Cuando la solución más obvia, no es la mejor

MAMÁ

Cortaste todda la looz

Holus

Acá Diego Cañizares

Coder. Pythonero. Formador.

diegocanizares@gmail.com

1 DISCLAIMER


SAQUE EL PAN, RICARDO

2

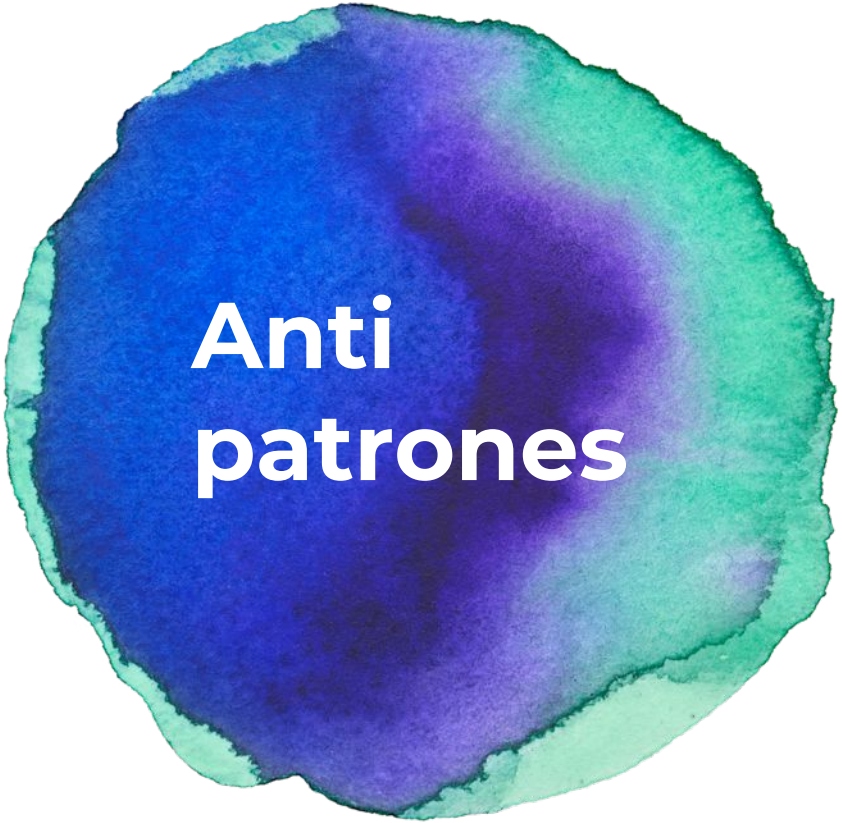
¿QUÉ VAMOS A VER?

3

DURACIÓN:

60 MINUTOS

ARRANCAMO'



- El desarrollo de software es caótico
- Seguro caíste en alguno
- Todos implican refactor

IMPORTANTE: ¡está bien refactorizar!

“Hay dos formas de escribir programas sin errores. Sólo la tercera funciona.”

Algunos antipatrones

The Blob

*"Esta clase es realmente **el corazón** de la solución"*

Golden Hammer

*"Por ahí después de todo en realidad no teníamos que usar **Excel** para esto"*

Spaguetti Code

"&%$#@?!*$#*@"*

Un par más

Cut & Paste

*"Creí que arreglaron ese bug, pero en esta otra sección **está pasando de nuevo**"*

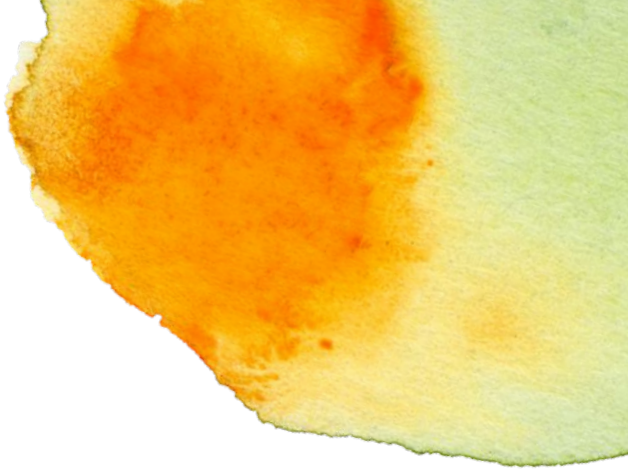
Lava Flow

*"Ahhh, ¡**eso**! Pau y Juan lo escribieron cuando Ro quería ayudar a Irene. Ni idea si se sigue usando, nadie lo documentó. **Igual anda.**"*


OK
¿Y EN PYTHON?

