Programação e Algoritmia
--x--
Programming and Algorithms

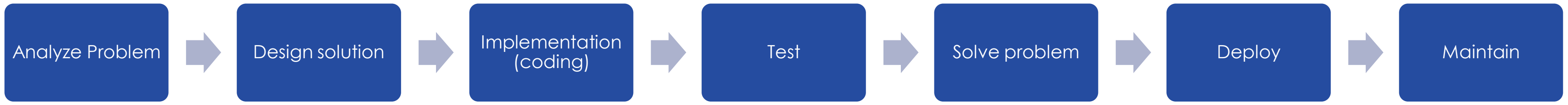
1 – Object-Oriented Programming

Object Oriented Development (in Python)

Object Oriented Analysis and Design (OOAD)

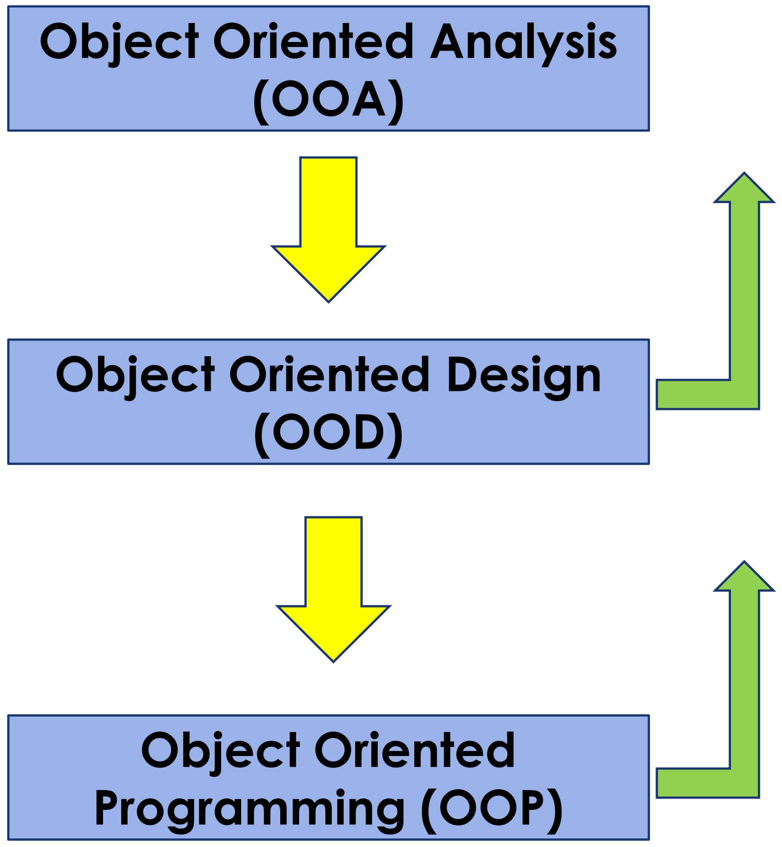
- How to create the code (i.e., the program instructions) for your programs?
- Perhaps, like many programmers, you'll simply turn on your computer and start typing.
- This approach may work for small programs, but what if you were asked to create a software system to control thousands of automated teller machines for a major bank?
  - Or suppose you were asked to work on a team of 1,000 software developers building the next generation of air traffic control systems?
- For projects so large and complex, **you should not simply sit down and start writing programs.**

The complete cycle



Object Oriented Software Engineering

- In a software project adopting an object-oriented approach:
- 1st is performed an analysis (object-oriented)
- 2nd , the design (object-oriented)
- 3rd , implementation of the solution (design model) by development of a set of programs (source code)
- …


Object Oriented Analysis (OOA)
Object Oriented Design (OOD)
Object Oriented Programming (OOP)

Object Oriented analysis-and-design

- To create the best solutions, you should follow:
  - a detailed analysis process for determining your project's requirements

    i.e., defining what the system is supposed to do
  - then develop a design that satisfies them

    i.e., specifying how the system should do it

- Ideally, you'd go through this process and carefully review the design (and have your design reviewed by other software professionals) before writing any code.

- If this process involves analyzing and designing your system from an object-oriented point of view, it's called an object-oriented analysis-and-design (OOAD) process.

Object Oriented Analysis

- The analysis process aims the identification and representation of objects, classes, operations and relations among the domain objects
- In this phase, several questions must be made:
  - Objects needed?
  - Attributes of an objects?
  - Operations of an object?
  - What type of information is produced, consumed or used by an object?
  - Type of relations among objects of the domain?

