Writing Zenlike Python

Jason C. McDonald

About Me

CEO, Lead Developer
MousePaw Media

Author, “Dead Simple Python”

CodeMouse92
IndelibleBluePen.com

The Zen of Python

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

The History of the Zen

comp.lang.python circa 1999

PEP 20

import this

Was Tim Serious?

“It was a throwaway python-list post. But like all great triumphs of literature, it was written during commercials breaks while watching professional wrestling on TV, and munching on a ham sandwich. All true!”

-Tim Peters
(to Barry Warsaw, 2020)

Readability

Readability counts.

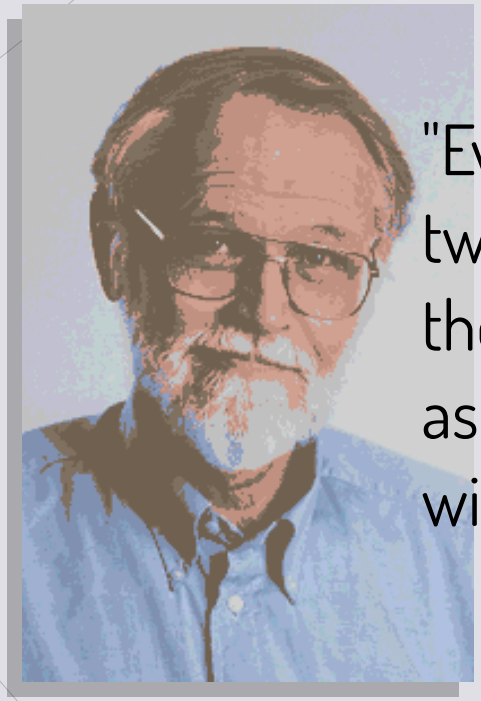
We write code for people.

We write code for our future self.

We should NOT write code to be clever.

Readability

Readability counts.



"Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?"

-Brian Kernighan
Co-author, "The C Programming Language"

Obvious

There should be one -- and preferably only one -- obvious way to do it.

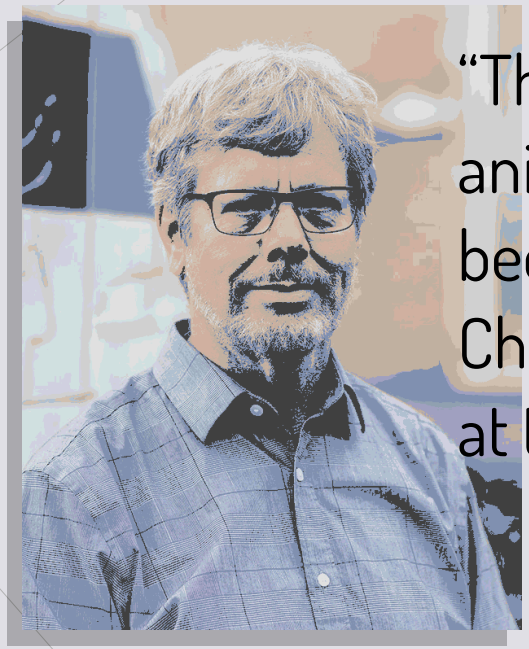
Finding the optimal solution.

Obvious once you know it.

One Way or Multiple Ways?

Obvious

There should be one -- and preferably only one -- obvious way to do it.



"The funny thing is that while there is a lot of animosity in the lower ranks, I've actually been very friendly with Larry Wall and Tom Christiansen ever since we met five years ago at the VHLL symposium that Tom organized."

-Guido van Rossum
Python creator & former BDFL

Obvious

There should be one -- and preferably only one -- obvious way to do it.

TOOWTDI

There's Only One Way To Do It (Python)

Prioritizes optimality.

Discovered by experimentation.

Obvious

There should be one -- and preferably only one -- obvious way to do it.

TMTOWTDI

There's More Than One Way To Do It (Perl)

Prioritizes experimentation.

The goal is optimality.

“Unless You’re Dutch”

Although that way may not be obvious at first unless you're Dutch.

Obvious in retrospect.

We’re not all Core Developers.