CORRECTNESS

“Los programadores de verdad no documentan.
Si fue difícil de escribir, debe ser difícil de entender” (?)



Lambdas

No le enchufes nombre a lo que es anónimo.

Java style

Estás en Python. No escribas código Java.

Monkey patching

No, no está bueno que cambies cosas built-in.

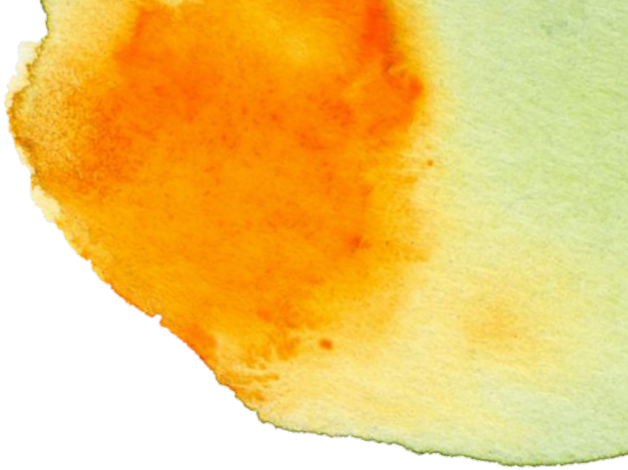
Lambdas

No:

```
f = lambda x: 2*x
```

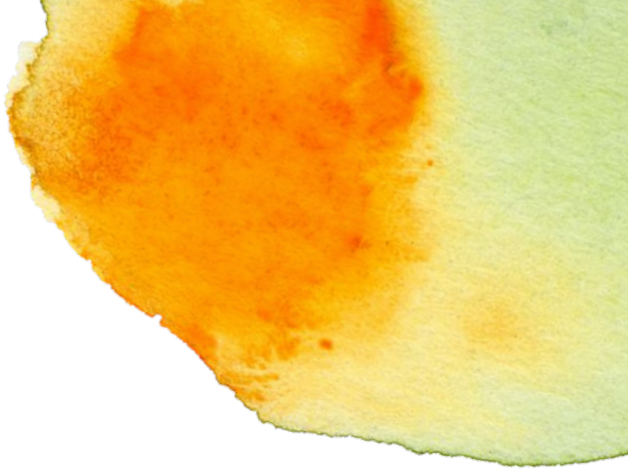
Sí:

```
def f(x): return 2*x
```



Java style

```
class Square(object):
    def __init__(self, length):
        self._length = length

    @property
    def length(self):
        return self._length

    @length.setter
    def length(self, value):
        self._length = value

r = Square(5)
r.length  # automatically calls getter
r.length = 6  # automatically calls setter
```



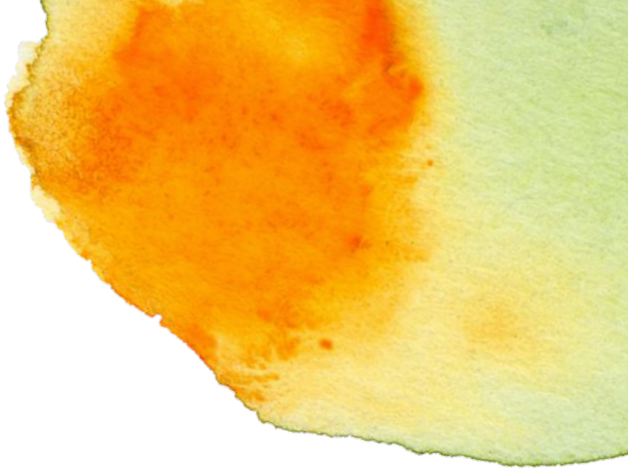
Monkey patching

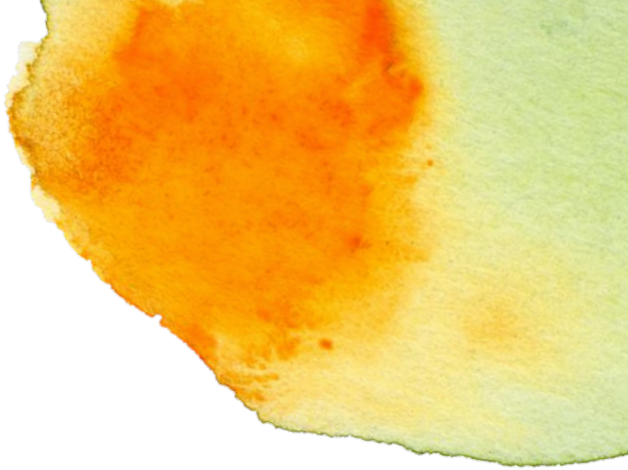
No:

```
list = [1, 2, 3]

students = list()
```

