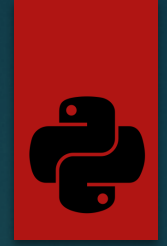
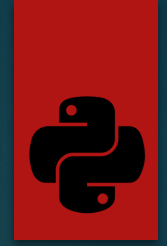


Introducción al estilo Pythonista

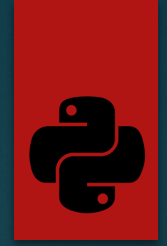
MILTON LENIS

Milton Lenis



- Desarrollador web
- 2 Años de experiencia trabajando con Python y Django
- Líder de desarrollo en RADY Consultores

The hitchhiker's guide to Python

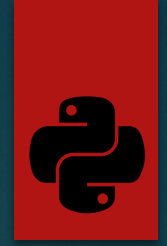


- "When a veteran Python developer (a Pythonista) calls portions of code not "Pythonic", they usually mean that these lines of code do not follow the common guidelines and fail to express its intent in what is considered the best (hear: most readable) way".

¿Qué es ser Pythonista?

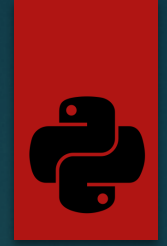
- Pasión
- Sed de aprendizaje
- Constancia
- Inconformismo
- **Idioma Pythonico**

¿Qué es un idioma?



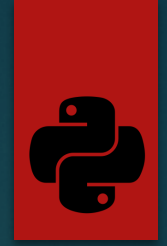
- Gramática, sintaxis y caracteres específicos de un lenguaje determinado
- **Una manera comúnmente utilizada y entendida de expresar un hecho, una idea o una intención**
- **Modo particular de hablar de algunas personas**

¿Porqué son importantes los idiomas Pythonicos?



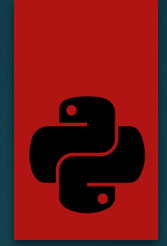
- “La mayoría de las veces el código está siendo leído en vez de escrito” - Guido Van Rossum
- Ayudan a producir código claro y legible
- Algunos idiomas pueden ser más rápidos o consumir menos memoria que su contraparte ‘no-idiomática’
- Son un punto de común acuerdo entre Pythonistas y facilitan la comunicación entre ellos, especialmente cuando se trabaja en Open Source

Recomendaciones Iniciales



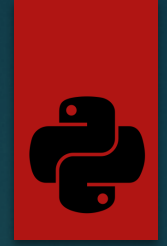
- Seguir fielmente PEP8
- Leer, entender y aplicar el Zen de Python

PEP8 (Python Enhancement Proposals # 8)

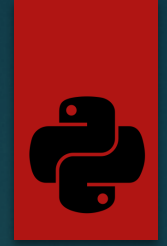


- Guía de estilos para Python
- Existen muchos otros PEPs (Cientos de ellos)
- El código sigue funcionando sin PEP8
- Algunos IDEs y herramientas ayudan a detectar errores de PEP8
- PEP8 For Humans – Kenneth Reitz (http://pep8.org/)

Zen of Python (PEP20)



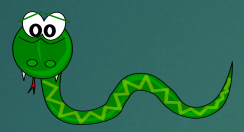
```
>>> import this
The Zen of Python, by Tim Peters

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!
```



Idiomas Pythonicos…

Expresiones por línea

No Pythonista

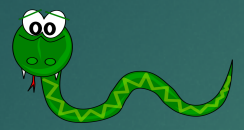


```
print ('one'); print ('two')

x = 1
if x == 1: print ('one')

for number in [1, 2, 3]: print(number)

if (1 + 2) > 5 and max([1, 2, 3]) == 5:
    print("WTF!")

```

Pythonista

```
print ('one')
print ('two')

x = 1
if x == 1:
    print ('one')

for number in [1, 2, 3]:
    print(number)

condition1 = (1 + 2) > 5
condition2 = max([1, 2, 3]) == 5

if condition1 and condition2:
    print("WTF!")

```

Representación Booleana

No Pythonista 

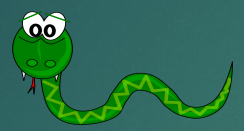
```
1    if x == True:
2        print(x)
3
4    if len(my_list) > 0:
5        print(my_list)
```

Pythonista 

```
1    if x:
2        print(x)
3
4    if my_list:
5        print(my_list)
```

False	True
`False` (== 0)	`True` (== 1)
`""` (empty string)	any string but `""` (`" "`, `"anything"`)
`0`, `0.0`	any number but `0` (1, 0.1, -1, 3.14)
`[]`, `()`, `{}`, `set()`	any non-empty container (`[0]`, `(None,)`, `['']`)
`None`	almost any object that's not explicitly False

