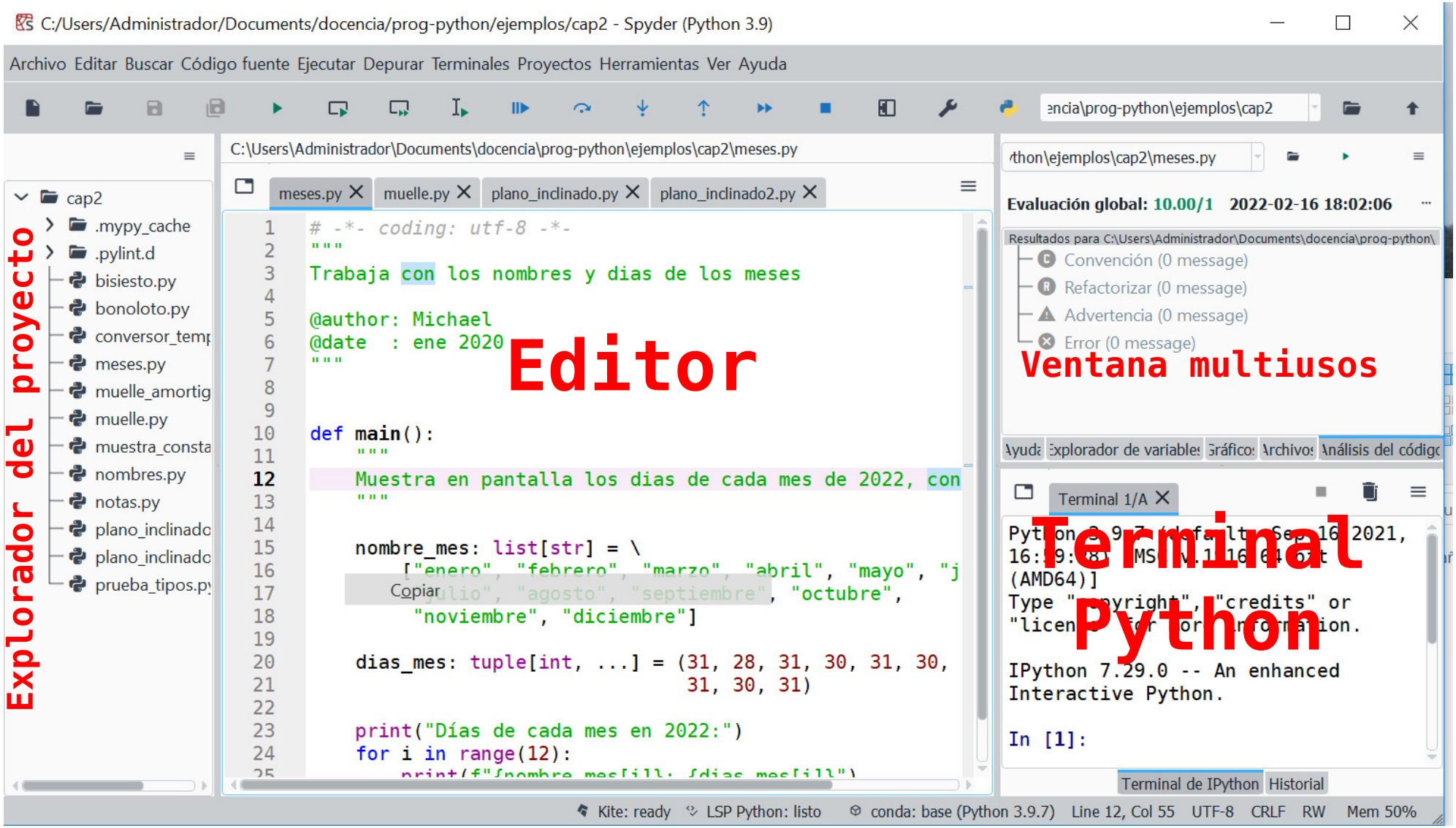
Bloque II. Herramientas

Capítulo 10. *Uso de un entorno integrado de desarrollo de programas*

- Entorno de desarrollo de programas ***spyder***
- Gestión de proyectos
- Analizar, cargar y ejecutar el programa
- La depuración
- Generación de documentos