The One Obvious Way can evolve.

Beautiful

Beautiful is better than ugly.

Beautiful code is a pleasure to read.

You know it when you see it.

PEP 8 helps!

Beautiful

Beautiful is better than ugly.

```
def fizz_buzz(max):
    n=-1;r=[]
    while n<max:
        n+=1;s=""
        if n%3==0:s+="fizz"
        if n%5==0:s+="buzz"
        if n%3!=0 and n%5!=0:s=str(n)
        r+=[s]
    return r
```

Ugly

- ✗ Poor spacing.
- ✗ “Crammed” lines.
- ✗ Over-complicated.
- ✗ Hard to read.

Beautiful

Beautiful is better than ugly.

```
def fizz_buzz(max):
    return [
        "fizz" * (not n % 3) +
        "buzz" * (not n % 5)
        or str(n)
        for n in range(max + 1)
    ]
```

Beautiful

- ✔Good spacing.
- ✔One “thought” per line.
- ✔Idiomatic.
- ✔Easy to read.

Explicit

Explicit is better than implicit.

Surprises are bugs-in-waiting.

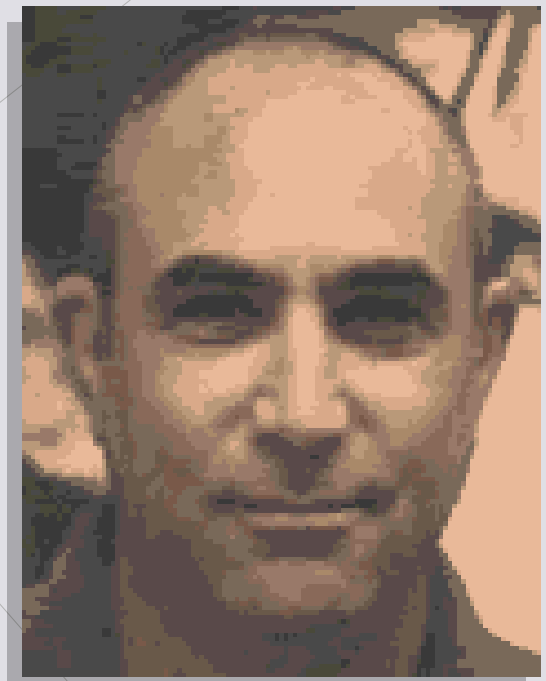
Implementation comments are failures.

(Intent comments are good!)

Naming is hard...but important.

Explicit

Explicit is better than implicit.



“There are only two hard things in Computer Science: cache invalidation and naming things.”

-Phil Karlton
Netscape Developer

Explicit

Explicit is better than implicit.

```
from constants import *

def parse(data):
    return data.split(SEP)[0]

with open("file.txt", 'r') as file:
    data = [
        parse(line)
        for line in file
    ]
```

Implicit

✗ Vague names.

✗ Where is SEP from?

✗ What does this do??!?

Explicit

Explicit is better than implicit.

```
import constants

def parse_name(line):
    return line.split(constants.SEP)[0]

# Retrieve all speaker names from file.
with open("file.txt", 'r') as file:
    speakers = [
        parse_name(line)
        for line in file
    ]
```

Explicit

- ✔Names = Purpose
- ✔Clear import.
- ✔Obvious intent.
- ✔Useful comment.

Namespaces

Namespaces are one honking great idea -- let's do more of those!

import * is evil!

Shadowing is pesky.

Where did x come from, anyway?

Namespaces

Namespaces are one honking great idea -- let's do more of those!

```
from door import *
from window import *

print("Knock, knock.")
open()
print("No solicitors!")
slam()
post_in_window("No Solicitors")
```

No Namespaces

- ✗ Shadowing.
- ✗ Unclear intent.
- ✗ Where do I edit slam()?
- ✗ If we import * this...

Namespaces

Namespaces are one honking great idea -- let's do more of those!

```
import door
import window

print("Knock, knock.")
door.open()
print("No solicitors!")
door.slam()
window.post_in_window("No Solicitors")
```

Namespaces

- ✔No shadowing.
- ✔Clear intention.
- ✔Clear origin.