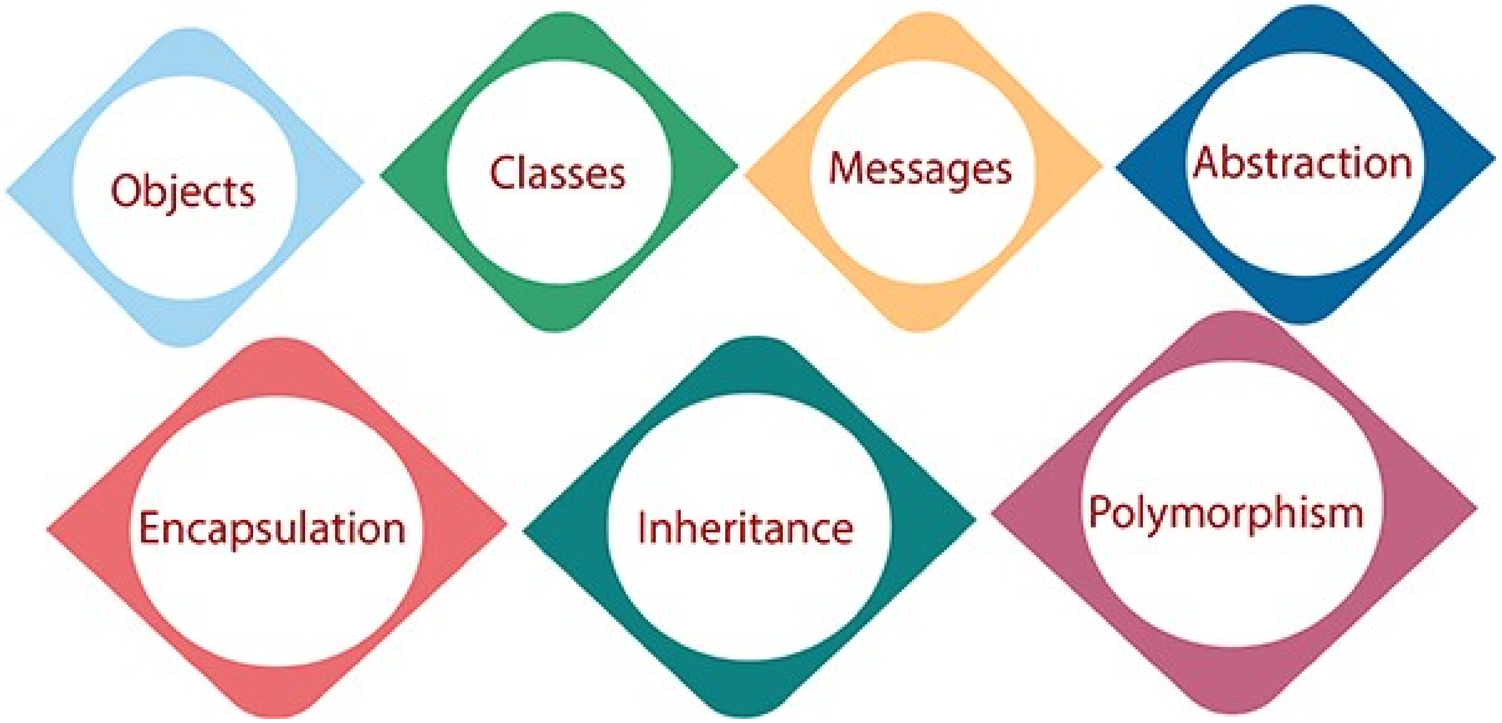
OO Analysis

- Given the description of a problem (associated to a domain), OO analysis involves the following tasks:
  - Identification of the **classes** that represent the domain in analysis
  - Definition of the **attributes** for each class
  - Identification of the **relations** (inheritance, association, aggregation, composition) among classes
  - Identification of the **operations** that define the behavior of each class

Object Oriented Sofware Design (1)

- In the object-oriented design method, the system is viewed as a collection of objects (i.e., entities).
- The state is distributed among the objects, and each object handles its state data.
- Objects have their internal data which represent their state.
- Similar objects create a class.
  - In other words, each object is a member of some class.
- Classes may inherit features from the superclass.
- Has as basis and main approach modeling as close as possible the real word

