Code Review

Learning Objectives

The learning objectives are to

- To define code quality
- To understand Python coding guidelines
- To understand benefits of code review
- To define Static code analysis

Does this code have good quality?

```
#main.py
def f(a):
    return a + 5
```

Pros:

- It is pretty straightforward
- easy to read
- complexity is low

Cons:

- no documentation
- no tests
- inadequate name

Quantifying Code Quality

- How many defects do we have in the code base?
- Do we comply with the coding style?
- How much code is covered by the tests?
- How fast the code is?
- How much of the code do we really use?
- How much of the code base are code lines and how much space documentation occupies?
- How complex is the code? Can a new developer in the team can dive into the project easily?
- How many and how often the tests fail? Does it change overtime?

Improved
Quality
12 lines are
added

```
# main.py
def add_5(a):
    """
    Add 5 to the provided number

    Parameters
    ----------
    a : int

    Returns
    -------
    int
        Number incremented by 5
    """
    return a + 5
```

Guidelines about Indentations


```
# Arguments on first line forbidden when not
using vertical alignment.
foo = long_function_name(var_one, var_two,
    var_three, var_four)
```

```
# Aligned with opening delimiter.
foo = long_function_name(var_one, var_two,
                         var_three, var_four)
```


```
my_list = [
    1, 2, 3,
    4, 5, 6,
    ]
```

```
result = some_function_that_takes_arguments(
    'a', 'b', 'c',
    'd', 'e', 'f'
    )
```


```
my_list = [
    1, 2, 3,
    4, 5, 6,
]
)
```

```
result = some_function_that_takes_arguments(
    'a', 'b', 'c',
    'd', 'e', 'f',
)
```

```
# Wrong:
# operators sit far away from their operands
income = (gross_wages +
          taxable_interest +
          (dividends - qualified_dividends) -
          ira_deduction -
          student_loan_interest)
```

```
# Correct:
# easy to match operators with operands
income = (gross_wages
          + taxable_interest
          + (dividends - qualified_dividends)
          - ira_deduction
          - student_loan_interest)
```

Line Spacing

- Guidelines for Line spacing
  - Two blank lines around top level functions
  - Two blank lines around classes
  - One blank line between functions in a class
  - One blank line between logical groups in a function *(sparingly)*
  - Extra blank lines between groups of groups of related functions *(why are they in the same file?)*

```
import sys,os
```

```
import os
import sys
```

- Imports should usually be on separate lines
- Imports should be grouped in the following order:
  - Standard library imports.
  - Related third party imports.
  - Local application specific imports.
- Wildcard imports (from <module> import *) should be avoided,

  from myclass **import** MyClass

  from foo.bar.yourclass **import** YourClass

Whitespace Guidelines

– No trailing spaces at end of a line
– Do not pad ([{ with spaces,
– Do not pad before : ; , , ,

```
78  spam(ham[1], {eggs: 2})
79  spam( ham[ 1 ], { eggs: 2 } )
```

```
82  if x == 4: print x, y; x, y = y, x
83  if x == 4 : print x , y ; x , y = y , x
```

Whitespace Guidelines

Always surround =, +=, -=, == , < , > , != , <> , <= , >= , in , not in , is , is not, and, or, not with a single space

```
86   i = i + 1
87   submitted += 1
88   x = x*2 - 1
89   hypot2 = x*x + y*y
90   c = (a+b) * (a-b)
```

```
93   i=i+1
94   submitted +=1
95   x = x * 2 - 1
96   hypot2 = x * x + y * y
97   c = (a + b) * (a - b)
```

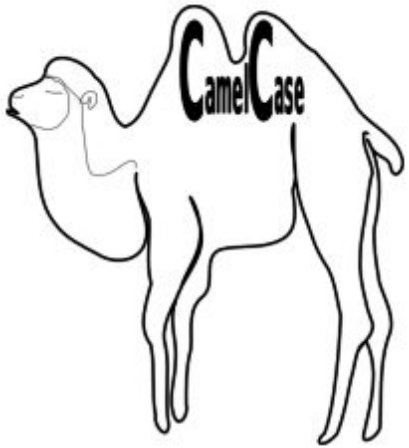
Whitespace Guidelines

Never surround = with a space as a function parameter argument

```
101  def complex(real, imag=0.0):
102      return magic(r=real, i=imag)
103
104
105  def complex(real, imag = 0.0):
106      return magic(r = real, i = imag)
```

