

UNIVERSIDAD NACIONAL DEL ALTIPLANO  
FACULTAD DE INGENIERÍA ESTADÍSTICA E INFORMÁTICA  
ESCUELA PROFESIONAL DE INGENIERÍA ESTADÍSTICA E INFORMÁTICA



**Zen Python**

**Presentado por:**  
Maquera Andrade Aldair Jose

**Docente:**  
Ing. Fred Torres Cruz

**Semestre:**  
7mo

**Curso:**  
INGENIERIA DE SOFTWARE I

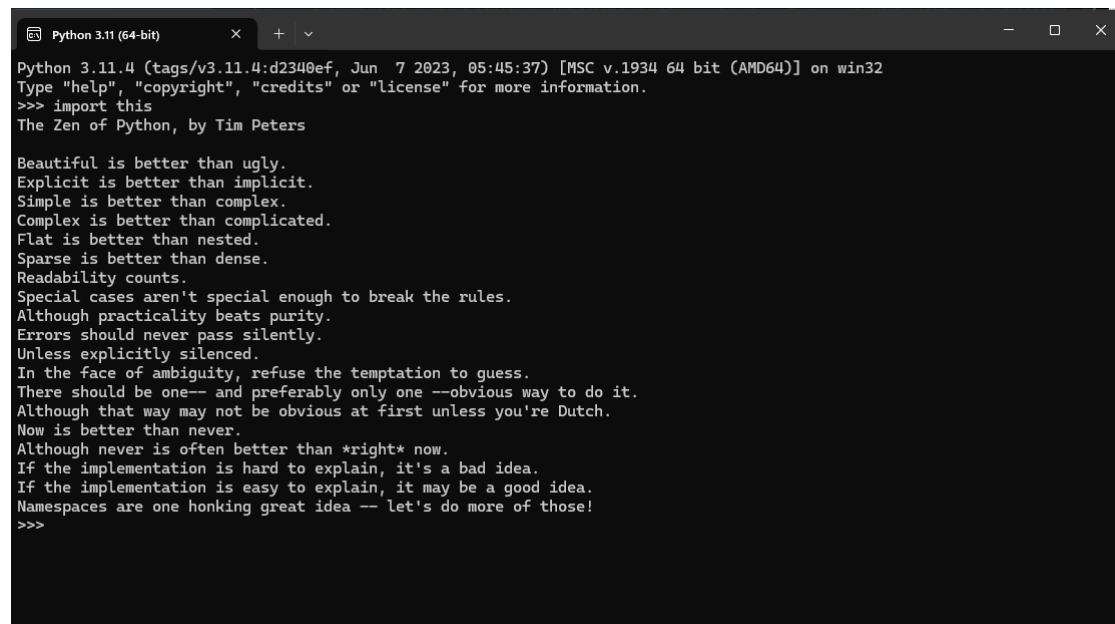
PUNO - PERÚ  
2024

## Índice

0.1. Bonito es mejor que feo . . . . .	3
0.2. Explícito es mejor que implícito . . . . .	3
0.3. Simple es mejor que complejo . . . . .	4
0.4. Complejo es mejor que complicado . . . . .	4
0.5. Plano es mejor que anidado . . . . .	4
0.6. Espaciado es mejor que denso . . . . .	5
0.7. La legibilidad importa . . . . .	5
0.8. Los casos especiales no son tan especiales como para romper las reglas . . . . .	5
0.9. Aunque la practicidad le gana a la pureza . . . . .	5
0.10. Los errores nunca deberían dejarse pasar silenciosamente . . . . .	6
0.11. A menos que hayan sido silenciados explícitamente . . . . .	6
0.12. Frente a la ambigüedad, rechaza la tentación de adivinar . . . . .	6
0.13. Debería haber una -y preferiblemente sólo una- manera obvia de hacerlo. . . . .	6
0.14. Aunque la forma no parezca obvia a la primera, a no ser que seas Holandés . . . .	6
0.15. Ahora es mejor que nunca . . . . .	7
0.16. Aunque nunca es a menudo mejor que ahora mismo . . . . .	7
0.17. Si la implementación es difícil de explicar, es una mala idea . . . . .	7
0.18. Si la implementación es fácil de explicar, es una buena idea . . . . .	7
0.19. Los 'namespaces' son una gran idea ¡Hagamos más de esas cosas! . . . . .	7

# Zen Python

If you ever open the Python console and type (import this) you will see the lines with the famous Python Zen:

A screenshot of a Python 3.11.4 (64-bit) console window. The window title is "Python 3.11 (64-bit)". The console output shows the version and build information, followed by the command ">>> import this". The output is the "Zen of Python" by Tim Peters, which lists 19 principles of Python programming. The text is as follows:

```
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun 7 2023, 05:45:37) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import this
The Zen of Python, by Tim Peters

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!
>>>
```

Python Zen consists of 19 rules that must be correctly applied to any code to turn it into Python code; by Tim Peters.

