|  |
| --- |
| Test-Python-Project-Linux  Version 1.0  Code analysis |

|  |
| --- |
| **By: default**  **2022-05-31** |

# Content

[Content 1](#_Toc97156258)

[Introduction 2](#_Toc97156259)

[Configuration 2](#_Toc97156260)

[Synthesis 3](#_Toc97156261)

[Analysis Status 3](#_Toc97156262)

[Quality gate status 3](#_Toc97156263)

[Metrics 3](#_Toc97156264)

[Tests 3](#_Toc97156265)

[Detailed technical debt 3](#_Toc97156266)

[Metrics Range 5](#_Toc97156267)

[Volume 5](#_Toc97156268)

[Issues 6](#_Toc97156269)

[Charts 6](#_Toc97156270)

[Issues count by severity and type 8](#_Toc97156271)

[Issues List 8](#_Toc97156272)

[Security Hotspots 9](#_Toc97156273)

[Security hotspots count by category and priority 9](#_Toc97156274)

[Security hotspots List 9](#_Toc97156275)

# Introduction

This document contains results of the code analysis of Test-Python-Project-Linux.

# Configuration

* Quality Profiles
  + Names: Sonar way [CSS]; Sonar way [JavaScript]; Sonar way [Python]; Sonar way [HTML]; Sonar way [XML];
  + Files: AYBqrOruN0\_tCdZCZzzU.json; AYBqrOzON0\_tCdZCZz9Z.json; AYBqrO2WN0\_tCdZCZ0EW.json; AYBqrPFyN0\_tCdZCZ0ql.json; AYBqrPHMN0\_tCdZCZ0rb.json;
* Quality Gate
  + Name: Sonar way
  + File: Sonar way.xml

# Synthesis

## Analysis Status

|  |  |  |  |
| --- | --- | --- | --- |
| Reliability | Security | Security Review | Maintainability |
| C.png | **A.png** | **E.png** | **A.png** |

## Quality gate status

|  |  |
| --- | --- |
| Quality Gate Status | **ERROR.png** |

|  |  |
| --- | --- |
| Metric | Value |
| Reliability Rating on New Code | OK |
| Security Rating on New Code | OK |
| Maintainability Rating on New Code | OK |
| Coverage on New Code | ERROR (0.0% is less than 80%) |
| Duplicated Lines (%) on New Code | ERROR (74.1% is greater than 3%) |

## Metrics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Coverage | Duplication | Comment  density | Median number of lines of code per file | Adherence to coding standard |
| 0.0 % | **83.3 %** | **1.3 %** | **124.0** | **98.3 %** |

## Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Total | Success Rate | Skipped | Errors | Failures |
| 0 | **0 %** | **0** | **0** | **0** |

## Detailed technical debt

|  |  |  |  |
| --- | --- | --- | --- |
| Reliability | Security | Maintainability | Total |
| 0d 3h 13min | - | 1d 1h 40min | 1d 4h 53min |

## Metrics Range

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Cyclomatic  Complexity | Cognitive  Complexity | Lines of code per file | Comment  density (%) | Coverage | Duplication (%) |
| Min | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Max | 115.0 | 217.0 | 44850.0 | 32.9 | 0.0 | 100.0 |

## Volume

|  |  |
| --- | --- |
| Language | Number |
| CSS | 5520 |
| Python | 989 |
| HTML | 38331 |
| XML | 10 |
| Total | 44850 |

# Issues

## Charts

## Issues count by severity and type

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type / Severity | INFO | MINOR | MAJOR | CRITICAL | BLOCKER |
| BUG | 0 | 36 | 82 | 0 | 0 |
| VULNERABILITY | 0 | 0 | 0 | 0 | 0 |
| CODE\_SMELL | 0 | 16 | 74 | 30 | 0 |