Naming Conventions

- How you name functions, classes, and variables can have a huge impact on **readability**
- Avoid the following ***variable names***:
  - Lower case L (l)
  - Uppercase O (O)
  - Uppercase I (I)
- **Class names** should be in CapWords Also known as CamelCase
- **Functions (e.g. generate_otp())**
  - Lowercase, with words separated by underscores as necessary to improve readability
  - Function Names are usually verbs
- To indicate a variable is a constant, use all CAPS, (e.g.. SENDER_MAIL_ID)



Bad Example

```
1  from statistics import mean
2  import numpy.random as nprnd
3  from statistics import stdev
4  def MyFuNcTiOn(ARGUMENT):
5      m = mean(ARGUMENT)
6      s = stdev(ARGUMENT)
7      gt3sd = 0
8      lt3sd = 0
9      for m in ARGUMENT:
10         if m > m + (s * 2):
11             gt3sd += 1
12         elif m < m - (s * 2):
13             lt3sd += 1
14     return(gt3sd,lt3sd)
15 def AnotherFunction(anumber, anothernumber):
16     l = nprnd.randint(anothernumber, size = anumber)
17     return(MyFuNcTiOn(l))
18 a,b=AnotherFunction(anumber = 1000, anothernumber = 1000)
19 print('found %d random values greather than 2 * sd and %d less than 2 * sd' % (a, b))
```

Documentations

Code need to be documented for two things

- Code readers:What the code is doing and why

  Code comments

- Users: How to use your code

  README.md

Code Comments

Here are some key points to remember when adding comments to your code:

- Limit the line length of comments and docstrings to 72 characters.
- Use complete sentences, starting with a capital letter.
- Make sure to update comments if you change your code.
- Code comments are of three types: block comments, inline comments and

Code Comments

Here are some key points to remember when adding comments to your code:

- Limit the line length of comments and docstrings to 72 characters.
- Use complete sentences, starting with a capital letter.
- Make sure to update comments if you change your code.

Block Comments

- Indent block comments to the same level as the code they describe.
- Start each line with a # followed by a single space.
- Separate paragraphs by a line containing a single #.

Python

```
for i in range(0, 10):
    # Loop over i ten times and print out the value of i, followed by a
    # new line character
    print(i, '\n')
```

Inline Comments

- Use inline comments sparingly.
- Write inline comments on the same line as the statement they refer to.
- Separate inline comments by two or more spaces from the statement.
- Start inline comments with a # and a single space, like block comments.
- Don’t use them to explain the obvious.

```python
x = 5  # This is an inline comment
```

Docstrings

- Surround docstrings with three double quotes on either side, as in """This is a docstring""".
- Write them for all public modules, functions, classes, and methods.
- Put the """ that ends a multi line docstring on a line by itself:

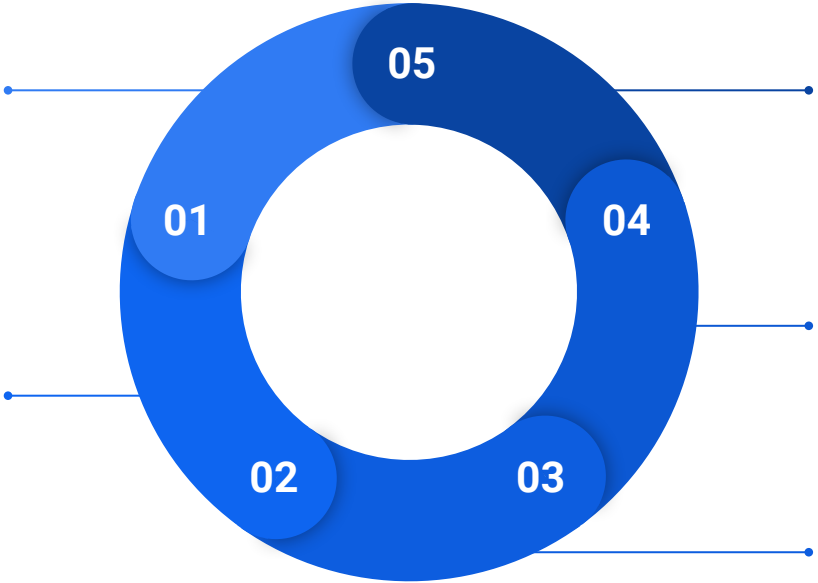
```
def quadratic(a, b, c, x):
    """Solve quadratic equation via the quadratic formula.

    A quadratic equation has the following form:
    ax**2 + bx + c = 0

    There always two solutions to a quadratic equation: x_1 & x_2.
    """
    x_1 = (- b+(b**2-4*a*c)**(1/2)) / (2*a)
    x_2 = (- b-(b**2-4*a*c)**(1/2)) / (2*a)

    return x_1, x_2
```