10.1 Entorno de desarrollo **spyder**




```
# -*- coding: utf-8 -*-
"""
Trabaja con los nombres y dias de los meses

@author: Michael
@date  : ene 2020
"""


def main():
    """
    Muestra en pantalla los dias de cada mes de 2022, con
    """

    nombre_mes: list[str] = \
        ["enero", "febrero", "marzo", "abril", "mayo", "j
         "julio", "agosto", "septiembre", "octubre",
         "noviembre", "diciembre"]

    dias_mes: tuple[int, ...] = (31, 28, 31, 30, 31, 30,
                                 31, 30, 31)

    print("Días de cada mes en 2022:")
    for i in range(12):
        print(f"{nombre_mes[i]: {dias_mes[i]}")
```

10.2. Gestión de proyectos

Para programar utilizando ***spyder*** es conveniente crearse un "***proyecto***" por cada programa o práctica

El proyecto es un directorio en el disco donde se guarda toda la información que se va generando relacionada con el programa:

- código fuente
- módulos compilados
- documentación
- imágenes
- etc.

Creación de un nuevo proyecto

Elegir `Proyectos=>Nuevo Proyecto`, y luego elegir el directorio deseado y darle al proyecto un nombre. Ej:

```
practica2
```

Añadir un nuevo módulo al proyecto

- Para ello, con *botón derecho* sobre el nombre del proyecto seleccionar:

```
Crear Nuevo -> Archivo de Python
```

- Luego elegir el directorio del proyecto y darle un nombre

Para abrir el módulo para editarlo, hacer "doble click" sobre él en el explorador del proyecto

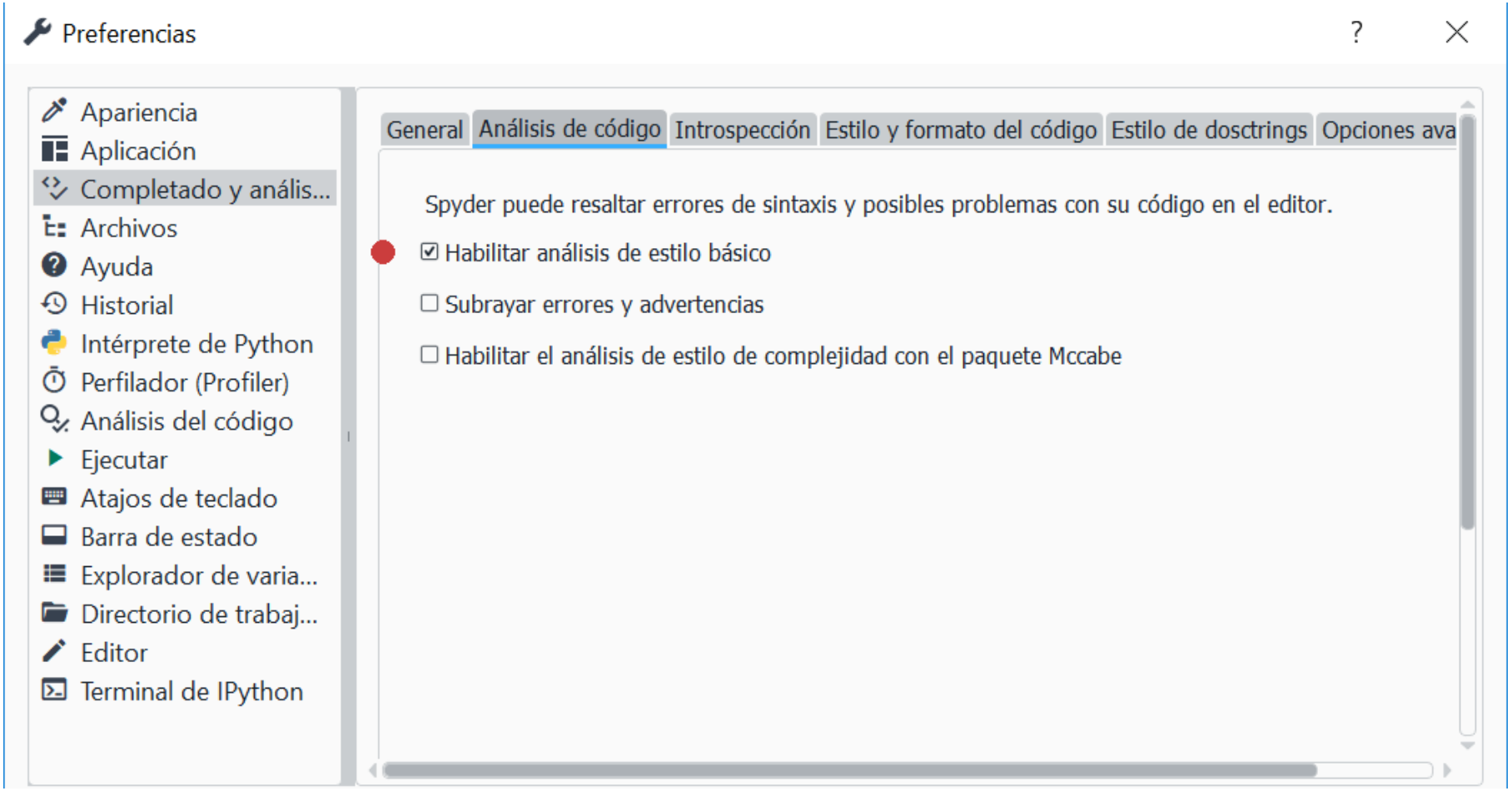
Ejercicio 1: editar este programa

```
# -*- coding: utf-8 -*-
"""
Pone un mensaje de bienvenida en pantalla

@author: <tu nombre>
@date  : feb/2023
"""

def main():
    """
    Hola Mundo

    Este programa pone un mensaje en pantalla
    """

    print("Hola, ¿qué tal?")
```