Refuse to Guess

In the face of ambiguity, refuse the temptation to guess.

Guessing leads to bugs.

Look it up!

Refactor as needed.

Again: “Explicit is better than implicit.”

Simple

Simple is better than complex.

Don’t be clever.

The “basics” are your friends.

Simplicity takes skill.

Simple

Simple is better than complex.

```
import sys

def get_input():
    r = input("Hash: ")
    if r.lower() == "quit":
        sys.exit()
    return r

while True:
    string = get_input()
    print(f"{string} => {hash(string)}")
```

Complex

✗ Clever.

✗ DRY...nearing *arid*.

✗ Not bad, but...

Simple

Simple is better than complex.

```
while True:
    s = input("Hash: ")
    if s.lower() == "quit":
        break
    else:
        print(f"{s} => {hash(s)}")
```

Simple

- ✔Hooray for the classics!
- ✔Easy to understand.
- ✔Easy to maintain.

Complex

Complex is better than complicated.

Not everything is simple.

Elegance is not obfuscation.

Complex takes more time to read, not more effort.

Complex

Complex is better than complicated.

```
def find_gcf(n1, n2):
    factors1 = set()
    for f in range(1, n1 + 1):
        if n1 % f == 0:
            factors1.add(f)

    factors2 = set()
    for f in range(1, n2 + 1):
        if n2 % f == 0:
            factors2.add(f)

    factors_common = factors1 & factors2
    return max(factors_common)
```

Complicated

- ✗ Too many steps.
- ✗ Obfuscated logic.
- ✗ Ignores syntactic sugar.

Complex

Complex is better than complicated.

```
def find_gcf(n1, n2):
    factors = {
        factor
        for factor
        in range(1, min(n1, n2) + 1)
        if not n1 % factor
        and not n2 % factor
    }
    return max(factors)
```

Complex

- ✔Tightened logic.
- ✔Clear intention.
- ✔Utilize syntactic sugar.

Easy to Explain

If the implementation is hard to explain, it's a bad idea.

Simple, or at least complex.

Obvious — to the reader.

Again: Don’t be clever.

Easy to Explain

If the implementation is easy to explain, it may be a good idea.

All good code is simple (or reasonably complex.)

Not all simple code is good.

All good code is obvious.

Not all obvious code is good.

Flat

Flat is better than nested.

Nesting is hard to follow.

Nesting is easy to get wrong.

(Except in data.)

Flat

Flat is better than nested.

```
alpha = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

for r in ('WK'):
    for l1 in alpha:
        for l2 in alpha:
            for l3 in alpha:
                print(f"{r}{l1}{l2}{l3}")
```

Nested

✗ Prone to indent errors.

✗ Hard to reason about.

✗ Line limits now hurt!

Flat

Flat is better than nested.

```
import itertools

alpha = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

for r, l1, l2, l3 in itertools.product(
    'WK', alpha, alpha, alpha)
):
    print(f"{r}{l1}{l2}{l3}")
```

Flat

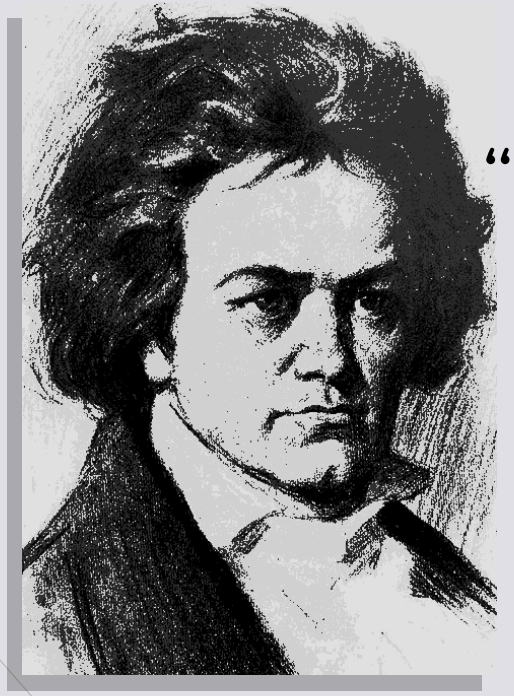
- ✔Indent errors unlikely.
- ✔Easy to reason about.
- ✔Line limits okay.

