Formatters en Python

Mateo Cumbal

2024-11-30

Tabla de Contenidos

1 Código Original

El código proporcionado presenta varias malas prácticas de estilo de programación y convenciones de Python:

1. Falta de indentación consistente

2. Líneas demasiado largas

3. Espacios alrededor de operadores

4. Uso inconsistente de paréntesis

5. No separación entre bloques de código

```
import math
import sympy
import matplotlib.pyplot as plt

def cubic_spline(xs: list[float], ys: list[float]) -> list[sympy.Symbol]:
  points = zip(xs,ys)
  n = len(xs)-1
  h=[xs[i+1]-xs[i] for i in range(n)]
  alpha=[0]*(n)
  for i in range(1, n): alpha[i]=(3/h[i])* (ys[i+1]-ys[i]) - (3/h[i-1])*(ys[i]-ys[i-1])
  l=[1]+[0]*n
  u=[0]*n
  z=[0]*(n+1)
  for i in range(1,n):
    l[i]=2*(xs[i+1]-xs[i-1])-h[i-1]*u[i-1]
    u[i]=h[i]/l[i]
    z[i]=(alpha[i]-h[i-1]*z[i-1])/l[i]
  l[n]=1
  z[n]=0
  c=[0]*(n+1)
  b=[0]*n
  d=[0]*n
  a=[ys[i] for i in range(n)]
  x=sympy.Symbol('x')
  splines=[]
  for j in range(n-1,-1,-1):
    c[j]=z[j]-u[j]*c[j+1]
    b[j]=(ys[j+1]-ys[j])/h[j]-h[j]*(c[j+1]+2*c[j])/3
    d[j]=(c[j+1]-c[j])/(3*h[j])
    a[j]=ys[j]
    S=(a[j]+b[j]*(x-xs[j])+c[j]*(x-xs[j])**2+d[j]*(x-xs[j])**3)
    splines.append(S)
  splines.reverse()
  return splines
```

2 Black

Black es un formateador de código “opinado”, lo que significa que toma decisiones de formato por ti. Su principal objetivo es hacer que el código sea consistente sin necesidad de configuraciones adicionales. Black se encarga de que el código siga un conjunto de **reglas estrictas** y lo formatea de una forma predeterminada.

- Líneas con un máximo de 88 caracteres.
- Añade espacios alrededor de operadores binarios.
- Prefiere comillas dobles para las cadenas.
- Usa una indentación de 4 espacios.
- Proporciona un estilo muy rígido, eliminando cualquier ambigüedad sobre el formato.

```
import math
import sympy
import matplotlib.pyplot as plt


def cubic_spline(xs: list[float], ys: list[float]) -> list[sympy.Symbol]:
    points = zip(xs, ys)
    n = len(xs) - 1
    h = [xs[i + 1] - xs[i] for i in range(n)]
    alpha = [0] * (n)
    for i in range(1, n):
        alpha[i] = (3 / h[i]) * (ys[i + 1] - ys[i]) - (3 / h[i - 1]) * (
            ys[i] - ys[i - 1]
        )
    l = [1] + [0] * n
    u = [0] * n
    z = [0] * (n + 1)
    for i in range(1, n):
        l[i] = 2 * (xs[i + 1] - xs[i - 1]) - h[i - 1] * u[i - 1]
        u[i] = h[i] / l[i]
        z[i] = (alpha[i] - h[i - 1] * z[i - 1]) / l[i]
    l[n] = 1
    z[n] = 0
    c = [0] * (n + 1)
    b = [0] * n
    d = [0] * n
    a = [ys[i] for i in range(n)]
    x = sympy.Symbol("x")
    splines = []
    for j in range(n - 1, -1, -1):
        c[j] = z[j] - u[j] * c[j + 1]
        b[j] = (ys[j + 1] - ys[j]) / h[j] - h[j] * (c[j + 1] + 2 * c[j]) / 3
        d[j] = (c[j + 1] - c[j]) / (3 * h[j])
        a[j] = ys[j]
```

```
        S = (
            a[j]
            + b[j] * (x - xs[j])
            + c[j] * (x - xs[j]) ** 2
            + d[j] * (x - xs[j]) ** 3
        )
        splines.append(S)
    splines.reverse()
    return splines
```

3 Autopep8

Autopep8 es un formateador automático que ajusta el código según las recomendaciones de PEP 8. Es un formateador más flexible que Black, ya que no impone tantas reglas y permite configuraciones.