## 0.1. Bonito es mejor que feo

Es preferible escribir código que sea estéticamente agradable y fácil de entender en lugar de código confuso o desordenado.

```
# ejemplo de codigo feo
gatos=4;perros=6;patas=34;assert patas==(gatos*perros*4),'Número de patas dispar';

# ejemplo de codigo bonito
gatos = 4
perros = 6
patas = 34
assert patas == (gatos * 4) + (perros * 4), 'Número de patas dispar'
```

## 0.2. Explícito es mejor que implícito

Es mejor ser claro y explícito en lugar de dejar cosas al azar o a la imaginación del lector.

```
# version implicita
def metros_a_pulgadas_dobles(metros):
    return metros * 39.3701 * 2

# version explicita
def metros_a_pulgadas_dobles(metros):
    pulgadas_por_metro = 39.3701
    multi_doble = 2
    return metros * pulgada_por_metro * multi_doble
```

### 0.3. Simple es mejor que complejo

Es preferible la simplicidad en el diseño y la implementación del código.

```
# Complejo
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

# Simple
import math
factorial = math.factorial
```

### 0.4. Complejo es mejor que complicado

Si una tarea requiere complejidad, es mejor que esta sea evidente y comprensible en lugar de confusa.

```
# Complicado
try:
    resultado = accion()
except Excepcion1:
    resultado = fallback()
except Excepcion2:
    resultado = fallback()
except Excepcion3:
    resultado = fallback()
...

# Complejo
from functools import partial
fallback = partial(accion)
resultado = fallback()
```

### 0.5. Plano es mejor que anidado

Evitar la anidación excesiva de estructuras de control como bucles y condicionales, ya que puede hacer que el código sea más difícil de entender.

```
# Anidado
for i in range(3):
    for j in range(3):
        print(i, j)

# Plano
for i in range(3):
    print(i)
for j in range(3):
    print(j)
```

## 0.6. Espaciado es mejor que denso

Es preferible que el código sea espaciado y bien estructurado en lugar de estar sobrecargado y difícil de leer.

```
# version densa
print(','.join(map(str, [x ** 2 for x in range(5)])))

# version dispersa
a = [x ** 2 for x in range(5)]
b = map(str, a)
c = ','.join(b)
print(c)
```

## 0.7. La legibilidad importa

El código debe ser escrito de manera que sea fácil de entender para otros programadores, así como para tu yo futuro.

```
# Poco legible
a = 4; b = 5; c = a + b

# Legible
ancho = 4
largo = 5
suma = ancho + largo
```

---

```
# version legible
lista_de_cuadrados = [x ** 2 for x in range(5)]
cadenas_de_cuadrados = map(str, lista_de_cuadrados)
cuadrados_formateados = ','.join(cadenas_de_cuadrados)
print(cuadrados_formateados)
```

## 0.8. Los casos especiales no son tan especiales como para romper las reglas

Trata de seguir las convenciones y prácticas estándar de codificación en lugar de crear casos especiales que puedan causar confusión.

```
# Especial
if condicion_especial():
    hacer_algo_especial()
else:
    hacer_algo_normal()

# General
if not condicion_especial():
    hacer_algo_normal()
```

## 0.9. Aunque la practicidad le gana a la pureza

A veces, es necesario sacrificar la pureza del diseño en favor de la practicidad y la utilidad.

```
# Puro
from math import sqrt

# Práctico
import numpy as np
```

### 0.10. Los errores nunca deberían dejarse pasar silenciosamente

Es importante manejar adecuadamente los errores en el código y no ignorarlos a menos que haya una razón específica para hacerlo.

```
try:
    codigo_erroneo()
except Exception:
    pass
```

### 0.11. A menos que hayan sido silenciados explícitamente

Existen excepciones al caso anterior que se dan cuando se ha estudiado la situación, se conoce el error y explícitamente se silencia (o se actúa en consecuencia).

Si el error que se ha detectado es un `ValueError` (por ejemplo) pero se sabe que no es problemático, se debe de manejar adecuadamente, incluso pudiendo ser silenciado.

```
try:
    codigo_erroneo()
except ValueError:
    logger.debug('Value Error manejado correctamente')
```

### 0.12. Frente a la ambigüedad, rechaza la tentación de adivinar

Este concepto aplica a muchos ámbitos del desarrollo, es mucho mejor tener claro qué se está construyendo y poder crear tests, que exactamente definan y comprueben el buen comportamiento del sistema, eliminando cualquier ambigüedad.

### 0.13. Debería haber una -y preferiblemente sólo una- manera obvia de hacerlo.

Python fomenta la simplicidad y la consistencia en el diseño y la implementación.

```
# Varios caminos
resultado = funcion()
resultado = objeto.funcion()

# Una manera obvia
resultado = objeto.funcion()
```

### 0.14. Aunque la forma no parezca obvia a la primera, a no ser que seas Holandés

Es normal que no aparezca la forma obvia la primera vez que piensa la solución, por lo que puede requerir de un método iterativo.

El guiño que se añade en esta regla sobre los holandeses hace referencia la nacionalidad del Dictador Benevolente y creador de Python Guido van Rossum.

### **0.15. Ahora es mejor que nunca**

En el desarrollo de software siempre hay tareas que realizar y si no se priorizan las tareas importantes para hacerlas cuando salen los problemas, estos pueden alargarse en el tiempo hasta no hacerlas nunca.

### **0.16. Aunque nunca es a menudo mejor que ahora mismo**

Aunque es importante actuar en el momento presente, es preferible hacer las cosas correctamente en lugar de apresurarse y arriesgar la calidad.

### **0.17. Si la implementación es difícil de explicar, es una mala idea**

La complejidad excesiva en la implementación puede ser un indicador de un diseño deficiente.

### **0.18. Si la implementación es fácil de explicar, es una buena idea**

La simplicidad y la claridad en la implementación son generalmente indicativas de un buen diseño.

### **0.19. Los 'namespaces' son una gran idea ¡Hagamos más de esas cosas!**

El uso de espacios de nombres ayuda a evitar colisiones de nombres y a organizar el código de manera más limpia y modular.

## **Importancia del Zen de Python**

El Zen de Python podría exportarse a otros lenguajes dado que muchas de sus reglas no son únicas para Python sino aplicables a cualquier lenguaje.

Dado que Python es uno de los pocos lenguajes con unas directrices tan claras hace que el lenguaje sea aún más extraordinario.