Formatear strings

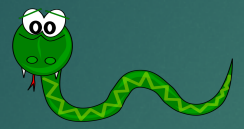
No Pythonista 

Pythonista 

```
1    nombre = "Milton"
2    apellido = "Lenis"
3    edad = "23"
4
5    cadena = "Mi nombre es: " + nombre + " " + apellido + " (" + edad + ")"
6    # Produce 'Mi nombre es Milton Lenis (23)'
```

```
1    nombre = "Milton"
2    apellido = "Lenis"
3    edad = "23"
4
5    cadena = "Mi nombre es: {0} {1} ({2})"
6    cadena = cadena.format(nombre, apellido, edad)
7    # Produce 'Mi nombre es Milton Lenis (23)'
```

Intercambiar valores entre variables

No Pythonista 

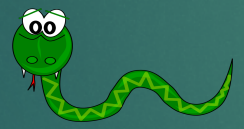
```
1    a = 1
2    b = 2
3
4    temp = a
5    a = b
6    b = temp
7
```

Pythonista 

```
1    a = 1
2    b = 2
3
4    a, b = b, a
5
```

Acceder a los valores de un diccionario

```python
my_dict = {
    'key1': 1,
    'key2': 2
}
```

No Pythonista 

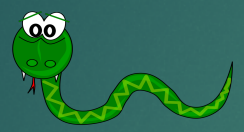
```python
try:
    value = my_dict['some_key']
except KeyError:
    value = None
```

Pythonista 

```python
value = my_dict.get('some_key', None)
```

For Loops

No Pythonista



```
1    colors = ['red', 'green', 'blue']
2    i = 0
3    while i < len(colors):
4        print(i, colors[i]))
5        i += 1
```

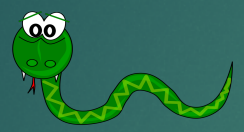
```
1    colors = ['red', 'green', 'blue']
2    for i in range(len(colors)):
3        print(i, color[i])
4
```

Pythonista

```
1    colors = ['red', 'green', 'blue']
2    for color in colors:
3        print(color)
4
```

```
1    colors = ['red', 'green', 'blue']
2    for index, color in enumerate(colors):
3        print(index, color)
4
```

Buscar ítems en un iterable

No Pythonista 

```
3    message = "Python is so cool"
4
5    if message.find('Java') != -1:
6        print("Are you kidding me?")
7
```

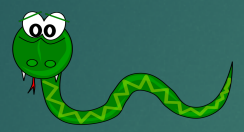
Pythonista 

```
3    pets = ['Cats', 'Dogs', 'Birds']
4
5    if 'Dogs' in pets:
6        print("I like Dogs")
7
```

```
3    message = "Python is so cool"
4
5    if 'Java' in message:
6        print("Are you kidding me?")
7
```

Crear strings a partir de una lista

No Pythonista



```
colors = ['red', 'blue', 'green', 'yellow']
result = ''
for color in colors:
    result += color
```

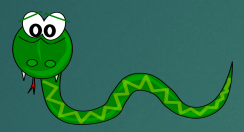
```
skills = ['Dancing', 'Hiking', 'Joking', 'Teaching']

skills_with_comma = ''
for skill in skills:
    skills_with_comma += skill + ', '

```

Pythonista

```
colors = ['red', 'blue', 'green', 'yellow']
result = ' '.join(colors)
```

```
skills = ['Dancing', 'Hiking', 'Joking', 'Teaching']

', '.join(skills)

```

Leer un archivo

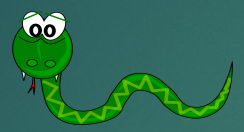
No Pythonista 

Pythonista 

```
2    text_file = open(path, "rb")
3    result = do_something_with(text_file)
4    text_file.close()
5    print("Result is: {}".format(result))
```

```
3    with open(path, "rb") as text_file:
4        result = do_something_with(text_file)
5    print("Got result: {}".format(result))
6
```

Preguntas VS Excepciones

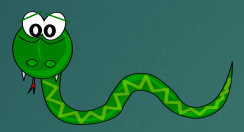
No Pythonista 

```
3    my_dict = {'x': '5'}
4
5    my_dict = {'x': '5'}
6    if 'x' in my_dict and \
7        isinstance(my_dict['x'], str) and\
8        my_dict['x'].isdigit():
9        value = int(my_dict['x'])
10   else:
11       value = None
12
```

Pythonista 

```
3    my_dict = {'x': '5'}
4
5    try:
6        value = int(my_dict['x'])
7    except (KeyError, TypeError, ValueError):
8        value = None
9
```

Generar listas

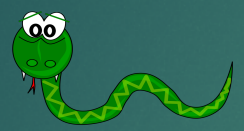
No Pythonista 

```
4    numbers = [1, 2, 3, 4, 5, 6, 7]
5
6    selected_numbers = []
7    for number in numbers:
8        if number > 5:
9            selected_numbers.append(number * 2)
0
```