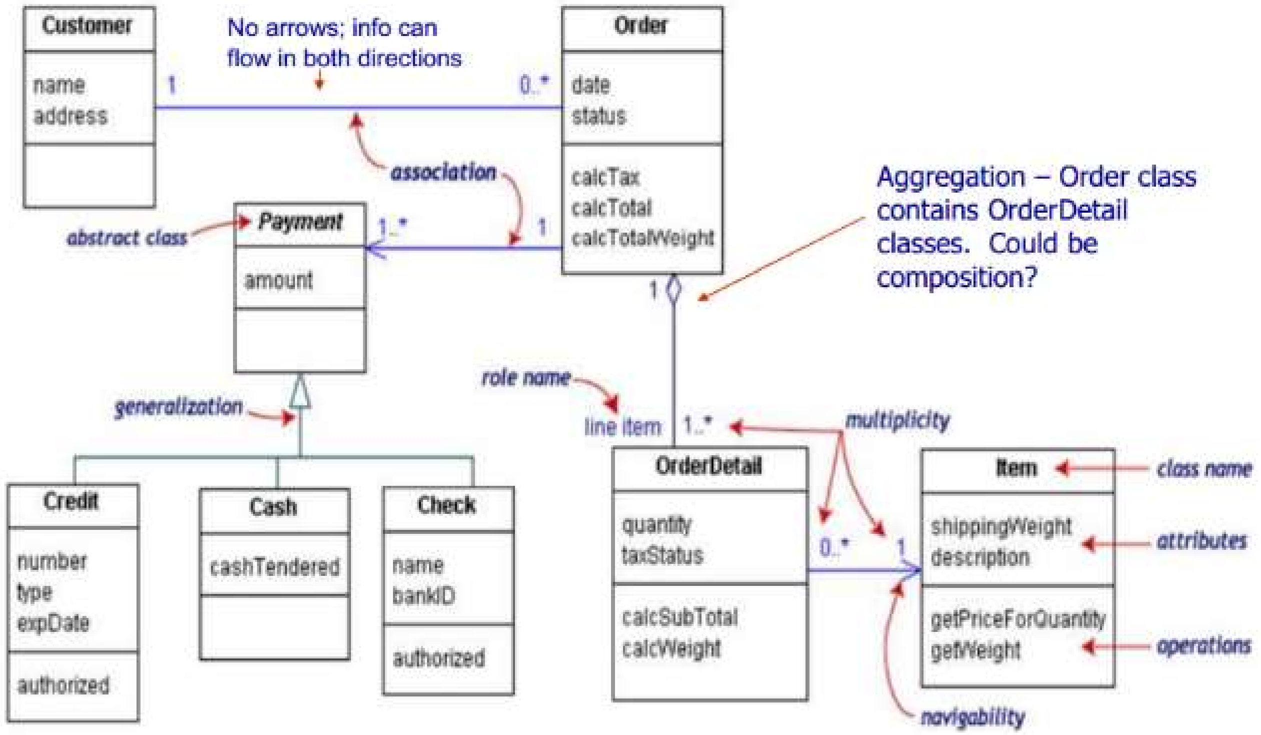
Object Oriented Sofware Design (2)



Object Oriented Sofware Design (3)

- The result of analysis and design can be (and often is) represented using UML

- Example:



Structured Programming Paradigm

- It is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program
  - by making extensive use of the structured control flow constructs of selection (if/then/else) and repetition (while and for), block structures, and subroutines.
- It emerged in the late 1950s

- The main domain of the problem is divided into subproblems
- Use of functions and procedures to solve each of the subproblems
- Use of variables (data structures) to store data

- Use of parameters to pass data to functions and procedures

Object Oriented Programming (1)

- Incorporates and extends structured programming

- OOP integrates modules, functions, procedures and data structures and data into a unique design and programming unit, the class

- Many of the most widely used programming languages are multi-paradigm and they support object-oriented programming to a greater or lesser degree
  - Python, C++, Java, …
- Programming adopting this paradigm is called object-oriented programming (OOP)
  - It allows you to implement an object-oriented design as a working system

Style

Style Guides

- Publishers, journals, and institutions often have style guides designed to instil a certain uniformity in the use of English (or other human languages).
- Similarly, style guides exist for programming languages.

Style Guide for Python Code (PEP 8)

- The Python community has an essentially unified and very widely followed set of conventions.

- These are codified in one of the Python standards documents, PEP 8 1.
  - PEP 8 – Style Guide for Python Code | peps.python.org

- PEP 8 isn't all that long, and it is worth taking the time to read.

- Some highlights follow…

Code layout

- One of the most important factors in making code readable is the space, or lack of it, between and around the text which makes up the code.

- Whitespace affects readability in many ways.
  - Too much code bunched together makes it hard for the eye to separate program statements, while leaving too much space limits the amount of code which fits in the editor window at once.
- Whitespace can also convey information by grouping together concepts which are related and separating distinct ideas.
  - This gives the reader visual clues which can aid in understanding the code.
- With this in mind, PEP 8 defines rules around white space and code formatting.

PEP 8 - Blank lines

- Classes and functions defined at the top level of a module (i.e. not nested in other classes or functions) have two blank lines before and after them.
- Methods within a class are separated by a single blank line.
- Statements within functions usually follow on the immediate next line, except that logical groups of statements, can be separated by single blank lines.
  - Think of each statement as a sentence following on from the previous, with blank lines used to divide the function into short paragraphs.