Sí:

```
numbers = [1, 2, 3]

students = list()
```





Mutables

OJOTA con los tipos de datos mutables por parámetro

Else en for

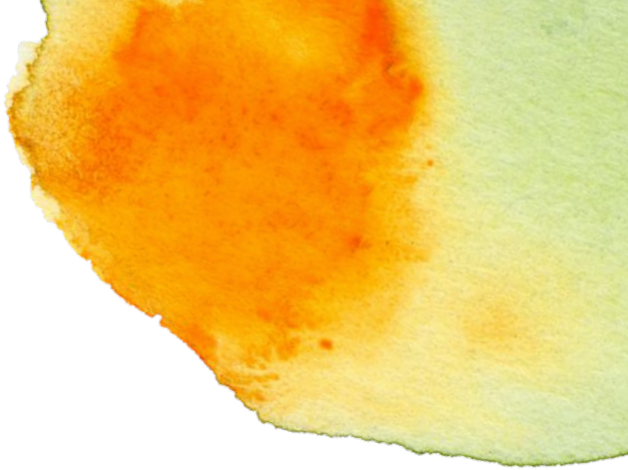
¿Sabías que el for tiene else? Usalo, ¡vale la pena!

No usar get

Usar diccionarios SIN get es receta para quilombo

Mutables

```
def append(number, number_list=[]):
    number_list.append(number)
    print(number_list)
    return number_list

append(5) # [5] => [5]
append(7) # [7] => [5, 7]
append(2) # [2] => [5, 7, 2]
```



Else en for

```
l = [1, 2, 3]
magic_number = 4

for n in l:
    if n == magic_number:
        print("Magic number!")
        break
else:
    print("No magic number :(")
```

No usar get

No:

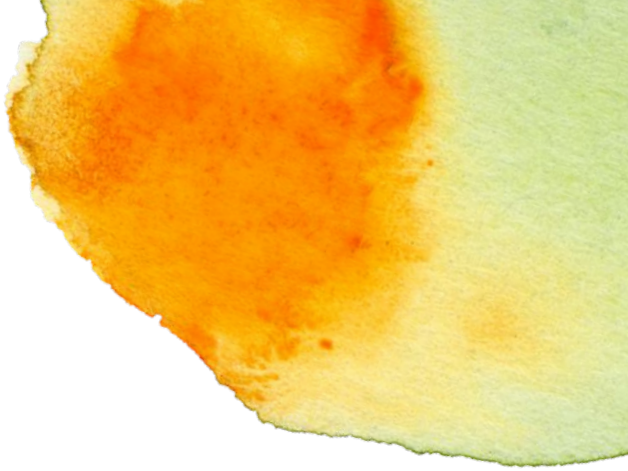
```
dictionary = {"message": "Hello, World!"}
print(dictionary["message"])  # "Hello, World!"
print(dictionary["clave_inexistente"])  # Explota explota que explo
```

Sí:

```
dictionary = {"message": "Hello, World!"}
print(dictionary.get("message", "Te la debo"))  # "Hello, World!"
print(dictionary.get("clave_inexistente", "Te la debo"))  # "Te la debo"
```

MAINTAINABILITY

“Los programas deben ser escritos para que la gente los lea y sólo incidentalmente, para que las máquinas los ejecuten.”



Imports locos

No, definitivamente NO necesitás importar todo. Nos calmamos.

Globales

¡Digale "NO" a las variables globales!

Variables de una letra

```
d = {'data': [{'a': 'b'}, {'b': 'c'},
{'c': 'd'}], 'texts': ['a', 'b', 'c']}

for k, v in d.iteritems():
    if k == 'data':
        for i in v:
            # ¿Qué se está iterando?
            for k2, v2 in i.iteritems():
                print(k2, v2)
```