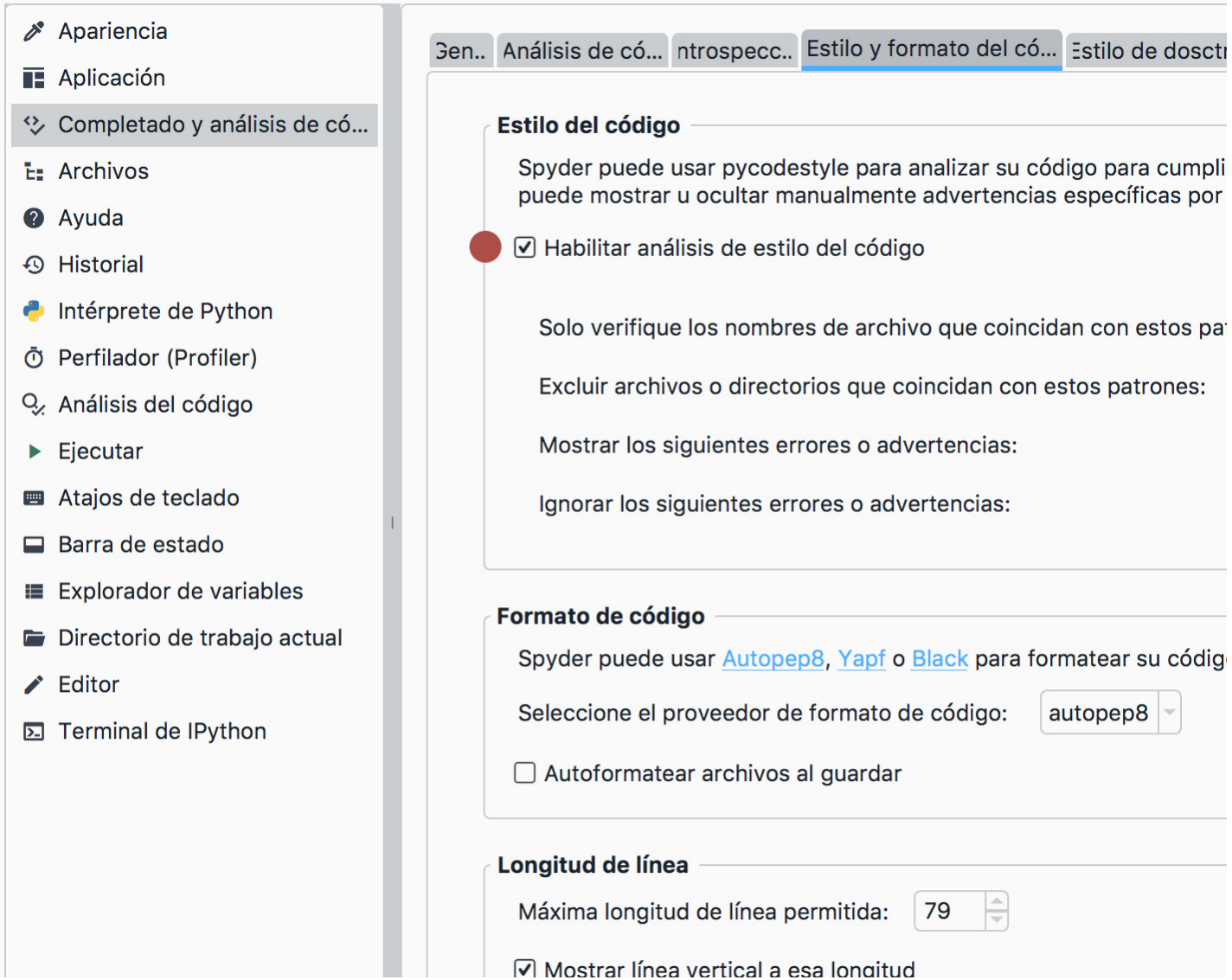
10.3 Analizar, cargar y ejecutar

Analizar: el editor va mostrando errores y avisos al escribir

