score a batch of variants with a CNN model. Write tab delimited temp file with scores.
69
70      This function is tightly coupled with the CNNScoreVariants.java
71      It requires data written to the fifo in the order given by transferToPythonViaFifo
72
73      Arguments
74          model: a keras model
75          file_out: The temporary VCF-like file where variants scores will be written
76          batch_size: The total number of variants available in the fifo
77          python_batch_size: the number of variants to process in each inference
78          tensor_type: The name for the type of tensor to make
79          annotation_set: The name for the set of annotations to use
80          window_size: The size of the context window of genomic bases, i.e the width of the tensor
81          read_limit: The maximum number of reads to encode in a tensor, i.e. the height of the tensor
82          tensor_dir : If this path exists write hd5 files for each tensor (optional for debugging)
83      """
84      annotation_batch = []
85      reference_batch = []
86      variant_types = []
87      variant_data = []
88      read_batch = []
89      for _ in range(batch_size):
90          fifo_line = tool.readDataFIFO()
91          fifo_data = fifo_line.split(defines.DATA_TYPE_SEPARATOR)
92
93          variant_data.append(fifo_data[CONTIG_FIFO_INDEX] + defines.DATA_TYPE_SEPARATOR
94                              + fifo_data[POS_FIFO_INDEX] + defines.DATA_TYPE_SEPARATOR
95                              + fifo_data[REF_FIFO_INDEX] + defines.DATA_TYPE_SEPARATOR + fifo_data[ALT_FIFO_INDEX])
96          reference_batch.append(reference_string_to_tensor(fifo_data[REF_STRING_FIFO_INDEX]))
97          annotation_batch.append(annotation_string_to_tensor(annotation_set, fifo_data[ANNOTATION_FIFO_INDEX]))
```

Function doing two things
Score and write batch

Examples from existing libraries

gatk (Genome analysis toolkit)

```
147  def reference_string_to_tensor(reference: str) -> np.ndarray:
148      dna_data = np.zeros((len(reference), len(defines.DNA_SYMBOLS)))
149      for i,b in enumerate(reference):
150          if b in defines.DNA_SYMBOLS:
151              dna_data[i, defines.DNA_SYMBOLS[b]] = 1.0
152          elif b in defines.AMBIGUITY_CODES:
153              dna_data[i] = defines.AMBIGUITY_CODES[b]
154          elif b == '\x00':
155              break
156          else:
157              raise ValueError('Error! Unknown code:', b)
158      return dna_data
```

https://github.com/broadinstitute/gatk

https://github.com/broadinstitute/gatk/blob/master/src/main/python/org/broadinstitute/hellbender/vqsr_cnn/vqsr_cnn/inference.py

Function started nicely and then used a constant value and a break

Resources

- https://www.freecodecamp.org/news/clean-coding-for-beginners/
- https://intenthq.com/blog/what-is-good-code-a-scientific-definition/
- https://medium.com/@kevin.r.webster/what-is-good-code-11c401e8a1ec
- https://workat.tech/machine-coding/tutorial/software-design-principles-dry-yagni-eytrxfhz1fla
- https://garywoodfine.com/what-is-clean-code/
- https://testdriven.io/blog/clean-code-python/
- https://qntm.org/clean
- https://realpython.com/python-code-quality/
- https://www.freecodecamp.org/news/these-tools-will-help-you-write-clean-code-da4b5401f68e/
- https://dzone.com/articles/clean-code-explanation-benefits-amp-examples
- https://workat.tech/machine-coding/tutorial/introduction-clean-code-software-design-principles-nwu4qqc63e09
- https://workat.tech/machine-coding/tutorial/solid-design-principles-8yu7bjegrxs5
- https://cvuorinen.net/2014/04/what-is-clean-code-and-why-should-you-care/
- https://www.sonarsource.com/blog/power-of-clean-code/
- https://dzone.com/articles/clean-code-explanation-benefits-amp-examples
- https://ropensci.org/blog/2020/04/21/rclean/
- https://medium.com/swlh/clean-code-famous-quotes-6d62d396cfe0

Tools for writing clean code – R

`lintr` provides static code analysis for R. It checks for adherence to a given style, identifying syntax errors and possible semantic issues, then reports them to you so you can take action.

`lintr` is complementary to the `styler` package which automatically restyles code, eliminating some of the problems that `lintr` can detect.

Installation

Install the stable version from CRAN:

```
install.packages("lintr")
```

Or the development version from GitHub:

```
devtools::install_github("r-lib/lintr")
```

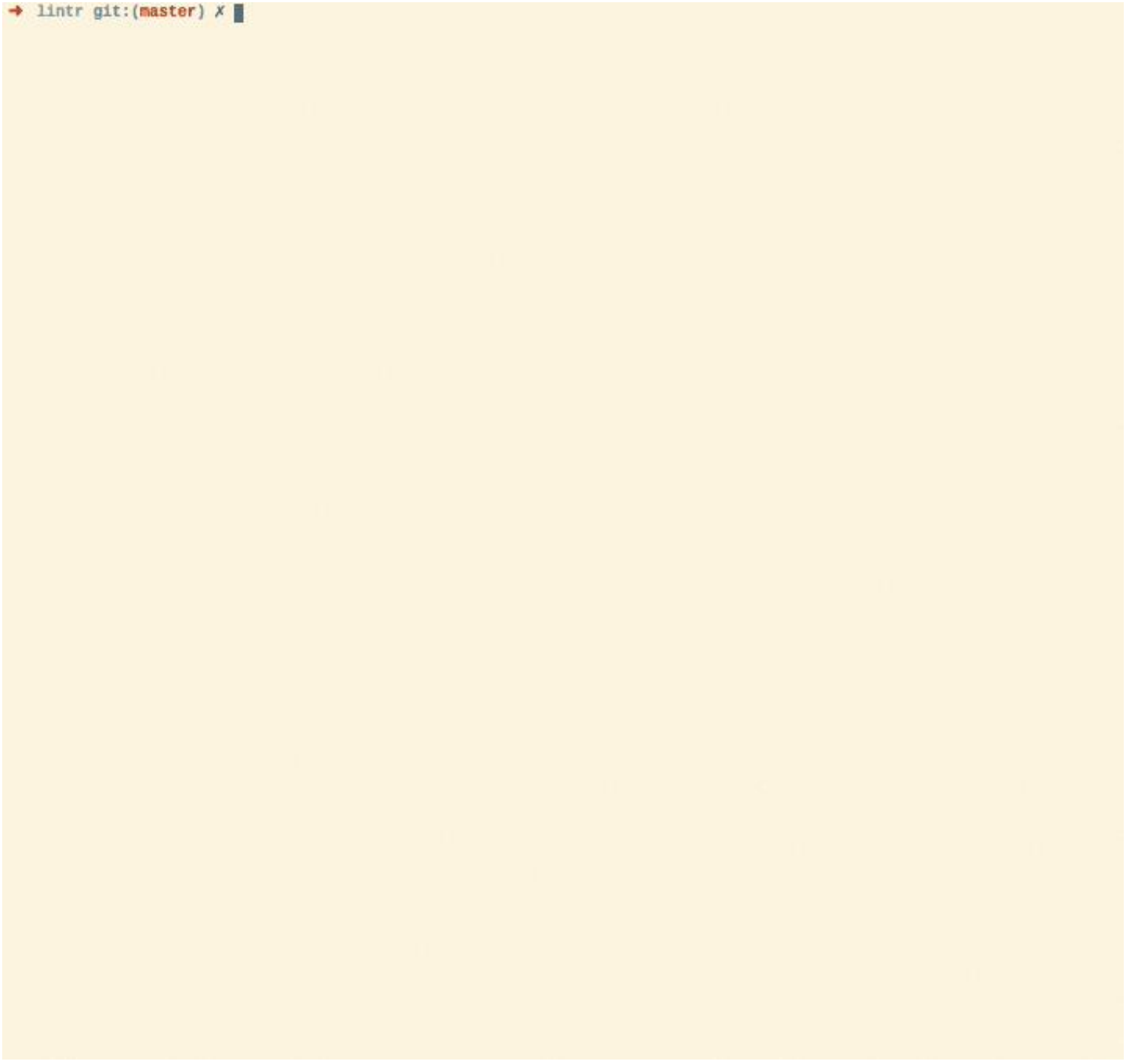
Usage

