
python
LIST
COMPREHENSIONS

BENEFITS:

Create a new list based on the values of an existing list..

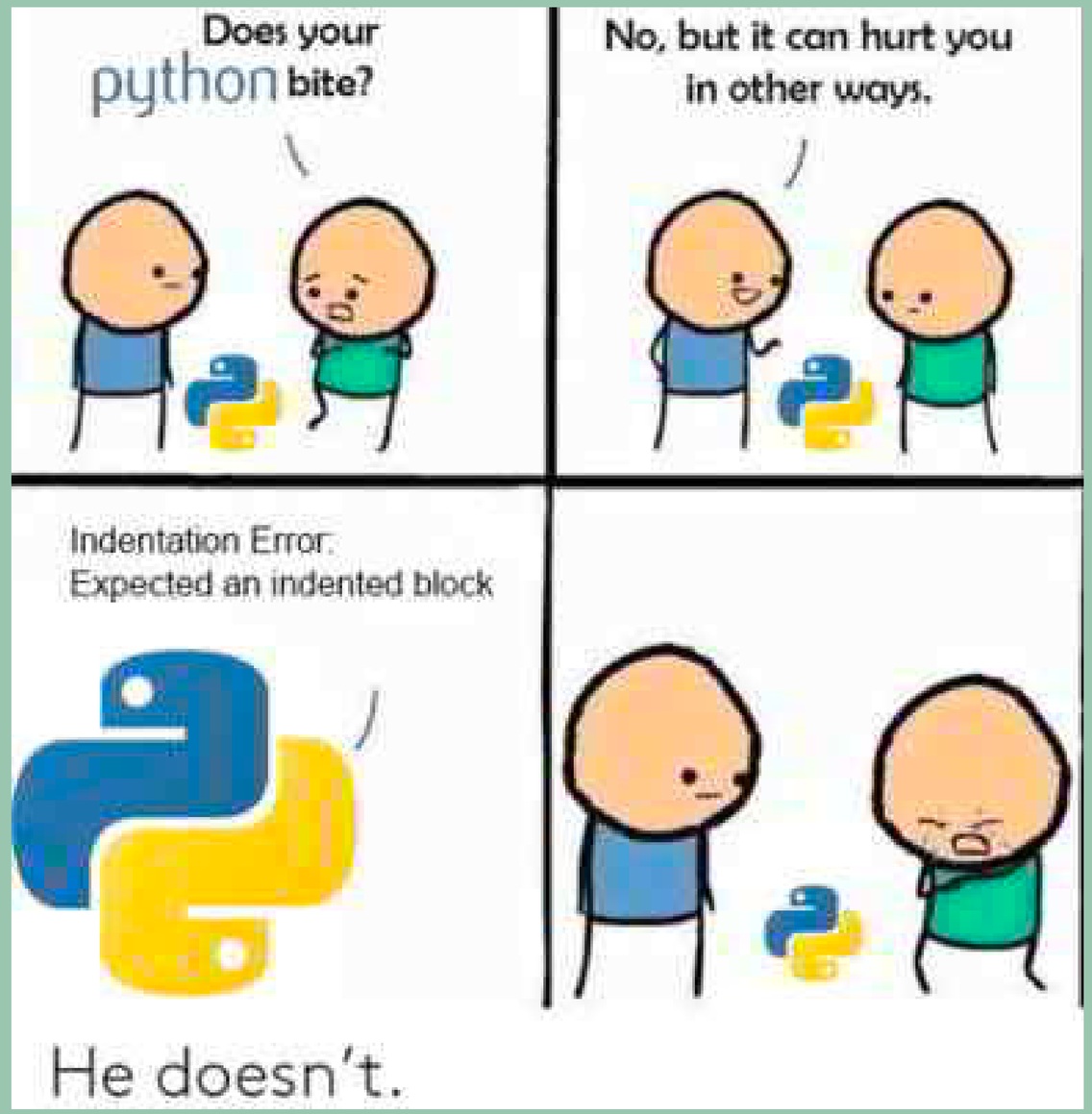
- Clean, elegant way of writing a for loop.
- **Declarative** – tell it what to do, instead of how to do it! (*Imperative)*
- Single tool that can be used in many situations: i.e. mapping, filtering.
- One-liner!