Code Reviews
Part II

Why do programmers do code reviews

Finding Defects.

Code Improvement.



Alternative Solutions

Knowledge Transfer

Share Code Ownership

Code Review Best practices

Set Clear Objectives	Use Checklist	Review in small chunks	Understand the Context	Ensure Standards
1. Finding Defects 2. Improving Code Readability 3. Knowledge Sharing	Create and use code review checklists that outline common issues to look for	Break down code changes into smaller, manageable piece	Understanding the problem the code is solving, the design decisions, and the impact on the overall system.	Check code adheres to coding standards and guidelines established for the project

Practice Problem

```python
def calculate_sum(n, m):
    """
    Calculates the sum of all integers between n and m, inclusive.
    """
    if n > m:
        return 0
    else:
        total = 0
        for i in range(n, m + 1):
            total = total + i
        return total
```

Practice Problem

Function Docstring:

- **Issue:** The docstring mentions what the function does, but it doesn't specify the input parameters or the expected return value.
- **Recommendation:** Improve the docstring to include information about the function's parameters and the return value.

Conditional Statement:

- **Issue:** The if-else statement is not necessary; the else branch is redundant.
- **Recommendation:** Simplify the code by removing the unnecessary else branch.

Loop:

- **Issue:** The loop can be simplified by using the arithmetic sum formula.
- **Recommendation:** Replace the loop with a direct calculation using the formula: (m - n + 1) * (n + m) / 2.

Practice Problem

```python
def calculate_sum(n, m):
    """
    Calculates the sum of all integers between n and m, inclusive.

    Args:
        n (int): The starting integer.
        m (int): The ending integer.

    Returns:
        int: The sum of integers from n to m.
    """
    if n > m:
        return 0
    return (m - n + 1) * (n + m) // 2
```

Code Reviews
Checklist

General Code Quality

Is the code readable and well-documented? Are variable and function names clear and meaningful?	✔	X
Does the code follow the project's coding standards and style guidelines?	✔	X
Are there any code smells or anti-patterns in the code?	✔	X
Are there any unnecessary or commented-out code sections?	✔	X
Are there any magic numbers or hard-coded values that should be replaced with constants or configuration settings?	✔	X

Functionality and Logic

Does the code solve the intended problem or implement the required feature?	✔	X
Are edge cases and error conditions handled properly?	✔	X
Are there any missing validations or input checks?	✔	X
Does the code handle exceptions and errors gracefully, with meaningful error messages?	✔	X
Are there any redundant or duplicated code blocks that can be refactored?	✔	X

Testing and Documentation

Are there unit tests, and do they cover different code paths and scenarios?	✔	X
Are test cases included for the code changes?	✔	X
Do the tests pass, and do they fail when expected?	✔	X
Is the code adequately documented, including comments, function descriptions, and API documentation?	✔	X
Are there updates in the project's documentation, README files, or changelogs to reflect the code changes?	✔	X

Static Code Analysis

What is SCA

A method of reviewing and analyzing source code, bytecode, or binary code without executing it.

Why

To identify potential issues, vulnerabilities, and defects in software before it is run or deployed.