Pythonista 

```
4    numbers = [1, 2, 3, 4, 5, 6, 7]
5
6    selected_numbers = [number * 2 for number in numbers if number > 5]
7
```

Crear un string de tamaño N

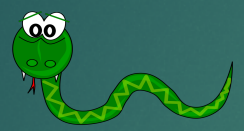
No Pythonista 

```
3    password = "MiltonLn04"
4
5    hidden_password = ''
6    for _ in password:
7        hidden_password += '*'
8
```

Pythonista 

```
3    password = "MiltonLn04"
4    hidden_password = '*' * len(password)
```

Importación de paquetes

No Pythonista 

```
3    from pets.models import *
4    from friends.models import *
5
6    Photos.objects.all()
```

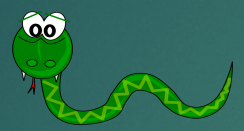
Pythonista 

```
3    from pets import models as pets_models
4    from friends import models as friends_models
5
6    pets_models.Photos.objects.all()
7    friends_models.Photos.objects.all()
```

```
3    from friends.models import Friend, Photos, Moments
4    from pets.views import PetCreateView, PetUpdateView
5
```

Documentación del código

No Pythonista

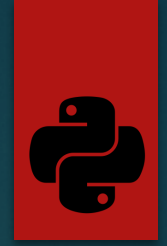


```python
def validate(self, data):
    fields_to_validate = ['volume', 'folio', 'serial', 'registration_date']
    volume, folio, serial, registration_date = [data.get(field, None) for field in fields_to_validate]

    if serial and any([volume, folio, registration_date]):
        raise serializers.ValidationError(
            _("Si el serial está incluido, no puede haber volume, folio o registration_date en la petición")
        )
    elif registration_date and any([volume, folio, serial]):
        raise serializers.ValidationError(
            _("Si el registration_date está incluido, no puede haber volume, folio o serial en la petición")
        )
    elif volume and not folio:
        raise serializers.ValidationError(
            _("Se envió el volume pero no se envío el folio")
        )
    elif folio and not volume:
        raise serializers.ValidationError(
            _("Se envió el folio pero no se envío el volume")
        )
    return data
```

Pythonista

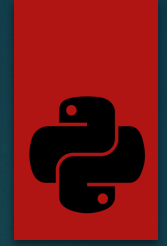
```python
def validate(self, data):
    """
    Autor: Milton Lenis
    Fecha: Mayo 24 2017
    Se utiliza el método validate para hacer la validación de los campos condicionales
    Los casos son:
    1. Si llega el serial, no puede llegar el folio, tomo ni fecha de registro
    2. Si llega la fecha de registro, no puede llegar el tomo, folio ni serial
    3. Si llega el tomo, debe llegar el folio y no puede llegar el serial y la fecha
    4. Si llega el folio, debe llegar el tomo y no puede llegar el serial y la fecha
    :param data: Instancia de los datos para validar
    :return: Instancia de los datos validados
    """
    fields_to_validate = ['volume', 'folio', 'serial', 'registration_date']
    volume, folio, serial, registration_date = [data.get(field, None) for field in fields_to_validate]

    # Caso #1
    if serial and any([volume, folio, registration_date]):
        raise serializers.ValidationError(
            _("Si el serial está incluido, no puede haber volume, folio o registration_date en la petición")
        )
    # Caso #2
    elif registration_date and any([volume, folio, serial]):
        raise serializers.ValidationError(
            _("Si el registration_date está incluido, no puede haber volume, folio o serial en la petición")
        )
    # Caso #3
    elif volume and not folio:
        raise serializers.ValidationError(
            _("Se envió el volume pero no se envío el folio")
        )
    # Caso #4
    elif folio and not volume:
        raise serializers.ValidationError(
            _("Se envió el folio pero no se envío el volume")
        )
    return data
```

Recomendaciones Finales

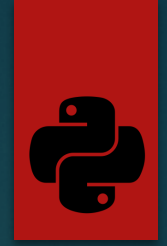


- Un gran poder conlleva una gran responsabilidad
- Siempre ser muy críticos con su trabajo y mejorar constantemente
- Leer código de desarrolladores más experimentados
- Enseñar a otros es una buena manera de afianzar sus conocimientos

Lecturas recomendadas



- Pythonic Code Review - https://access.redhat.com/blogs/766093/posts/2802001
- The Hitchhiker's Guide to Python - http://docs.python-guide.org/en/latest/writing/style/
- Code like a Pythonista: Idiomatic Python, David Googer - http://python.net/~goodger/projects/pycon/2007/idiomatic/handout.html#advanced-string-formatting
- Python Idioms, Safe Hammad - http://safehammad.com/downloads/python-idioms-2014-01-16.pdf
- Powerful Python - https://powerfulpython.com/blog/favorite-underused-python-idiom/



```
''.join(['G', 'R', 'A', 'C', 'I', 'A', 'S', '!'])
```