SYNTAX:

```
def multiply_by_two(numbers):
    results = []
    for num in numbers:
        results.append(num * 2)
    return results
```

```
def multiply_by_two(numbers):
    return [num * 2 for num in numbers]
```

Can transform any data type, and from one data type to another.



WRITE PYTHONIC CODE.

WRITE PYTHONIC CODE.

PEP 8 — the Style Guide for Python Code

This stylized presentation of the well-established PEP 8 was created by Kenneth Reitz (for humans).

Introduction

This document gives coding conventions for the Python code comprising the standard library in the main Python distribution. Please see the companion informational PEP describing style guidelines for the C code in the C implementation of Python [1].

This document and PEP 257 (Docstring Conventions) were adapted from Guido's original Python Style Guide essay, with some additions from Barry's style guide [2].

This style guide evolves over time as additional conventions are identified and past conventions are rendered obsolete by changes in the language itself.

Many projects have their own coding style guidelines. In the event of any conflicts, such project-specific guides take precedence for that project.

A Foolish Consistency is the Hobgoblin of Little Minds

One of Guido's key insights is that code is read much more often than it is written. The guidelines provided here are intended to improve the readability of code and make it


python
BUBBLE SORT
& SWAPPING
Algorithms!

BUBBLE SORT

An algorithm for ordering a list by using a swapping function.

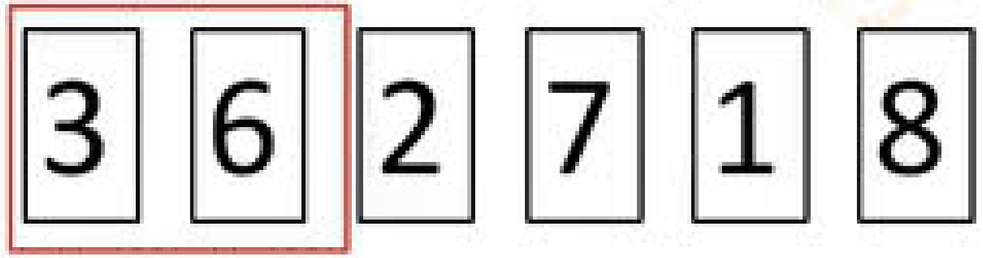
- Time complexity: O(n2)
- Space complexity: O(1)

CONSIDERED THE SIMPLEST SORTING ALGO.

Python swapping makes this cool.

HOW IT WORKS:

Bubble Sort:



BUBBLE SORT:

WRITE PYTHONIC CODE.

Swap two numbers:

```
# Python program to swap two variables

x = 5
y = 10
```

```
# create a temporary variable and swap the values
temp = x
x = y
y = temp
```

WRITE PYTHONIC CODE.

Swap two numbers, pythonically:

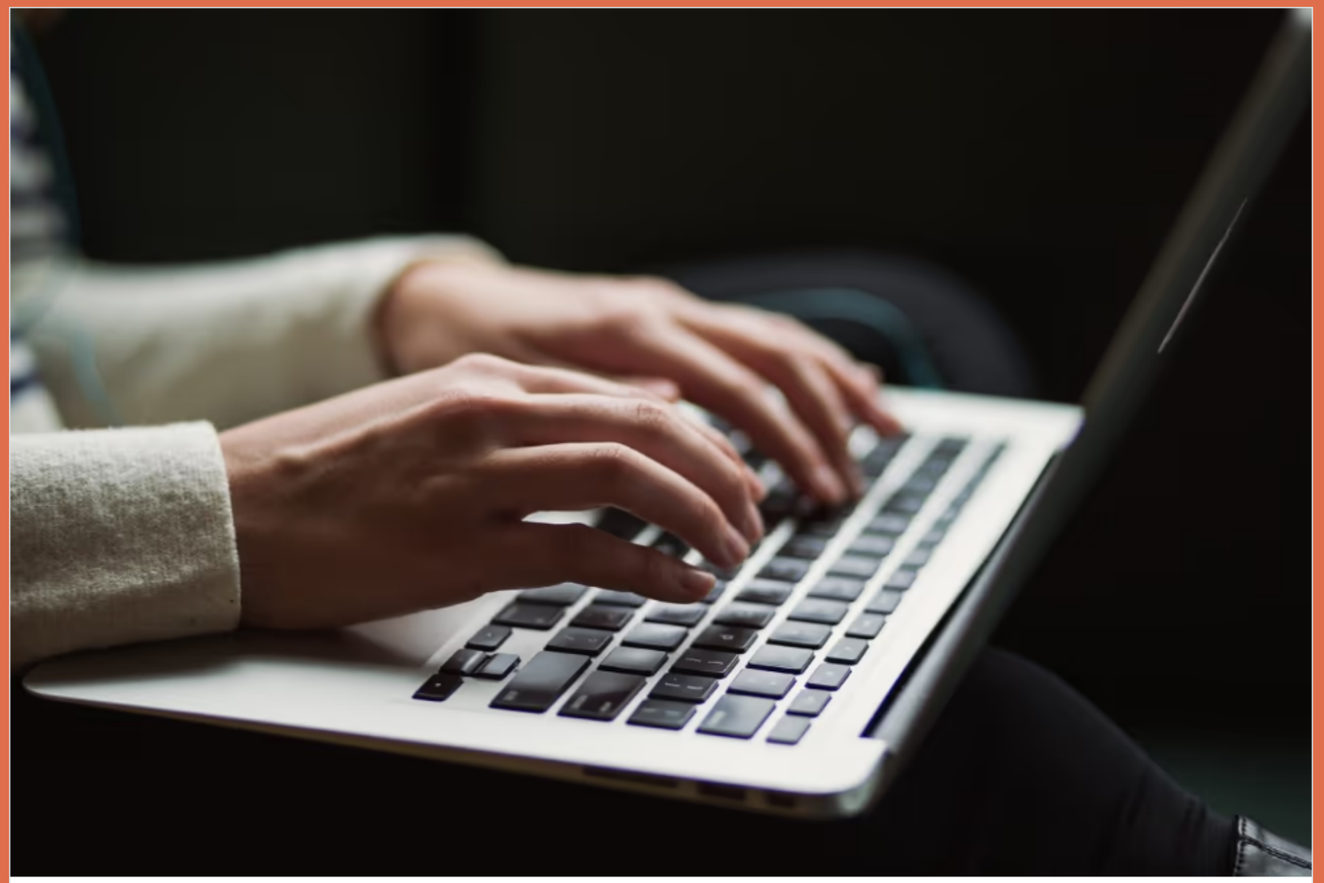
```
x, y = y, x
```

MULTIPLE ASSIGNMENT

```
import this
"""The Zen of Python, by Tim Peters. (poster by Joachim Jablon)"""

 1 Beautiful is better than ugly.
 2 Explicit is better than impl..
 3 Simple is better than cσmplεx.
 4 Cσmplεx is better than cOmp1|c@ted.
 5 Flat is better than nested.
 6 S p a r s e  is better than dense.
 7 Readability counts.
 8 Special cases aren't special enough to break the rules.
 9 Although practicality beats purity.
10 raise PythonicError("Errors should never pass silently.")
11 # Unless explicitly silenced.
12 In the face of ambiguity, refuse the temptation to guess.
13 There should be one-- and preferably only one --obvious way to do it.
14 # Although that way may not be obvious at first unless you're Dutch.
15 Now is better than ...                    never.
16 Although never is often better thanrightnow.
17 If the implementation is hard to explain, it's a bad idea.
18 If the implementation is easy to explain, it may be a good idea.
19 Namespaces are one honking great idea -- let's do more of those!
```

WRITE PYTHONIC CODE.