Sparse

Sparse is better than dense.

Sparse

Sparse is better than dense.

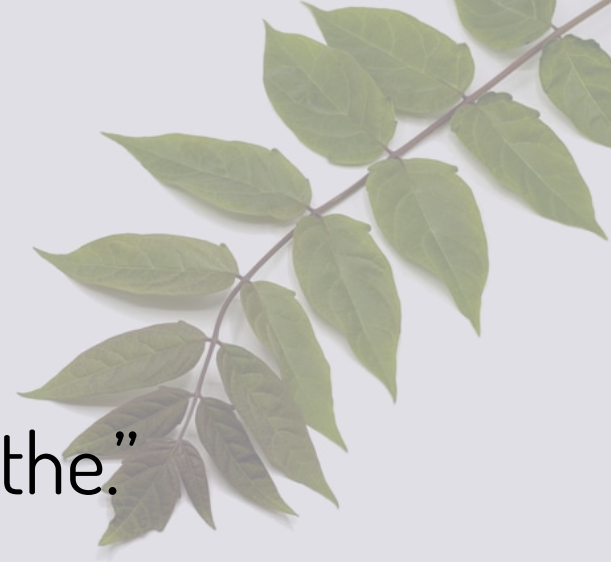


"The music is not in the notes, but in the silence between."

-Wolfgang Amadeus Mozart
Composer

Sparse

Sparse is better than dense.



Whitespace lets the code “breathe.”

Whitespace organizes code.

Beware one-liners.

Sparse

Sparse is better than dense.

```
import math
def is_palindrome(s):
    s=''.join(filter(str.isalpha,s.lower()))
    half=math.floor(len(s)/2)
    for i in range(half):
        if s[i]!=s[-(i+1)]:return False
    return True
```

Dense

✗ Do I really need to say it?

Sparse

Sparse is better than dense.

```
import math

def is_palindrome(s):
    s = ''.join(
        filter(str.isalpha, s.lower())
    )

    half = math.floor(len(s) / 2)
    for i in range(half):
        if s[i] != s[-(i + 1)]:
            return False

    return True
```

Sparse

- ✔Easy to parse.
- ✔Logical “chunks”.
- ✔Vertical space is there to use!

Errors

Errors should never pass silently.

An Error is an Exceptional State

Exceptional State requires Intelligent Intervention

No Intelligent Intervention = STOP!

‘Diaper Antipattern’

Errors

Errors should never pass silently.

In the harsh and unforgiving real world, however, the source may be tens, hundreds, or even thousands of lines away, buried under umpteen layers of abstractions in a different module or, worse, in some third-party library, and when it fails, just as diapers fail, it will fail in the small hours of the night when we would rather be sleeping, and someone important will make an awful fuss about it. It will not be fun.

-Mike Pirnat

"How to Make Mistakes in Python"

Explicitly Silenced

Unless explicitly silenced.

Intelligent Intervention already applied?

Exceptional State has been resolved.

That specific error may be silenced.

“Explicit is better than implicit.”

Now

Now is better than never.

The Lie: “I’ll Do It Later”

More delay = more refactoring

Get it over with...

...or at least pave the way!

Never

Although never is often better than *right* now.

Other Lie: “It’ll Only Take a Minute”

The urgency of the insignificant.

Beware creeping featurism.

Special Cases

Special cases aren't special enough to break the rules.

“My situation is unique.”

Yes, like every other situation.

If I make an exception for you...

Practicality

Although practicality beats purity.

Rules exist to establish order.

Don’t lose the end in the means.

The **end** grants the exception.

[ ]

“20...counting the one I'm leaving for Guido to fill in.”

The Zen is eternally unfinished.

Python is eternally unfinished.

“Pythonic” contains The Zen.

The Zen cannot contain “Pythonic.”

Writing Zenlike Python

Jason C. McDonald