```
# in a project:
lintr::use_lintr(type = "tidyverse")
usethis::use_github_action("lint-project")
lintr::lint_dir()

# in a package:
lintr::use_lintr(type = "tidyverse")
usethis::use_github_action("lint")
lintr::lint_package()
```

https://lintr.r-lib.org

Watch lintr in action in the following animation:



https://lintr.r-lib.org

Check lintr output by running the tool yourself

In an R console run the following:

```
> library(lintr)
> lint("~/Downloads/bad.R")
```

```
/Users/mitrea/Downloads/bad.R:1:5: style: [assignment_linter] Use <-, not =, for assignment.
fun = function(one)
    ^
/Users/mitrea/Downloads/bad.R:2:1: style: [brace_linter] Opening curly braces should never go on their own line and should always be followed by a new line.
{
^
/Users/mitrea/Downloads/bad.R:3:3: style: [object_name_linter] Variable and function name style should be snake_case or symbols.
  one.plus.one <- oen + 1
  ^~~~~~~~~~~~
/Users/mitrea/Downloads/bad.R:3:3: warning: [object_usage_linter] local variable 'one.plus.one' assigned but may not be used
  one.plus.one <- oen + 1
  ^~~~~~~~~~~~
/Users/mitrea/Downloads/bad.R:3:19: warning: [object_usage_linter] no visible binding for global variable 'oen'
  one.plus.one <- oen + 1
                  ^~~
/Users/mitrea/Downloads/bad.R:4:11: style: [object_name_linter] Variable and function name style should be snake_case or symbols.
  four <- newVar <- matrix(1:10,nrow = 2)
          ^~~~~~
/Users/mitrea/Downloads/bad.R:4:11: warning: [object_usage_linter] local variable 'newVar' assigned but may not be used
  four <- newVar <- matrix(1:10,nrow = 2)
          ^~~~~~
/Users/mitrea/Downloads/bad.R:4:33: style: [commas_linter] Commas should always have a space after.
  four <- newVar <- matrix(1:10,nrow = 2)
                                ^
/Users/mitrea/Downloads/bad.R:5:8: style: [spaces_inside_linter] Do not place spaces after square brackets.
  four[ 1, ]
       ^
/Users/mitrea/Downloads/bad.R:6:10: style: [single_quotes_linter] Only use double-quotes.
  txt <- 'hi'
         ^~~~
/Users/mitrea/Downloads/bad.R:7:3: warning: [object_usage_linter] local variable 'three' assigned but may not be used
  three <- two+ 1
  ^~~~~
/Users/mitrea/Downloads/bad.R:7:12: warning: [object_usage_linter] no visible binding for global variable 'two'
  three <- two+ 1
           ^~~
/Users/mitrea/Downloads/bad.R:7:15: style: [infix_spaces_linter] Put spaces around all infix operators.
  three <- two+ 1
              ^
/Users/mitrea/Downloads/bad.R:8:5: style: [spaces_left_parentheses_linter] Place a space before left parenthesis, except in a function call.
  if(txt == 'hi') 4
    ^
/Users/mitrea/Downloads/bad.R:8:13: style: [single_quotes_linter] Only use double-quotes.
  if(txt == 'hi') 4
            ^~~~
/Users/mitrea/Downloads/bad.R:9:4: style: [brace_linter] Closing curly-braces should always be on their own line, unless they are followed by an else.
  5}
   ^
/Users/mitrea/Downloads/bad.R:9:5: style: [trailing_whitespace_linter] Trailing whitespace is superfluous.
  5}
    ^~
/Users/mitrea/Downloads/bad.R:10:1: style: [brace_linter] Opening curly braces should never go on their own line and should always be followed by a new line.
{
^
/Users/mitrea/Downloads/bad.R:10:1: error: [error] unexpected end of input
{
```

https://lintr.r-lib.org

Tools for writing clean code – Rclean

The goal of the R package Rclean is to provide a automated tool to help scientists more easily write better code. Specifically, Rclean has been designed to facilitate the isolation of the code needed to produce one or more results, because more often then not, when someone is writing an R script, the ultimate goal is analytical results for inference, such as a set of statistical analyses and associated figures and tables. As the investigative process is inherently iterative, this set of results is nearly always a subset of a much larger set of possible ways to explore a dataset. There are many statistical tests and visualizations and other representations that can be employed in a myriad of ways depending on how the data are processed. This commonly leads to lengthy, complicated scripts from which researchers manually subset results, but which are likely never to be refactored because of the difficulty in disentangling the code.

https://ropensci.org/blog/2020/04/21/rclean/

Tools for writing clean code – Rclean

Installation

Through the helpful feedback from the rOpenSci community, the package has recently passed software review and a supporting article was recently published in the Journal of Open Source Software [1], in which you can find more details about the package. The package is hosted through the rOpenSci organization on GitHub, and the package can be installed using the devtools package [2] directly from the repository (https://github.com/ROpenSci/Rclean).