TUPLE UNPACKING



5 Features that Make Python Unique

Here are 5 features and syntactical sugars that make Python so different and unique.

Medium / Jun 28, 2021


python
TECHNICAL
COMMUNICATION

ALGO TEA...

How to talk through any problem.

QUICK REVIEW:

- JSON_loads, JSON_dumps
- Iterate through a dictionary & list

TECHNICAL COMMUNICATION:

Input, Output, Pseudocode

WRITE PYTHONIC CODE.

JSON Encoding / Serializing

'javascript object notation'

```
import json
```

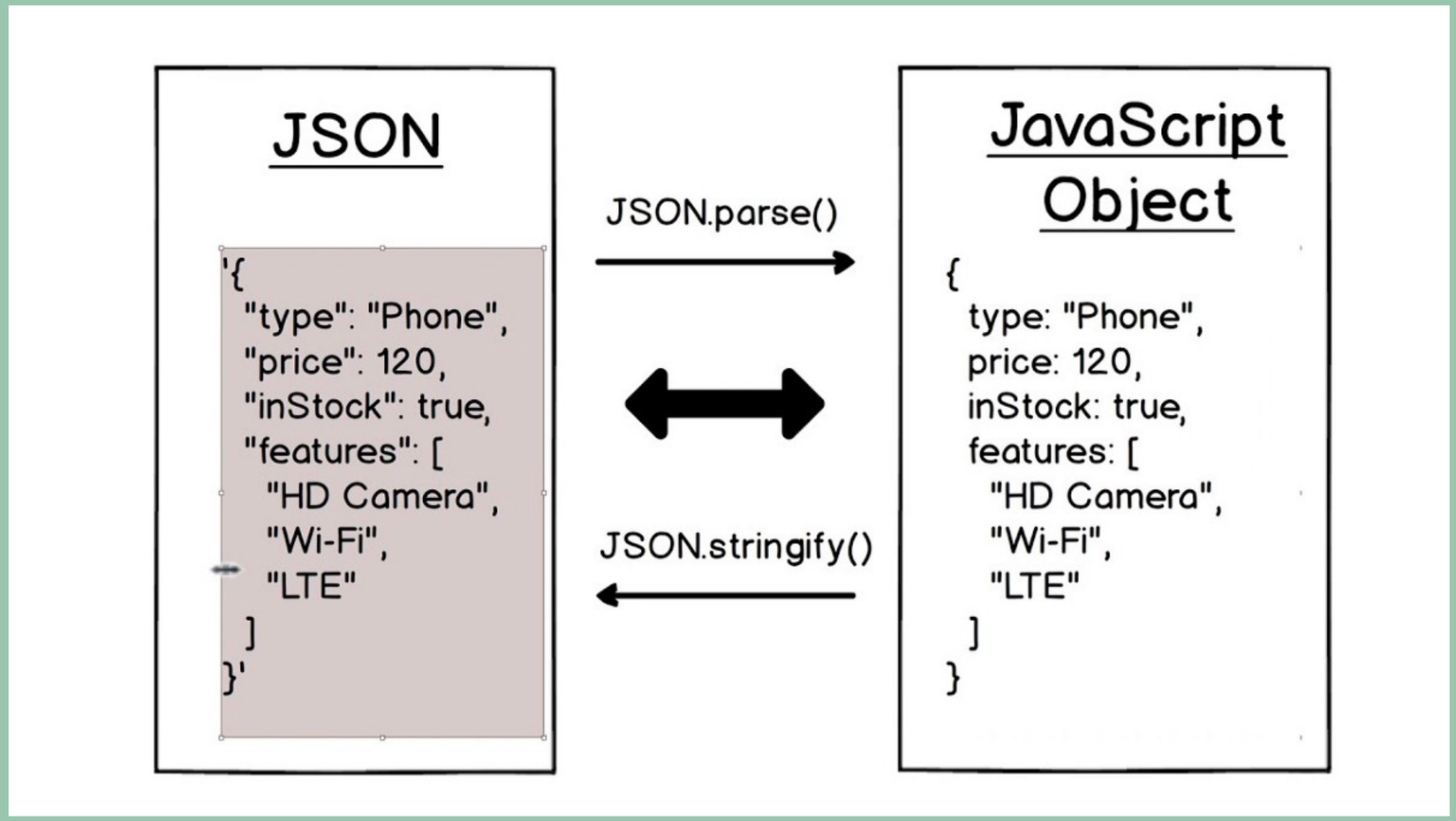
`json.loads()` json string --> python object