## Issues List

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Description | Type | Severity | Number |
| Tables should have headers |  | BUG | MAJOR | 3 |
| "<th>" tags should have "id" or "scope" attributes |  | BUG | MAJOR | 77 |
| Elements deprecated in HTML5 should not be used | With the advent of HTML5, many old elements were deprecated. To ensure the best user experience, deprecated elements should not be used. This rule checks for the following deprecated elements: Element Remediation Action basefont, big, blink, center, font, marquee, multicol, nobr, spacer, tt use CSS acronym use abbr applet use embed or object bgsound use audio frame, frameset, noframes restructure the page to remove frames isindex use form controls dir use ul hgroup use header or div listing use pre and code nextid use GUIDS noembed use object instead of embed when fallback is necessary plaintext use the "text/plain" MIME type strike use del or s xmp use pre or code, and escape "&lt;" and "&amp;" characters See W3C, Obsolete Features WHATWG, Obsolete Features | BUG | MAJOR | 2 |
| "<strong>" and "<em>" tags should be used |  | BUG | MINOR | 9 |
| "<fieldset>" tags should contain a "<legend>" |  | BUG | MINOR | 3 |
| Image, area and button with image tags should have an "alt" attribute |  | BUG | MINOR | 8 |
| "<table>" tags should have a description |  | BUG | MINOR | 16 |
| String literals should not be duplicated | Duplicated string literals make the process of refactoring error-prone, since you must be sure to update all occurrences. On the other hand, constants can be referenced from many places, but only need to be updated in a single place. Noncompliant Code Example With the default threshold of 3: def run(): prepare("this is a duplicate") # Noncompliant - "this is a duplicate" is duplicated 3 times execute("this is a duplicate") release("this is a duplicate") Compliant Solution ACTION\_1 = "action1" def run(): prepare(ACTION\_1) execute(ACTION\_1) release(ACTION\_1) Exceptions No issue will be raised on: duplicated string in decorators strings with less than 5 characters strings with only letters, numbers and underscores @app.route("/api/users/", methods=['GET', 'POST', 'PUT']) def users(): pass @app.route("/api/projects/", methods=['GET', 'POST', 'PUT']) # Compliant def projects(): pass | CODE\_SMELL | CRITICAL | 11 |
| Wildcard imports should not be used | Importing every public name from a module using a wildcard (from mymodule import \*) is a bad idea because: It could lead to conflicts between names defined locally and the ones imported. It reduces code readability as developers will have a hard time knowing where names come from. It clutters the local namespace, which makes debugging more difficult. Remember that imported names can change when you update your dependencies. A wildcard import which works today might be broken tomorrow. There are two ways to avoid a wildcard import: Replace it with import mymodule and access module members as mymodule.myfunction. If the module name is too long, alias it to a shorter name. Example: import pandas as pd List every imported name. If necessary import statements can be split on multiple lines using parentheses (preferred solution) or backslashes. Noncompliant Code Example from math import \* # Noncompliant def exp(x): pass print(exp(0)) # "None" will be printed Compliant Solution import math def exp(x): pass print(math.exp(0)) # "1.0" will be printed Or from math import exp as m\_exp def exp(x): pass print(m\_exp(0)) # "1.0" will be printed Exceptions No issue will be raised in \_\_init\_\_.py files. Wildcard imports are a common way of populating these modules. No issue will be raised in modules doing only imports. Local modules are sometimes created as a proxy for third-party modules. # file: mylibrary/pyplot.py try: from guiqwt.pyplot import \* # Ok except Exception: from matplotlib.pyplot import \* # Ok Just keep in mind that wildcard imports might still create issues in these cases. It's always better to import only what you need. See Python documentation - The import statement | CODE\_SMELL | CRITICAL | 12 |
| Cognitive Complexity of functions should not be too high | Cognitive Complexity is a measure of how hard the control flow of a function is to understand. Functions with high Cognitive Complexity will be difficult to maintain. See Cognitive Complexity | CODE\_SMELL | CRITICAL | 4 |
| "SystemExit" should be re-raised | SystemExit&nbsp;is raised when sys.exit() is called. This exception is expected to propagate up until the application stops. It is ok to catch it when a clean-up is necessary but it should be raised again immediately. A bare except: statement, i.e. an except without any exception class, is equivalent to except BaseException. Both statements will catch every exception, including SystemExit. It is recommended to catch instead a specific exception. If it is not possible, the exception should be raised again. Note that it is also a good idea to reraise the KeyboardInterrupt exception. This rule raises an issue when a bare except:, an except BaseException or an except SystemExit don't reraise the exception caught. Noncompliant Code Example try: open("foo.txt", "r") except SystemExit: # Noncompliant pass except KeyboardInterrupt: # No issue raised but be careful when you do this pass try: open("bar.txt", "r") except BaseException: # Noncompliant pass except: # Noncompliant pass Compliant Solution try: open("foo.txt", "r") except SystemExit: # clean-up raise except KeyboardInterrupt: # clean-up raise try: open("bar.txt", "r") except BaseException as e: # clean-up raise e except: # Noncompliant # clean-up raise # or use a more specific exception try: open("bar.txt", "r") except FileNotFoundError: # process the exception See PEP 352 - Required Superclass for Exceptions Python Documentation - Built-in exceptions Python Documentation - The try statement MITRE, CWE-391 - Unchecked Error Condition | CODE\_SMELL | CRITICAL | 1 |