```
library(devtools)
install_github("ROpenSci/Rclean")
```

If you do not already have RGraphviz, you will need to install it using the following code *before* installing Rclean:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("Rgraphviz")
```

https://ropensci.org/blog/2020/04/21/rclean/

Tools for writing clean code

Rclean

`long_script.R`

https://ropensci.org/blog/2020/04/21/rclean/

Isolating code for a set of results

Analytical scripts that have not been refactored are often both long and complicated. However, a script doesn't need to be long to be complicated. The following example script presents some challenges such that even though it's not a long script, picking through it to get a result would likely prove to be frustrating.

```
library(stats)
x <- 1:100
x <- log(x)
x <- x * 2
x <- lapply(x, rep, times = 4)
### This is a note that I made for myself.
### Next time, make sure to use a different analysis.
### Also, check with someone about how to run some other analysis.
x <- do.call(cbind, x)

### Now I'm going to create a different variable.
### This is the best variable the world has ever seen.

x2 <- sample(10:1000, 100)
x2 <- lapply(x2, rnorm)

### Wait, now I had another thought about x that I want to work through.

x <- x * 2
colnames(x) <- paste0("X", seq_len(ncol(x)))
rownames(x) <- LETTERS[seq_len(nrow(x))]
x <- t(x)
x[, "A"] <- sqrt(x[, "A"])
```

Tools for writing clean code

Rclean – extract code to compute `fit_sqrt_A`

```
BiocManager::install("Rgraphviz")
devtools::install_github("ROpenSci/Rclean")
install.packages("prettycode")

library(Rclean)
clean(script = "~/Downloads/long_script.R",
      vars = "fit_sqrt_A")
```

```
x <- 1:100
x <- log(x)
x <- x * 2
x <- lapply(x, rep, times = 4)
x <- do.call(cbind, x)
x <- x * 2
colnames(x) <- paste0("X", seq_len(ncol(x)))
rownames(x) <- LETTERS[seq_len(nrow(x))]
x <- t(x)
x[, "A"] <- sqrt(x[, "A"])
for (i in seq_along(colnames(x))) {
  set.seed(17)
  x[, i] <- x[, i] + runif(length(x[, i]), -1, 1)
}
x[, 1] <- x[, 1] * 2 + 10
x[, 2] <- x[, 1] + x[, 2]
x[, "A"] <- x[, "A"] * 2
x <- data.frame(x)
fit_sqrt_A <- lm(I(sqrt(A)) ~ B, data = x)
```

```
library(Rclean)
script <- system.file("example",
        "long_script.R",
        package = "Rclean")
clean(script, "fit_sqrt_A")
```

```
x <- 1:100
x <- log(x)
x <- x * 2
x <- lapply(x, rep, times = 4)
x <- do.call(cbind, x)
x <- x * 2
colnames(x) <- paste0("X", seq_len(ncol(x)))
rownames(x) <- LETTERS[seq_len(nrow(x))]
x <- t(x)
x[, "A"] <- sqrt(x[, "A"])
```

https://ropensci.org/blog/2020/04/21/rclean/