`json.dumps()` python object --> json string

JSON Encoding / Serializing

'javascript object notation'

WRITE JAVASCRIPT:



WRITE PYTHONIC CODE.

Iterate Through a Dictionary

.items()

Python >>>

```
>>> a_dict = {'color': 'blue', 'fruit': 'apple', 'pet': 'dog'}
>>> d_items = a_dict.items()
```

Python

```
>>> for item in a_dict.items():
...     print(item)
...
('color', 'blue')
('fruit', 'apple')
('pet', 'dog')
```

WRITE PYTHONIC CODE.

Iterate Through a Dictionary

.keys()

Python

```
>>> a_dict = {'color': 'blue', 'fruit': 'apple', 'pet': 'dog'}
>>> keys = a_dict.keys()
>>> keys
dict_keys(['color', 'fruit', 'pet'])
```

Python

```
>>> for key in a_dict.keys():
...     print(key, '->', a_dict[key])
...
color -> blue
fruit -> apple
pet -> dog
```

WRITE PYTHONIC CODE.

Iterate Through a Dictionary

.values()

Python

```
>>> a_dict = {'color': 'blue', 'fruit': 'apple', 'pet': 'dog'}
>>> values = a_dict.values()
>>> values
dict_values(['blue', 'apple', 'dog'])
```

Python

```
>>> a_dict = {'color': 'blue', 'fruit': 'apple', 'pet': 'dog'}
>>> 'pet' in a_dict.keys()
True
>>> 'apple' in a_dict.values()
True
>>> 'onion' in a_dict.values()
False
```

WRITE PYTHONIC CODE.

Iterate Through a Dictionary

.items()

Python >>>

```
>>> a_dict = {'color': 'blue', 'fruit': 'apple', 'pet': 'dog'}
>>> d_items = a_dict.items()
```

Python

```
>>> prices = {'apple': 0.40, 'orange': 0.35, 'banana': 0.25}
>>> for k, v in prices.items():
...     prices[k] = round(v * 0.9, 2)  # Apply a 10% discount
...
>>> prices
{'apple': 0.36, 'orange': 0.32, 'banana': 0.23}
```

WRITE PYTHONIC CODE.

Iterate Through a List

range()

```
range (start, stop[, step])
```

WRITE PYTHONIC CODE.

Iterate Backwards: List

range()

```
# # Initializing number from which
# # iteration begins
N = 6


# Using slice syntax perform the backward iteration
k = range(N+1)[::-1]
print("The reversed numbers are : ",end='')
for i in k:
    print(i, end=' ')
```

Output

```
The reversed numbers are : 6 5 4 3 2 1 0
```

TECHNICAL COMMUNICATION:

Input, Output, Pseudocode

STEP 1. Define inputs and outputs:

Why are you choosing these data structures?

What is your expected return value?

STEP 2. Communicate overall strategy.

What technique will you use to solve this problem?

Time/ Space complexity?

STEP 3. Pseudocode before you code.

Communicate line by line.

Communicate what you don't know.

STEP 4. Summarize.


python
TECHNICAL
COMMUNICATION
2

TECHNICAL COMMUNICATION

How to talk through any problem.

STEPS:

- 1. Define inputs and outputs.
- 2. Communicate overall strategy.
  - Think through edge cases & tests.
  - Time & space complexity.
- 3. Pseudocode.
- 4. Summarize: time & space complexity, potential refactoring.

Input, Output, Pseudocode

WRITE PYTHONIC CODE.

Python DocStrings

Python docstrings are the string literals that appear right after the definition of a function, method, class, or module.