- Aplica las reglas de PEP 8, pero de manera menos estricta que Black.
- Corrige los errores más comunes como líneas demasiado largas, indentación incorrecta, entre otros.
- Permite configuraciones personalizadas a través de parámetros.

```
import math
import sympy
import matplotlib.pyplot as plt


def cubic_spline(xs: list[float], ys: list[float]) -> list[sympy.Symbol]:
    points = zip(xs, ys)
    n = len(xs)-1
    h = [xs[i+1]-xs[i] for i in range(n)]
    alpha = [0]*(n)
    for i in range(1, n):
        alpha[i] = (3/h[i]) * (ys[i+1]-ys[i]) - (3/h[i-1])*(ys[i]-ys[i-1])
    l = [1]+[0]*n
    u = [0]*n
    z = [0]*(n+1)
    for i in range(1, n):
        l[i] = 2*(xs[i+1]-xs[i-1])-h[i-1]*u[i-1]
        u[i] = h[i]/l[i]
        z[i] = (alpha[i]-h[i-1]*z[i-1])/l[i]
```

```
    l[n] = 1
    z[n] = 0
    c = [0]*(n+1)
    b = [0]*n
    d = [0]*n
    a = [ys[i] for i in range(n)]
    x = sympy.Symbol('x')
    splines = []
    for j in range(n-1, -1, -1):
        c[j] = z[j]-u[j]*c[j+1]
        b[j] = (ys[j+1]-ys[j])/h[j]-h[j]*(c[j+1]+2*c[j])/3
        d[j] = (c[j+1]-c[j])/(3*h[j])
        a[j] = ys[j]
        S = (a[j]+b[j]*(x-xs[j])+c[j]*(x-xs[j])**2+d[j]*(x-xs[j])**3)
        splines.append(S)
    splines.reverse()
    return splines
```

4 Yapf

YAPF (Yet Another Python Formatter) trata de formatear el código de manera similar a cómo lo haría un programador que sigue las recomendaciones de PEP 8. Se basa en una versión modificada de las reglas de PEP 8.

- Similar a Autopep8 pero con más flexibilidad en cuanto al estilo general.
- Intenta mantener el formato original tanto como sea posible, pero corrige las violaciones de PEP 8.
- Se puede configurar para que siga reglas específicas más allá de PEP 8.

```
import math
import sympy
import matplotlib.pyplot as plt


def cubic_spline(xs: list[float], ys: list[float]) -> list[sympy.Symbol]:
    points = zip(xs, ys)
    n = len(xs) - 1
    h = [xs[i + 1] - xs[i] for i in range(n)]
    alpha = [0] * (n)
    for i in range(1, n):
```

```
        alpha[i] = (3 / h[i]) * (ys[i + 1] -
                                 ys[i]) - (3 / h[i - 1]) * (ys[i] - ys[i - 1])
    l = [1] + [0] * n
    u = [0] * n
    z = [0] * (n + 1)
    for i in range(1, n):
        l[i] = 2 * (xs[i + 1] - xs[i - 1]) - h[i - 1] * u[i - 1]
        u[i] = h[i] / l[i]
        z[i] = (alpha[i] - h[i - 1] * z[i - 1]) / l[i]
    l[n] = 1
    z[n] = 0
    c = [0] * (n + 1)
    b = [0] * n
    d = [0] * n
    a = [ys[i] for i in range(n)]
    x = sympy.Symbol('x')
    splines = []
    for j in range(n - 1, -1, -1):
        c[j] = z[j] - u[j] * c[j + 1]
        b[j] = (ys[j + 1] - ys[j]) / h[j] - h[j] * (c[j + 1] + 2 * c[j]) / 3
        d[j] = (c[j + 1] - c[j]) / (3 * h[j])
        a[j] = ys[j]
        S = (a[j] + b[j] * (x - xs[j]) + c[j] * (x - xs[j])**2 + d[j] *
             (x - xs[j])**3)
        splines.append(S)
    splines.reverse()
    return splines
```