PEP 8 - White space within line

- Don’t put a space after an opening bracket (of any shape), or before a closing bracket.
- Don’t put a space between the function name and the opening round bracket of a function call, or between a variable name and the opening square bracket of an index.
- Put a space after a comma but not before it
  - exactly like you would in writing prose.
- Put exactly one space on each side of an assignment (=) and an augmented assignment (+=, -=, etc.)
- Do not put a space either before or after the equals sign of a keyword argument.
- Put exactly one space before and after the lowest priority mathematical operators in an expression.
- Never, ever have blank spaces at the end of a line, even a blank line.

```
( 1, 2) # Space after opening bracket.
(1, 2 ) # Space after closing bracket.
```
✗

```
(1, 2) # No space between brackets and contents.
```
✓

```
sin (1) # Space between function name and bracket.
x [0] # Space before index square bracket.
```
✗

```
sin(1)
x[0]
```
✓

```
(1,2,3) # Spaces missing after commas.
(1 ,2 ,3) # Spurious spaces before commas.
(1, ) # Space before closing bracket.
```
✗

```
(1, 2, 3) # Spaces after commas.
(1,) # No space before closing bracket.
```
✓

```
x=1 # Missing spaces around equals sign.
x+=1 # Missing spaces around augmented addition operator.

frog = 2
cat  = 3 # Additional space before equals sign.
```
✗

```
x = 1
x += 1

frog = 2
cat = 3
```
✓

```
myfunction(arg1 = val1, arg2 = val2)
```
✗

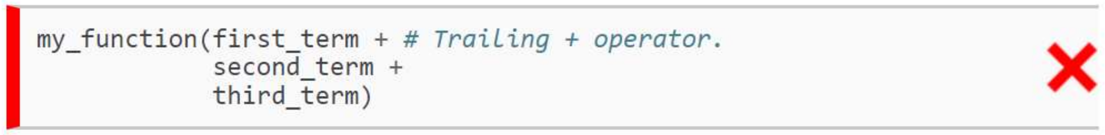
```
myfunction(arg1=val1, arg2=val2)
```
✓

```
y = 3*x**2+4*x+5 # No spaces around +
```
✗

```
y = 3*x**2 + 4*x + 5
```
✓

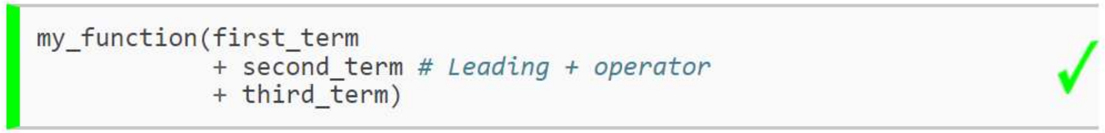
PEP 8 - Line breaks

- Have no lines longer than 79 characters.
  - Limiting the line length makes lines easier to read and prevents the editor from automatically wrapping the line in harder to read ways.
- When breaking lines to fit under 79 characters, it's better to break the lines using the implied continuation within round, square or curly brackets than explicitly with a backslash
- When a mathematical operator occurs at a line break, always put the operator first on the next line, and not last on the first line.



```
my_function(first_term + # Trailing + operator.
            second_term +
            third_term)
```

✗



```
my_function(first_term
            + second_term # Leading + operator
            + third_term)
```

✓

PEP 8 - Name conventions (1)

- Class names
  - use the CapWords convention: each word in a name is capitalised and words are concatenated, without underscores between.

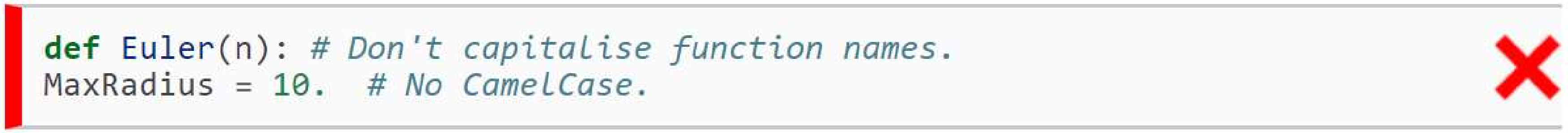
```
complex_polynomial # No capitals, underscore between words.
complexPolynomial # Missing leading capital.
Complex_Polynomial # Underscore between words.
```

```
ComplexPolynomial
```

- Exception names
  - Exceptions are classes, so the rules for class names apply with the addition that exceptions that designate errors should end in Error.
  - Ex: PolynomialDivisionError # OK

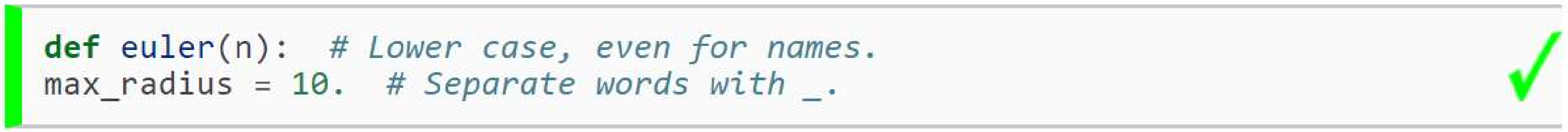
PEP 8 - Name conventions (2)