Python docstring

A Python docstring is **a string used to document a Python module, class, function or method**, so programmers can understand what it does without having to read the details of the implementation. Also, it is a common practice to generate online (html) documentation automatically from docstrings.

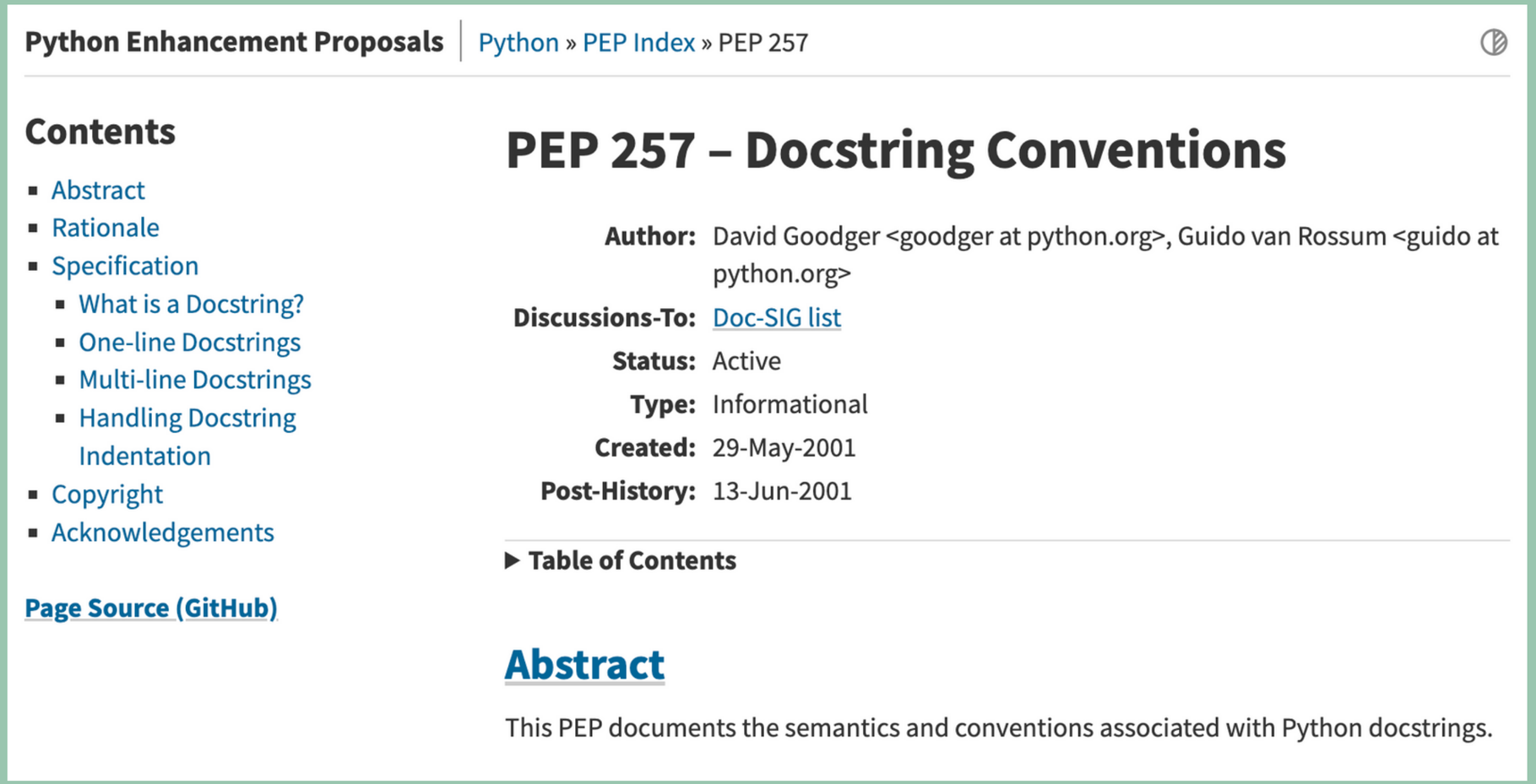
WRITE PYTHONIC CODE.