5 Ruff

Ruff es un formateador basado en Flake8 y optimizado para ser muy rápido. Además de formatear el código, puede detectar errores de estilo, posibles problemas de código y mal uso de las convenciones.

- Muy rápido y eficiente, adecuado para proyectos grandes.
- Está diseñado no solo para formatear, sino también para detectar problemas de estilo y errores comunes.
- Se basa en Flake8, lo que le permite ser más analítico sobre el código.

```
import sympy as sym
from IPython.display import display


def cubic_spline(xs: list[float], ys: list[float]) -> list[sym.Symbol]:
    points = sorted(zip(xs, ys), key=lambda x: x[0])  # Sort points by x
    xs = [x for x, _ in points]
    ys = [y for _, y in points]

    n = len(points) - 1
    h = [xs[i + 1] - xs[i] for i in range(n)]

    alpha = [0] * (n)
    for i in range(1, n):
        alpha[i] = 3 / h[i] * (ys[i + 1] - ys[i]) - 3 / h[i - 1] * (ys[i] - ys[i - 1])

    l = [1] + [0] * n
    u = [0] * n
    z = [0] * (n + 1)

    for i in range(1, n):
        l[i] = 2 * (xs[i + 1] - xs[i - 1]) - h[i - 1] * u[i - 1]
        u[i] = h[i] / l[i]
        z[i] = (alpha[i] - h[i - 1] * z[i - 1]) / l[i]

    l[n] = 1
    z[n] = 0

    c = [0] * (n + 1)
    b = [0] * n
    d = [0] * n
    a = [ys[i] for i in range(n)]

    x = sym.Symbol("x")
    splines = []

    for j in range(n - 1, -1, -1):
        c[j] = z[j] - u[j] * c[j + 1]
        b[j] = (ys[j + 1] - ys[j]) / h[j] - h[j] * (c[j + 1] + 2 * c[j]) / 3
        d[j] = (c[j + 1] - c[j]) / (3 * h[j])
        a[j] = ys[j]
```

```
        S = (
            a[j]
            + b[j] * (x - xs[j])
            + c[j] * (x - xs[j]) ** 2
            + d[j] * (x - xs[j]) ** 3
        )
        splines.append(S)

    splines.reverse()
    return splines
```

6 Comparación Final

Característica	**Black**	**Autopep8**	**YAPF**	**Ruff**
Objetivo	Formateo estricto y automático.	Cumple con PEP 8, pero es flexible.	Basado en PEP 8, pero flexible.	Formatea y detecta errores.
Reglas	Muy estrictas, pocas configuraciones.	Flexible, permite personalización.	Flexible, pero sigue PEP 8.	Estricto, con corrección de errores.
Longitud de líneas	Máximo de 88 caracteres.	Siguiendo PEP 8.	Depende de la configuración.	No especifica, similar a PEP 8.
Configuración	No se requiere, es todo automático.	Configurable, permite ajustes.	Configurable.	Configurable, con enfoque en errores.
Flexibilidad	Baja, tiene su propio estilo.	Alta, permite muchas configuraciones.	Alta, pero sigue más de cerca PEP 8.	Baja en formato, pero flexible en detección de errores.
Rendimiento	Muy rápido, optimizado.	Rápido.	Moderado.	Muy rápido, optimizado.