| Constants should not be used as conditions | When a constant is used as a condition, either it has no effect on the execution flow and it can be removed, or some code will never be executed and it is a bug. This rule raises an issue when a constant expression is used as a condition in an if, elif, a conditional expression or other boolean expressions. Noncompliant Code Example def func(param = None): param = (1,) if param: # Noncompliant. var is always set to (1,), the first branch of the if will always execute. return sum(param) else: return None var2 = 1 if func else 2 # Noncompliant. "func" will always be equivalent to True. var3 = func and 1 else 2 # Noncompliant. Compliant Solution def func(param = None): if param is None: param = (1,) if param: return sum(param) else: return None var2 = 1 if func() else 2 var3 = func() and 1 else 2 See PEP 285 - Adding a bool type Python documentation - Truth Value Testing | CODE\_SMELL | CRITICAL | 2 |
| Sections of code should not be commented out | Programmers should not comment out code as it bloats programs and reduces readability. Unused code should be deleted and can be retrieved from source control history if required. | CODE\_SMELL | MAJOR | 3 |
| "aria-label" or "aria-labelledby" attributes should be used to differentiate similar elements |  | CODE\_SMELL | MAJOR | 12 |
| Sections of code should not be commented out | Programmers should not comment out code as it bloats programs and reduces readability. Unused code should be deleted and can be retrieved from source control history if required. | CODE\_SMELL | MAJOR | 38 |
| Function names should comply with a naming convention | Shared coding conventions allow teams to collaborate efficiently. This rule checks that all function names match a provided regular expression. Noncompliant Code Example With the default provided regular expression: ^[a-z\_][a-z0-9\_]\*$ def MyFunction(a,b): ... Compliant Solution def my\_function(a,b): ... | CODE\_SMELL | MAJOR | 4 |
| Two branches in a conditional structure should not have exactly the same implementation |  | CODE\_SMELL | MAJOR | 1 |
| Implicit string and byte concatenations should not be confusing | Python concatenates adjacent string or byte literals at compile time. It means that "a" "b" is equivalent to "ab". This is sometimes used to split a long string on multiple lines. However an implicit string concatenation can also be very confusing. In the following contexts it might indicate that a comma was forgotten: when the two strings are on the same line it looks like a badly formatted tuple. Parenthesises are not mandatory to create a tuple, only the comma is. when the strings are in a list, set or tuple. Noncompliant Code Example def func(): return "item1" "item2" # Noncompliant ["1", "2" # Noncompliant "3", "a very very very" # Noncompliant "very very long string", "4"] Compliant Solution def func(): return "item1", "item2" ["1", "2", "3", "a very very very" + "very very long string", "4"] Exceptions No issue will be raised when there is a visible reason for the string concatenation: when the quotes used for both strings are different. This can be used to avoid escaping quotes when the strings or bytes have different prefixes, i.e. "f" for f-strings, "r" for raw, "u" for unicode and no prefix for normal strings. when strings are visibly split to avoid long lines of code. (Example: the first string ends with a space, punctuation or \n). | CODE\_SMELL | MAJOR | 16 |
| Class names should comply with a naming convention | Shared coding conventions allow teams to collaborate effectively. This rule allows to check that all class names match a provided regular expression. The default regular expression is based on PEP-8 standard. It allows "CapWords" convention and "snake\_case" in lowercase. The "snake\_case" convention is accepted by PEP-8 when the class is primarily used as a callable (ex: decorator, context manager, etc...). However the "CapWords" convention is recommended in every case. Noncompliant Code Example With default provided regular expression ^\_?([A-Z\_][a-zA-Z0-9]\*|[a-z\_][a-z0-9\_]\*)$: class myClass: # Noncompliant ... class my\_CONTEXT\_manager: # Noncompliant def \_\_enter\_\_(self): pass def \_\_exit\_\_(self, type, value, traceback): pass Compliant Solution class MyClass: ... class my\_context\_manager: def \_\_enter\_\_(self): pass def \_\_exit\_\_(self, type, value, traceback): pass | CODE\_SMELL | MINOR | 1 |
| Local variable and function parameter names should comply with a naming convention | Shared naming conventions allow teams to collaborate effectively. This rule raises an issue when a local variable or function parameter name does not match the provided regular expression. Exceptions Loop counters are ignored by this rule. for i in range(limit): # Compliant print(i) | CODE\_SMELL | MINOR | 12 |
| Unused local variables should be removed | If a local variable is declared but not used, it is dead code and should be removed. Doing so will improve maintainability because developers will not wonder what the variable is used for. Noncompliant Code Example def hello(name): message = "Hello " + name # Noncompliant print(name) for i in range(10): foo() Compliant Solution def hello(name): message = "Hello " + name print(message) for \_ in range(10): foo() Exceptions \_ as well as tuples will not raise an issue for this rule. The following examples are compliant: for \_ in range(10): do\_something() username, login, password = auth do\_something\_else(username, login) | CODE\_SMELL | MINOR | 3 |