DocString Convention Example

```
def my_function(arg1):
    """
    Summary line.

    Extended description of function.

    Parameters:
    arg1 (int): Description of arg1

    Returns:
    int: Description of return value

    """

    return arg1

print(my_function.__doc__)
```

WRITE PYTHONIC CODE.

Python DocStrings



Python Enhancement Proposals | Python » PEP Index » PEP 257

Contents

Page Source (GitHub)

PEP 257 – Docstring Conventions

Author: David Goodger <goodger at python.org>, Guido van Rossum <guido at python.org>

Discussions-To: Doc-SIG list

Status: Active

Type: Informational

Created: 29-May-2001

Post-History: 13-Jun-2001

▶ Table of Contents

Abstract

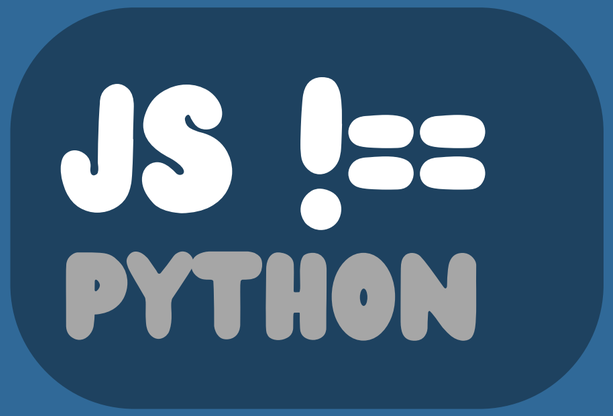
This PEP documents the semantics and conventions associated with Python docstrings.

A FEW RULES OF THUMB

JAVASCRIPT!

A few key differences from Python

- Declare your variables!
  - Use **const** for immutable variables.
  - **let** vs. **var**
- Curly brackets {} and Semicolons; !
  - Iterate through a list: **for...of**
  - Iterate through an object: **for...in**



Length is a Property, not a Method

Javascript: string1.length

Python: len(string2)

REDUCER

JAVASCRIPT + PYTHON

- Reducer is an **array method** in JS

- Reducer is a Python tool in the **functools** library

Mathematical technique:
FOLDING

The **reduce(fun,seq)** function is used to apply a particular function passed in its argument to all of the list elements.

Python

```
>>> def my_add(a, b):
...     result = a + b
...     print(f"{a} + {b} = {result}")
...     return result
```

Python

```
>>> from functools import reduce

>>> numbers = [0, 1, 2, 3, 4]

>>> reduce(my_add, numbers)
0 + 1 = 1
1 + 2 = 3
3 + 3 = 6
6 + 4 = 10
10
```

reduce() is defined in the **functools** module

Python

```
functools.reduce(function, iterable[, initializer])
```

Third optional argument:
initializer

Python

```
>>> from functools import reduce

>>> numbers = [0, 1, 2, 3, 4]

>>> reduce(my_add, numbers, 100)
100 + 0 = 100
100 + 1 = 101
101 + 2 = 103
103 + 3 = 106
106 + 4 = 110
110
```

.reduce() is an array method

reduce() Syntax

The syntax of the `reduce()` method is:

```
arr.reduce(callback(accumulator, currentValue), initialValue)
```

Example: Sum of All Values of Array


```
const numbers = [1, 2, 3, 4, 5, 6];

function sum_reducer(accumulator, currentValue) {
  return accumulator + currentValue;
}

let sum = numbers.reduce(sum_reducer);
console.log(sum); // 21
```

Example: Sum of All Values of Array

JS

```
// using arrow function
let summation = numbers.reduce(
  (accumulator, currentValue) => accumulator + currentValue
);
console.log(summation); // 21
```

SORTING NUMS

Sorting Strings

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();
fruits.reverse();
```

Sorting Numbers

```
array.sort(compareFunction)
```

- function(a, b){return a-b}

Sorting Numbers

```
const points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a-b});
```