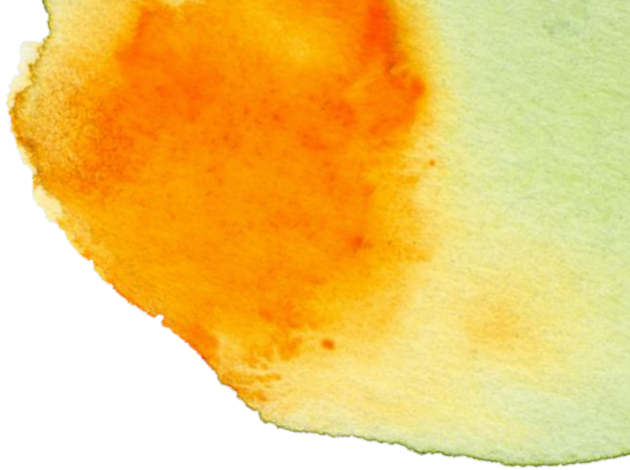
Imports locos

No:

```
from math import *
```

Sí:

```
from math import ceil
```



Globales

```
WIDTH = 0 # global variable
HEIGHT = 0 # global variable

def area(w, h):
    global WIDTH # global statement
    global HEIGHT # global statement
    WIDTH = w
    HEIGHT = h
    return WIDTH * HEIGHT

print("WIDTH:" , WIDTH) # "WIDTH: 0"
print("HEIGHT:" , HEIGHT) # "HEIGHT: 0"

print("area():" , area(3, 4)) # "area(): 12"

print("WIDTH:" , WIDTH) # "WIDTH: 3"
print("HEIGHT:" , HEIGHT) # "HEIGHT: 4"
```

Variables de una letra

```
data_dict = {
    'data': [{'a': 'b'}, {'b': 'c'}, {'c': 'd'}],
    'texts': ['a', 'b', 'c']
}


for key, value in data_dict.iteritems():
    if key == 'data':
        for data_item in value:
            # ¿Qué se está iterando? Mucho más claro ;-)
            for data_key, data_value in data_item.iteritems():
                print(data_key, data_value)
```

READABILITY

“Cualquier tonto puede escribir código que un ordenador entiende. Los buenos programadores escriben código que los humanos pueden entender.”

¿Perdón o permiso?

Es más fácil pedir perdón que pedir permiso, joven padawan.

Namedtuples

Amigate. Sirven y mucho.

¿Perdón o permiso?

No:

```
import os

if os.path.exists("file.txt"):
    os.unlink("file.txt")
```

Sí:

```
import os

try:
    os.unlink("file.txt")
except OSError as err:
    print(“¡Error!”, err)
```

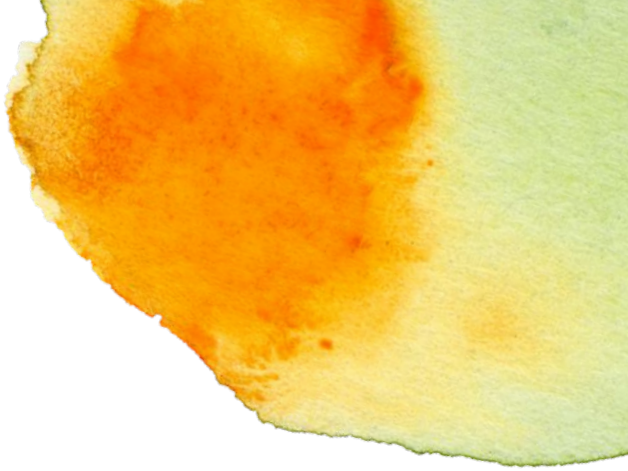
Namedtuples

No:

```
def get_name():
    return "Richard", "Xavier", "Jones"

name = get_name()

print(name[0], name[1], name[2])
```

Sí:

```
from collections import namedtuple

def get_name():
    name = namedtuple("name", ["first", "middle", "last"])
    return name("Richard", "Xavier", "Jones")

name = get_name()

print(name.first, name.middle, name.last)
```



CamelCase

No estamos en Java ¬¬

zip()

Iterá dos listas de una sola vez, de forma simple

Nombres de variables

Si ponen el tipo de dato en el nombre de la variable, **HAY TABLA**

CamelCase

No:

```
def someFunction():
    print("El PEP 8 se ponió triste")
```

Sí:

```
def some_function():
    print("El PEP 8 está felí")
```

zip()

No:

```
numbers = [1, 2, 3]
letters = ["A", "B", "C"]

for index in range(len(numbers)):
    print(numbers[index], letters[index])
```

Sí:

```
numbers = [1, 2, 3]
letters = ["A", "B", "C"]

for numbers_value, letters_value in zip(numbers, letters):
    print(numbers_value, letters_value)
```