How

Through specialized tools or software that scan the code for a wide range of problems

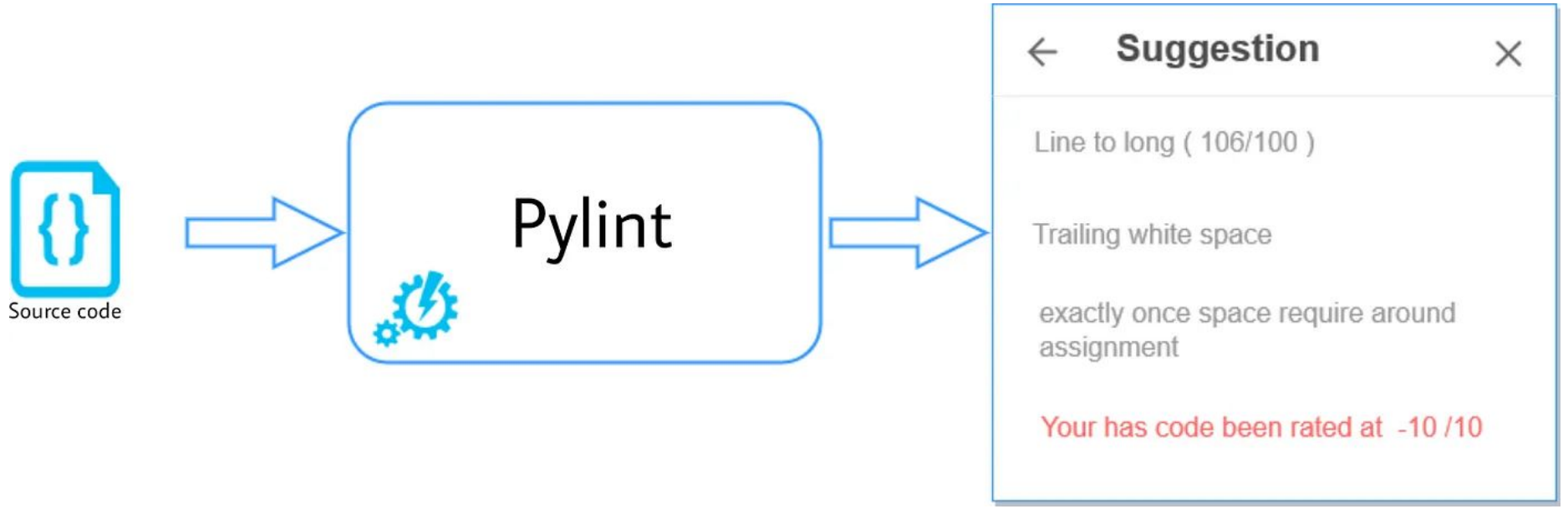
Key Aspects of SCA

Automated Process	Detection of Issues	Consistency	Integration	Customization
Static code analysis is an automated process that examines the code for issues by parsing and analyzing it	coding standards violations, potential security vulnerabilities code smells, memory leaks, and logic	enforce coding standards and best practices, promoting code consistency	**integrate with development environments, build systems, and CI/CD pipelines.**	Tools often allow for customization of analysis rules and settings to match the specific requirements and coding standards of a project

Comparison of various SCA tools

ESLint	JavaScript, TypeScript	Enforces coding standards and style guidelines.
Pylint	Python	Analyzes Python code for style and conventions.
Checkmarx	Java, C/C++, Python	Focuses on identifying security vulnerabilities and compliance issues.
SonarQube	Multiple languages (e.g., Java, Python, C++)	Detects code smells, bugs, vulnerabilities, and provides a holistic view of code quality and maintainability.

Improve Python code with Pylint



Quiz time

1. Which of the following THREE parameters are typically used to measure Code quality
   a. Number of defects in the program
   b. Algorithmic complexity of a function
   c. Compliance to industry guidelines
   d. Code coverage of the test cases
   e. Time taken to execute the program

Quiz time

1. Which of the following is NOT one of the best practices for code review

   a. Use camel case for declaring class names
   b. Set clear objectives
   c. Use check list
   d. Review in small chunks

Quiz time

1. Which of the following is static code analyzer checking Python program for the compliance of coding guidelines
   a. PyCharm
   b. PySCA
   c. Pylint
   d. PyCheck

Quiz time

1. What is the problem with following code segment

```
# Arguments on first line forbidden when not
using vertical alignment.
foo = long_function_name(var_one, var_two,
    var_three, var_four)
```