- function, variable, and module names
  - Almost all names other than classes are usually written in all lower case, with underscores separating words.
  - Even proper nouns are usually spelt with lower case letters to avoid being confused with class names.



```
def Euler(n): # Don't capitalise function names.
MaxRadius = 10.   # No CamelCase.
```

✗

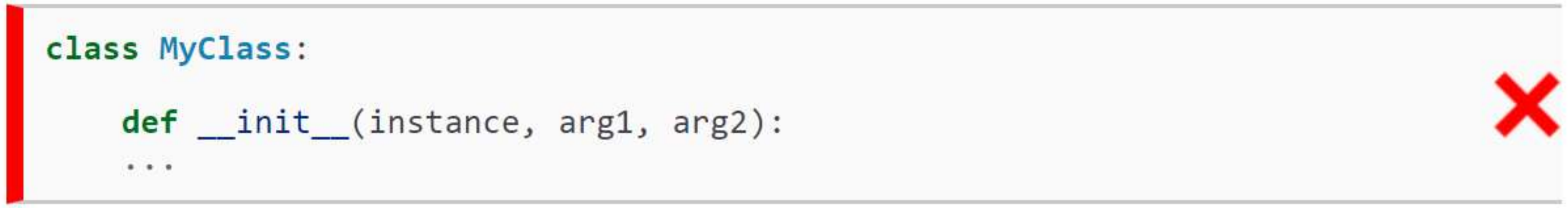


```
def euler(n):  # Lower case, even for names.
max_radius = 10.  # Separate words with _.
```

✓

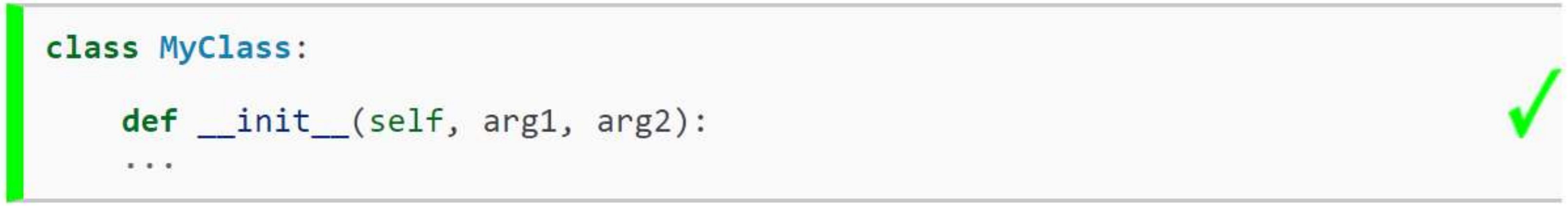
PEP 8 - Name conventions (3)

- method parameters
  - The first parameter to an instance method is the class itself.
  - Always and without exception name this parameter self.



```
class MyClass:

    def __init__(instance, arg1, arg2):
    ...
```

✗

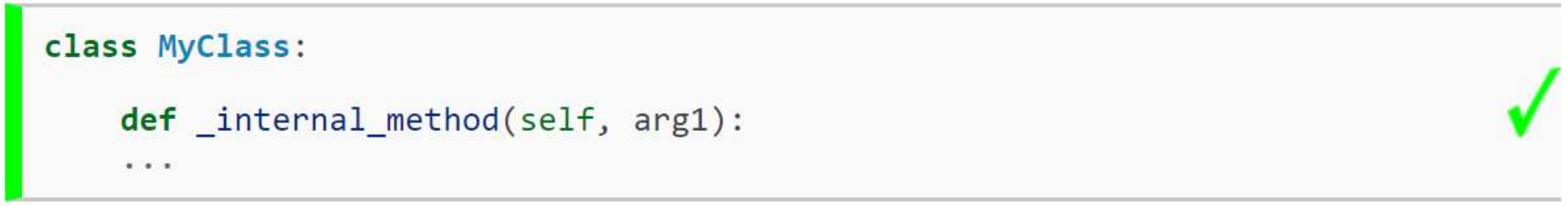


```
class MyClass:

    def __init__(self, arg1, arg2):
    ...
```

✓

PEP 8 - Name conventions (4)

- non-public methods and attributes
  - If a method or attribute is not intended to be directly accessed from outside the class, it should have a name starting with an underscore.
  - This provides a clear distinction between the public interface of a class and its internal implementation.



```
class MyClass:

    def _internal_method(self, arg1):
        ...
```

✓

Choosing names

- Short names help make short lines of code
  - which in turn makes it easier to read and understand what the code does to the values it is operating on.
- However short names can also be cryptic, making it difficult to establish what the names mean.
- This creates a tension: should names be short to create readable code, or long and descriptive to clarify their meaning?
- A good answer to this dilemma is that local variables should have short names.
  - These are often the most frequently occurring variables on a line of code, which makes the statement more intelligible.
  - Should a reader be unclear what a variable stands for, the definition of a local variable will not be very far away.
- Conversely, a module, class, or function which might be used far from its definition had better have a descriptive name which makes its purpose immediately apparent.

