

Python Avanzado



python SOFTWARE
FOUNDATION



HOLA!

Soy Nikos. Tengo 42 años. Llevo más de 10 años trabajando en tech.

Soy un sysadmin que evoluciono a DevOps.

Pueden seguirme en:

[Linkedin](#) o [Twitter](#)

Como estan?

¡Ahora es su turno!
Tomemos un momento para conocernos.



¿Qué es Python?

Es un lenguaje de programación, multiparadigma y multinivel, con soporte en programación orientada a objetos, imperativa y funcional.. También es un lenguaje de propósito general, lo que significa que se puede utilizar para una amplia variedad de tareas

Python es utilizado por grandes empresas como Google, Facebook, Netflix, Spotify y Dropbox, lo que demuestra su versatilidad y capacidad para manejar grandes cantidades de datos y aplicaciones complejas.

¿Por qué python?.

Lenguaje
interpretado
de alto nivel

El Zen de python

Multipropósito

IA, ML, data science,
finanzas, RPA, web,
gaming, robotica, big
data, IoT, scraping,
automatizacion.

Bibliotecas y
Frameworks

Enorme cantidad, en
mejora y expansión.

¿Por qué python?

Comunidad activa y amplia

Siempre hay soporte y recursos disponibles en línea. Ya que posee una comunidad enorme y activa.

Portabilidad

Múltiples plataformas

Fácil integración

Se puede integrar fácilmente con otros lenguajes:
C, java, .net, js...

¿Por qué python?

Open-source y
gratuito

No se requiere una inversión monetaria significativa para comenzar a usarlo.

Fácil
aprendizaje:

Ideal para principiantes y para aquellos que quieren aprender a programar de manera efectiva.



¿Qué es el ZEN de python?

Entendiendo sus valores centrales.

¿Qué es el ZEN de python?

Bello es mejor que feo.
Explícito es mejor que implícito.
Simple es mejor que complejo.

Complejo es mejor que complicado.
Plano es mejor que anidado.
Disperso es mejor que denso.

La legibilidad cuenta.
Los casos especiales no son tan especiales como para
quebrantar las reglas.
Aunque lo práctico gana a la pureza.
Los errores nunca deberían dejarse pasar
silenciosamente.

A menos que hayan sido silenciados explícitamente.
Frente a la ambigüedad, rechaza la tentación de
adivinar.
Debería haber una —y preferiblemente sólo una—
manera obvia de hacerlo.

Aunque esa manera puede no ser obvia al principio a
menos que usted sea holandés (Guido van Rossum).
Ahora es mejor que nunca.
Aunque nunca es a menudo mejor que ya mismo.

Si la implementación es difícil de explicar, es una mala
idea.
Si la implementación es fácil de explicar, puede que
sea una buena idea.
Los "namespaces" son una gran idea ¡Hagamos más
de esas cosas!.

Bello es mejor que feo

```
# 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'
```

Explícito es mejor que implícito

```
# version implicita
def mts_in2(m):
    return m * 39.3701 * 2

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


Simple es mejor que complejo

```
# Simple
numeros = [1,2,3,4,5,6,7,8,9]
for numero in numeros:
    print(numero)

# Complejo
a = [1,2,3,4,5,6,7,8,9]
for x in range(len(a)):
    print(a[x])
```

Complejo es mejor que complicado

```
import datetime as dt

# version complejo
int(numpy.ceil((end_date - start_date).days + (end_date - start_date).seconds / 86400 )) if
    isinstance(start_date:dt.datetime) and isinstance(end_date:dt.datetime) else 0

# version Simple
def obtener_dias(start_date:dt.datetime, end_date:dt.datetime)->int:
    if isinstance(start_date:dt.datetime) and isinstance(end_date:dt.datetime):
        diferencia_fechas = end_date - start_date
        segundos_en_dia = 86400
        dias = diferencia_fechas.days + diferencia_fechas.seconds / segundos_en_dia
        return int(numpy.ceil(dias))
    else:
        return 0

obtener_dias(start_date, end_date)
```

Simple > Complejo > Complicado

Plano es mejor que anidado

```
# version anidada
valores_ajustados = []
for s in sistemas:
    sensores = sensores_sistema(s)
    for sensor in sensores:
        valores_s = valores_sensor(sensor)
        for val in valores_s:
            valores_ajustados.append(ajustar_valor(val))

# version aplanada pero densa
valores_ajustados = [ajustar_valor(val) for val in valores_sensor(sensor) for sensor in sensores_sistema(s) for s in sistemas]