# Security Hotspots

## Security hotspots count by category and priority

|  |  |  |  |
| --- | --- | --- | --- |
| Category / Priority | LOW | MEDIUM | HIGH |
| LDAP Injection | 0 | 0 | 0 |
| Object Injection | 0 | 0 | 0 |
| Server-Side Request Forgery (SSRF) | 0 | 0 | 0 |
| XML External Entity (XXE) | 0 | 0 | 0 |
| Insecure Configuration | 0 | 0 | 0 |
| XPath Injection | 0 | 0 | 0 |
| Authentication | 0 | 0 | 3 |
| Weak Cryptography | 0 | 0 | 0 |
| Denial of Service (DoS) | 0 | 0 | 0 |
| Log Injection | 0 | 0 | 0 |
| Cross-Site Request Forgery (CSRF) | 0 | 0 | 2 |
| Open Redirect | 0 | 0 | 0 |
| SQL Injection | 0 | 0 | 0 |
| Buffer Overflow | 0 | 0 | 0 |
| File Manipulation | 0 | 0 | 0 |
| Code Injection (RCE) | 0 | 0 | 0 |
| Cross-Site Scripting (XSS) | 0 | 0 | 0 |
| Command Injection | 0 | 0 | 0 |
| Path Traversal Injection | 0 | 0 | 0 |
| HTTP Response Splitting | 0 | 0 | 0 |
| Others | 18 | 0 | 0 |

## Security hotspots List

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Category | Name | Priority | Severity | Count |
| Cross-Site Request Forgery (CSRF) | Disabling CSRF protections is security-sensitive | HIGH | CRITICAL | 1 |
| Cross-Site Request Forgery (CSRF) | Allowing both safe and unsafe HTTP methods is security-sensitive | HIGH | MINOR | 1 |
| Authentication | Hard-coded credentials are security-sensitive | HIGH | BLOCKER | 3 |
| Others | Links with "target=\_blank" are security-sensitive | LOW | MINOR | 3 |
| Others | Using clear-text protocols is security-sensitive | LOW | CRITICAL | 6 |
| Others | Using hardcoded IP addresses is security-sensitive | LOW | MINOR | 9 |