More information

- Sections 4.2, 4.3 and 4.4 of the book **Object-oriented Programming in Python for Mathematicians**
  - by David A. Ham
  - Online at https://object-oriented-python.github.io/4_style.html#pep-8

Documenting code

Why comment the code ?

1. If you see/edit code later, comments may help you to remember your logic that you have written while writing that code.
   - Sometimes, it happens with lazy programmers (who do not comment the code properly) that they forget their implemented logics and waste much more time solving the issue.
2. Well commented functions/logics are helpful to other programmers to understand the code better.
   - Essential when development is made by a team

Recommendation: please comment the code properly so that you or your colleagues can understand the logic better.

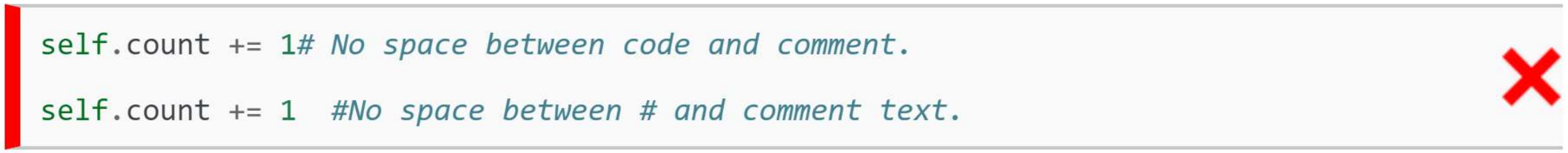
- Writing comments may take time, but it maintains the international coding standards.

Comments

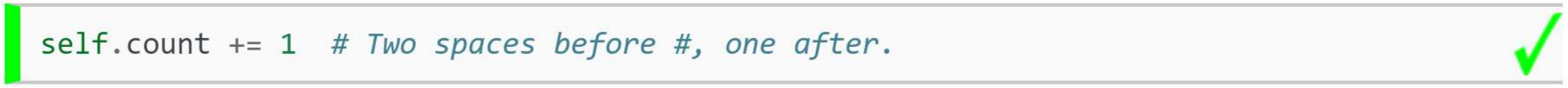
- Comments are non-code text included in programs to help explain what they do.
- Comments exist to aid understanding programs
- While judiciously deployed comments can be an essential aid to understanding, too many comments can be worse than too few.
  - If the code is simple, elegant, and closely follows how a reader would expect the algorithm to be written, then it will be readily understood without comments.
  - Conversely, attempting to rescue obscure, badly thought-through code by writing about it is unlikely to remedy the situation.

PEP 8 rules for comments

- Comments start with a single # followed by a single space.
- Inline comments are separated from the code by at least two spaces.



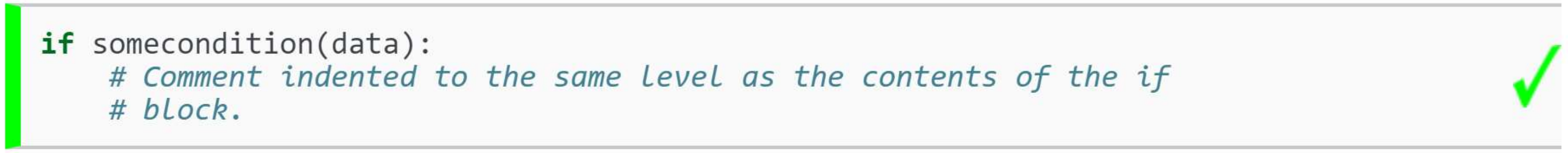
```python
self.count += 1# No space between code and comment.

self.count += 1  #No space between # and comment text.
```



```python
self.count += 1  # Two spaces before #, one after.
```

PEP 8 rules for comments (2)

- Each line of a block comment starts with a single # indented to the same level as a normal line of code.
- The # is followed by a single space, unless a particular piece of comment should be indented with respect to the paragraph it is in, in which case additional spaces are allowed.



```
if somecondition(data):
    # Comment indented to the same level as the contents of the if
    # block.
```

✓

Docstrings (“documentation strings”)

- Docstrings are comments at the start of modules, classes, and functions which describe public interfaces.
- The entire point of a public interface is that the programmer using it should not have to concern themselves with how it is implemented.
  - They should, therefore, not need to read the code in order to understand how to use it.
- The Python interpreter has special support for docstrings.
  - When a user calls help() on an object (including a function or method) then any docstring on that object is used as the body of the resulting help message.
- Docstrings are also understood by the Python documentation generation system, Sphinx.
  - This enables documentation webpages to be automatically generated from Python code.