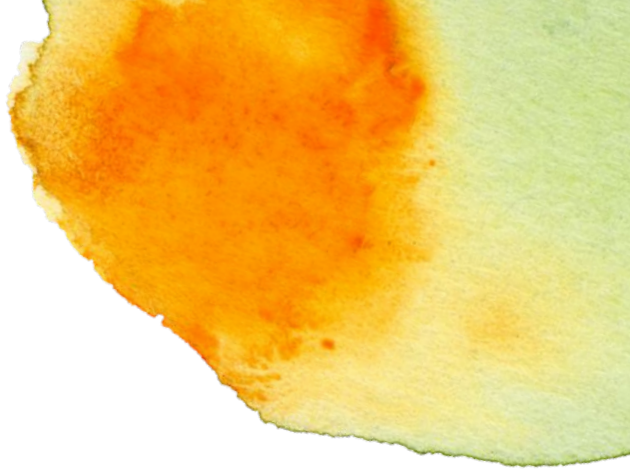
Nombres de variables

No:

```
n_int = "Hello, World!"

# Error por asumir que n_int es un número
4 / n_int
```

Sí:

```
n = "Hello, World!"

# Todavía hay error, pero menos enroscado.
4 / n
```



SECURITY & PERFORMANCE

“Computadora: dispositivo diseñado para automatizar errores y realizarlos más rápidamente”

Security

Usar exec

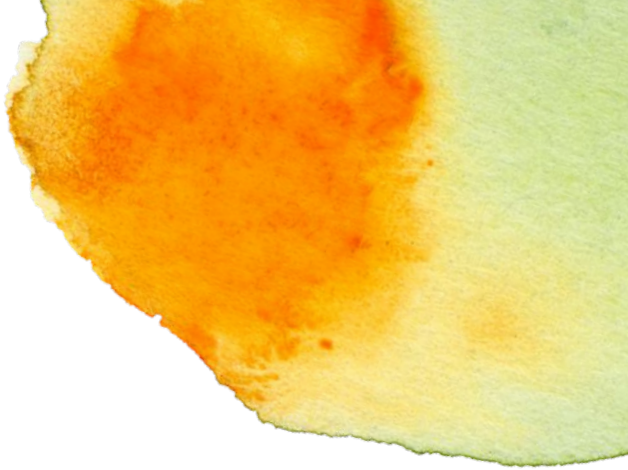
"¡Esa brecha de seguridad sí se puede ver!"

No:

```
s = "print(\"Hello, World!\")"
exec s
```

Sí:

```
def print_hello_world():
    print("Hello, World!")

print_hello_world()
```



Performance

Diccionarios grandes

Usar iteritems() en lugar de items()

Buscar eficientemente

¡Amigate con los conjuntos!

Diccionarios grandes

No:

```
d = {i: i * 2 for i in xrange(10000000)}

# Me vuelvo viejo...
for key, value in d.items():
    print("{0} = {1}".format(key, value))
```

Sí:

```
d = {i: i * 2 for i in xrange(10000000)}

# La vida e' beia
for key, value in d.iteritems():
    print("{0} = {1}".format(key, value))
```

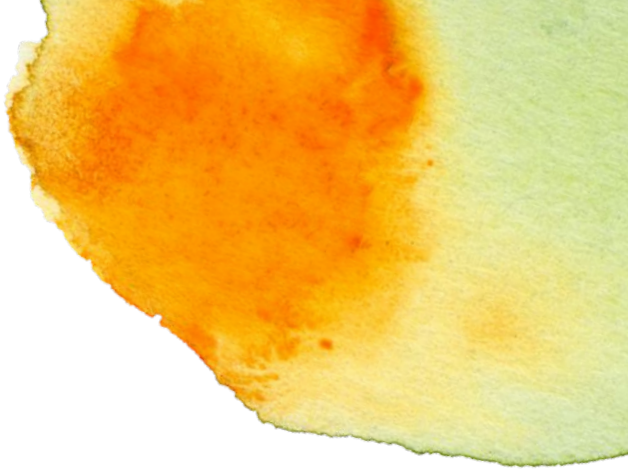
Buscar eficientemente

No:

```
l = [1, 2, 3, 4]

if 3 in l:
    print("The number 3 is in the list.")
else:
    print("The number 3 is NOT in the list.")
```

Sí:

```
s = set([1, 2, 3, 4])

if 3 in s:
    print("The number 3 is in the list.")
else:
    print("The number 3 is NOT in the list.")
```



Los tres puntos claves:

¡Muchamable!

Gracias, ¡vuelva prontos!

Me encontrás en:

linkedin/in/diegocanizares

diegocanizares@gmail.com