# version en funciones simples, testeables y escalables
def valores_del_sistema_ajustados(sistemas):
    for sistema in sistemas:
        yield valores_sistema(sistema)

! usage
def valores_sistema(sistema):
    for sensores in sistema:
        yield valores_sensores(sensores)

! usage
def valores_sensores(sensores):
    for sensor in sensores:
        yield valores_sensor_ajustados(sensor)

! usage
def valores_sensor_ajustados(sensor):
    for valor in valores_sensor(sensor):
        yield ajustar_valor(valor)
```


Disperso es mejor que denso

```
# version densa
return i**2 if i > 0 else 0 if i==0 else 2 * i

# version dispersa
if i > 0:
    return i**2
elif i == 0:
    return 0
else:
    return 2 * i

# version aceptada
if i > 0:
    return i**2
else:
    return 0 if i==0 else 2 * i
```

La legibilidad cuenta

No tan legible 1

```
def f(x):  
    y=3*x+1  
    return y
```

Más legible 1

```
def f(x):  
    y = 3 * x + 1  
    return y
```

Esta es fácil: D. Compare C y Python:

```
#include <stdio.h> int main(void) {  
    printf("Hello, world!\n"); return(0); }
```

vs

```
print "Hello world!"
```

¿Y qué hay de la sangría? El código bien sangrado es más legible. Por lo tanto, en Python es obligatorio.

No tan legible 2

```
for i in range(10):print(i)
```

Más legible 2

```
for i in range(10):  
    print(i)
```

Los errores nunca deberían de ocurrir silenciosamente

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


A no ser que se silencien explícitamente

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

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.

Los namespaces son una buena idea. ¡usemos más de ellos!

```
# Global Scope
x = 0

def outer():
    # Enclosed Scope
    y = 1
    x = 2
    print("outer x:", x) # Output: 2
    print("outer y:", y) # Output: 1
    def inner():
        # Local Scope
        z = 3
        x = 4
        print("inner x:", x) # Output: 4
        print("inner y:", y) # Output: 1
        print("inner z:", z) # Output: 3
    inner()

outer()
print("global x:", x) # Output: 0
```

Cada función define su propio ámbito local para las variables, y cuando se hace referencia a una variable dentro de una función, Python primero busca en el ámbito local, luego en cualquier ámbito contenedor (como el ámbito de la función exterior) y, finalmente, en el ámbito global si no se encuentra en ningún ámbito anterior.

El resto de los axiomas para reflexionar!

Los casos especiales no son tan especiales como para quebrantar las reglas.

Aunque lo práctico gana a la pureza.

Frente a la ambigüedad, rechaza la tentación de adivinar.

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

Aunque esa manera puede no ser obvia al principio a menos que usted sea holandés. Referencia al creador de Python, Guido van Rossum (Holandes).

Ahora es mejor que nunca.

Aunque nunca es a menudo mejor que ya mismo.

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

Si la implementación es fácil de explicar, puede que sea una buena idea.



Librerías y paquetes!

Conoce las bibliotecas esenciales de Python para cualquier proyecto

Ciencia de Datos!

NumPy

Librería para el cálculo numérico en Python.

Pandas

Librería para el manejo y análisis de datos en Python.

Matplotlib

Librería para la visualización de datos en Python.

Machine Learning!

Scikit-Learn

Librería para el aprendizaje automático en Python.

TensorFlow

Plataforma para el aprendizaje automático y el procesamiento de datos en Python.

PyTorch

Biblioteca de aprendizaje profundo y tensor en Python.

Desarrollo Web!

Flask

Microframework para el desarrollo web en Python.

Django

Framework de alto nivel para el desarrollo web en Python.

SQLAlchemy

Librería ORM para el manejo de bases de datos en Python.

Automatización y Sistemas

Fabric

Biblioteca para la automatización de tareas en Python.

Ansible

Herramienta de automatización y gestión de configuraciones en Python.

Requests

Librería para la comunicación con APIs y sitios web en Python.

Contenidos del curso

01

Gestión de dependencias
y workspaces

Intérpretes, VENV, docker

02

Collections

Chainmap, Counter,
OrderedDict, defaultdict

03

Decorators

Cómo personalizar una
función.

04

Flask
API rest

05

Python 2 vs Python 3

You can describe the topic
of the section here

06

Próximos desafíos

Documentation, logging,
testing, frontend/django,

Me encuentran en:

nlastra@grupoloyal.com



LinkedIn o Twitter

Gracias!

Hora de comenzar

Instalación

Abordar contenidos y
ejercicios

interprete

Curso

IDE

PyCharm o VSCode

Examen

Viernes salida a
produccion