Where to use docstrings

- Every **public** module, class, function, and method should have a docstring.

  – "Public" in this context means any code which is intended to be accessed from outside the module in which it is defined.

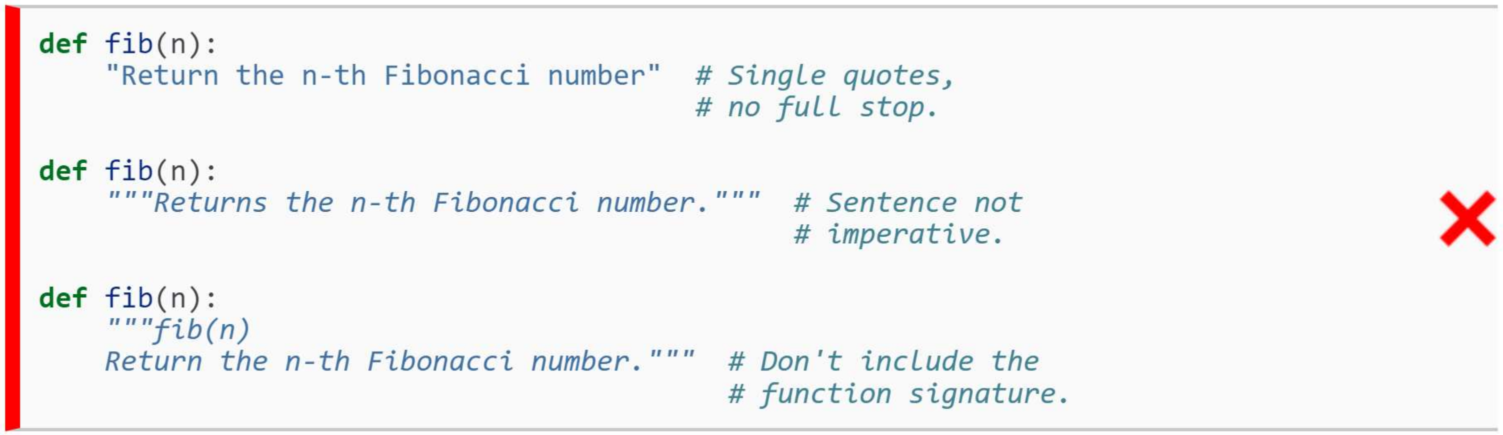
Docstring conventions

- Python itself doesn’t know anything about docstring contents, it will simply display the docstring when you ask for help.
- However, other tools such as those that generate websites from documentation depend on you following the conventions.

- By convention, docstrings are delimited by three double quote characters (""").

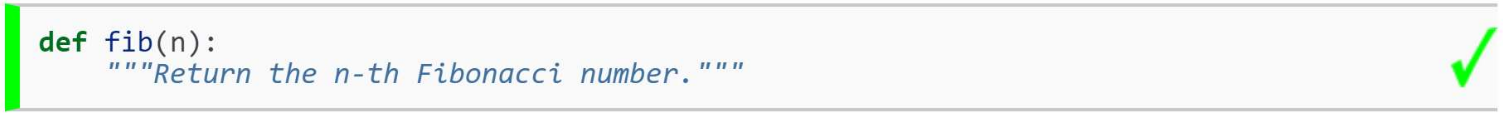
Docstring conventions (2)

- **Simple functions** which take one or two arguments can be documented with a single line docstring which simply says what the function does.
  - The single line should be an imperative sentence and end with a full stop.



```python
def fib(n):
    "Return the n-th Fibonacci number"  # Single quotes,
                                        # no full stop.

def fib(n):
    """Returns the n-th Fibonacci number."""  # Sentence not
                                              # imperative.

def fib(n):
    """fib(n)
    Return the n-th Fibonacci number."""  # Don't include the
                                          # function signature.
```

✗



```python
def fib(n):
    """Return the n-th Fibonacci number."""
```

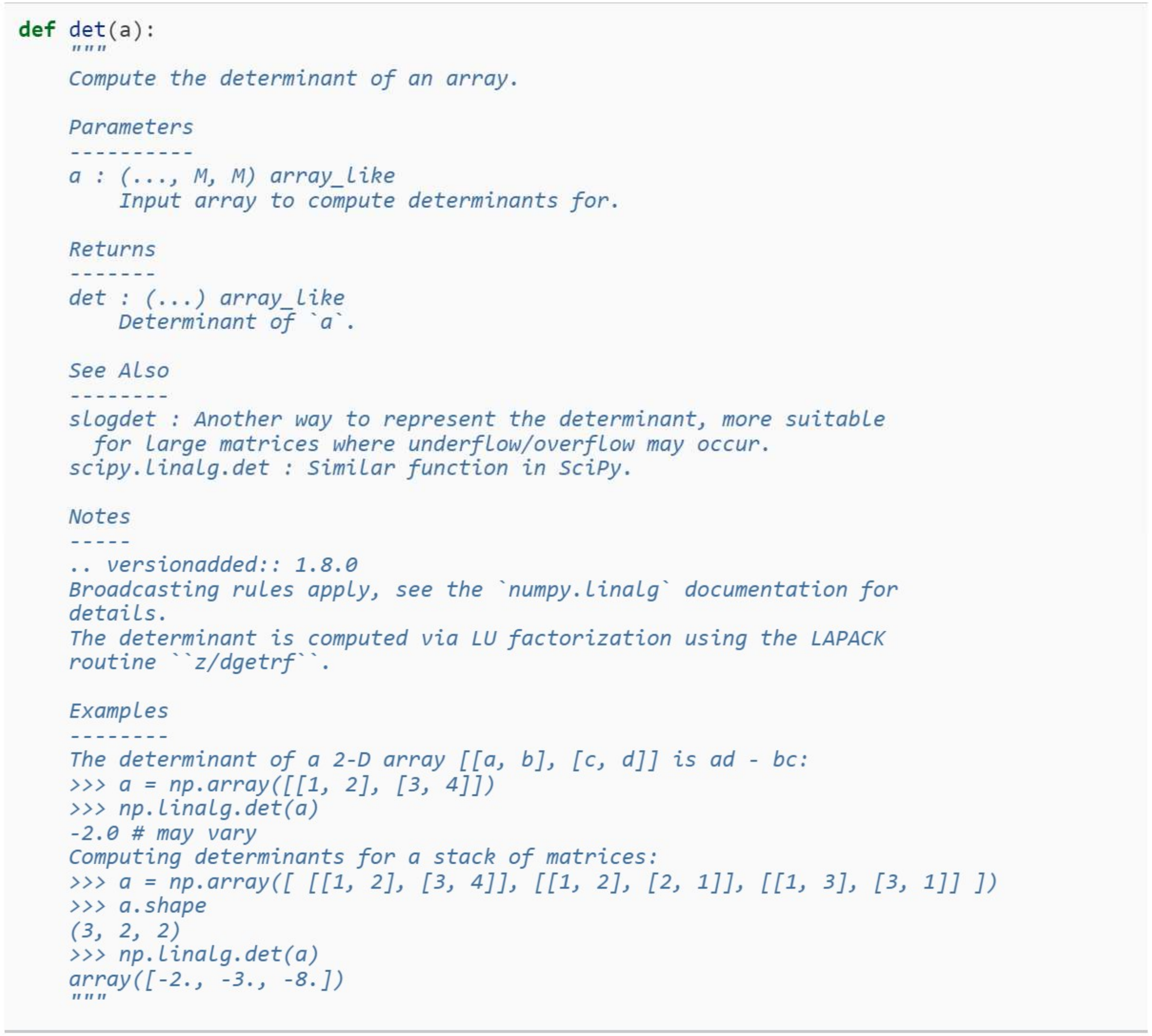
✓

Docstring conventions (3)

- A more complex object will require much more information in its docstring.
  - This covers the type and shape of the input parameter and return value, references to other implementations, and examples of usage.
- There is no single universal standard for the layout of a long docstring, but there are two project or institution-based conventions that are recognised by the web documentation system.
  - One from Google and the other from the Numpy project.
- You should consistently use one of these styles across a whole project.
- Clearly if you are contributing code to an existing project then you should follow their style.

Example - docstring for numpy.linalg.det()

```python
import numpy

help(numpy.linalg.det)
```



```python
def det(a):
    """
    Compute the determinant of an array.

    Parameters
    ----------
    a : (..., M, M) array_like
        Input array to compute determinants for.

    Returns
    -------
    det : (...) array_like
        Determinant of `a`.

    See Also
    --------
    slogdet : Another way to represent the determinant, more suitable
      for large matrices where underflow/overflow may occur.
    scipy.linalg.det : Similar function in SciPy.

    Notes
    -----
    .. versionadded:: 1.8.0
    Broadcasting rules apply, see the `numpy.linalg` documentation for
    details.
    The determinant is computed via LU factorization using the LAPACK
    routine ``z/dgetrf``.

    Examples
    --------
    The determinant of a 2-D array [[a, b], [c, d]] is ad - bc:
    >>> a = np.array([[1, 2], [3, 4]])
    >>> np.linalg.det(a)
    -2.0 # may vary
    Computing determinants for a stack of matrices:
    >>> a = np.array([ [[1, 2], [3, 4]], [[1, 2], [2, 1]], [[1, 3], [3, 1]] ])
    >>> a.shape
    (3, 2, 2)
    >>> np.linalg.det(a)
    array([-2., -3., -8.])
    """
```

More information

- Sections 4.6 and 4.7 of the book **Object-oriented Programming in Python for Mathematicians**
  - by David A. Ham
  - Online at https://object-oriented-python.github.io/4_style.html#comments