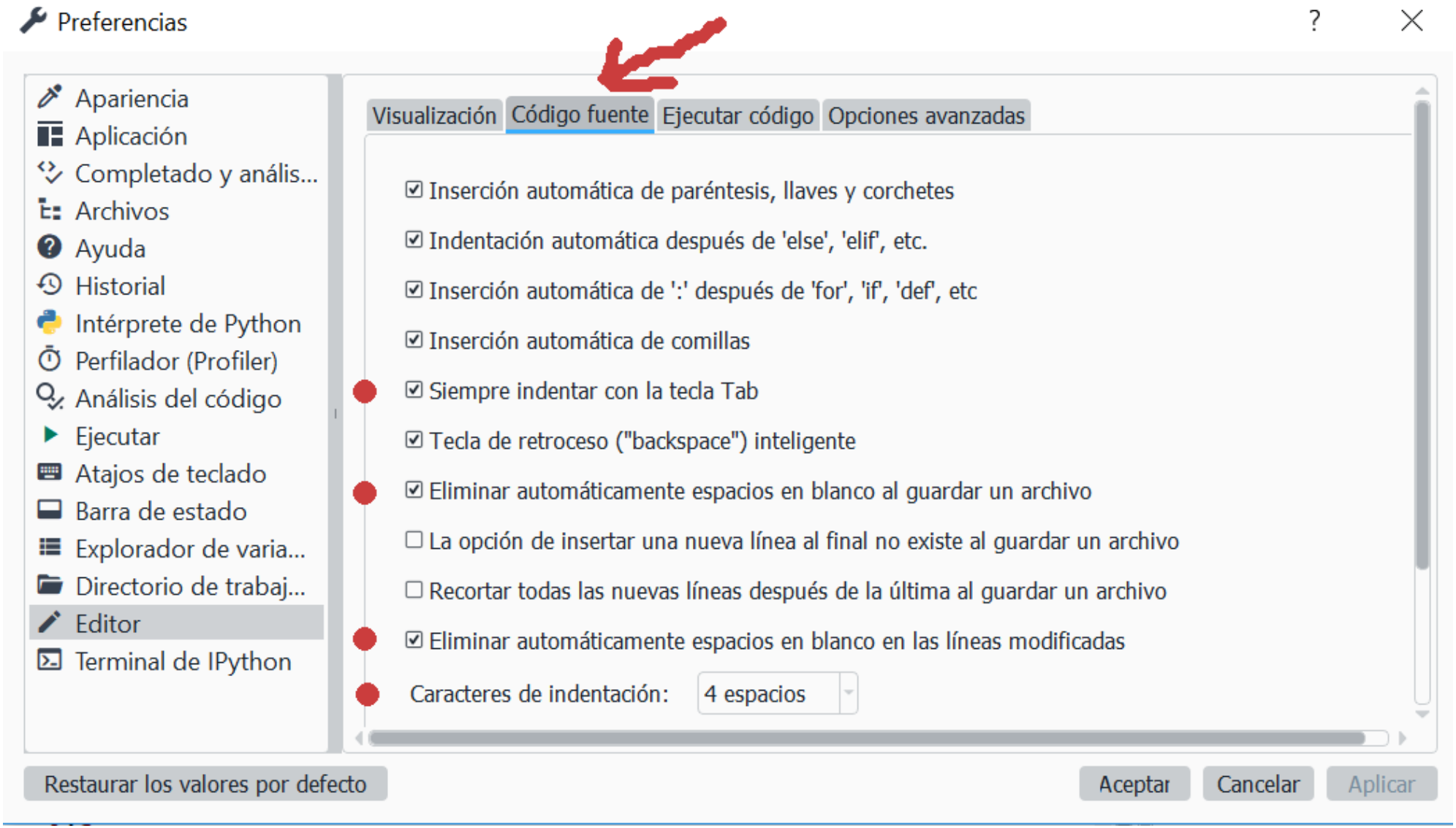
- habilitar la detección de fallos de estilo básico en las preferencias



Análisis de estilo (cont.)

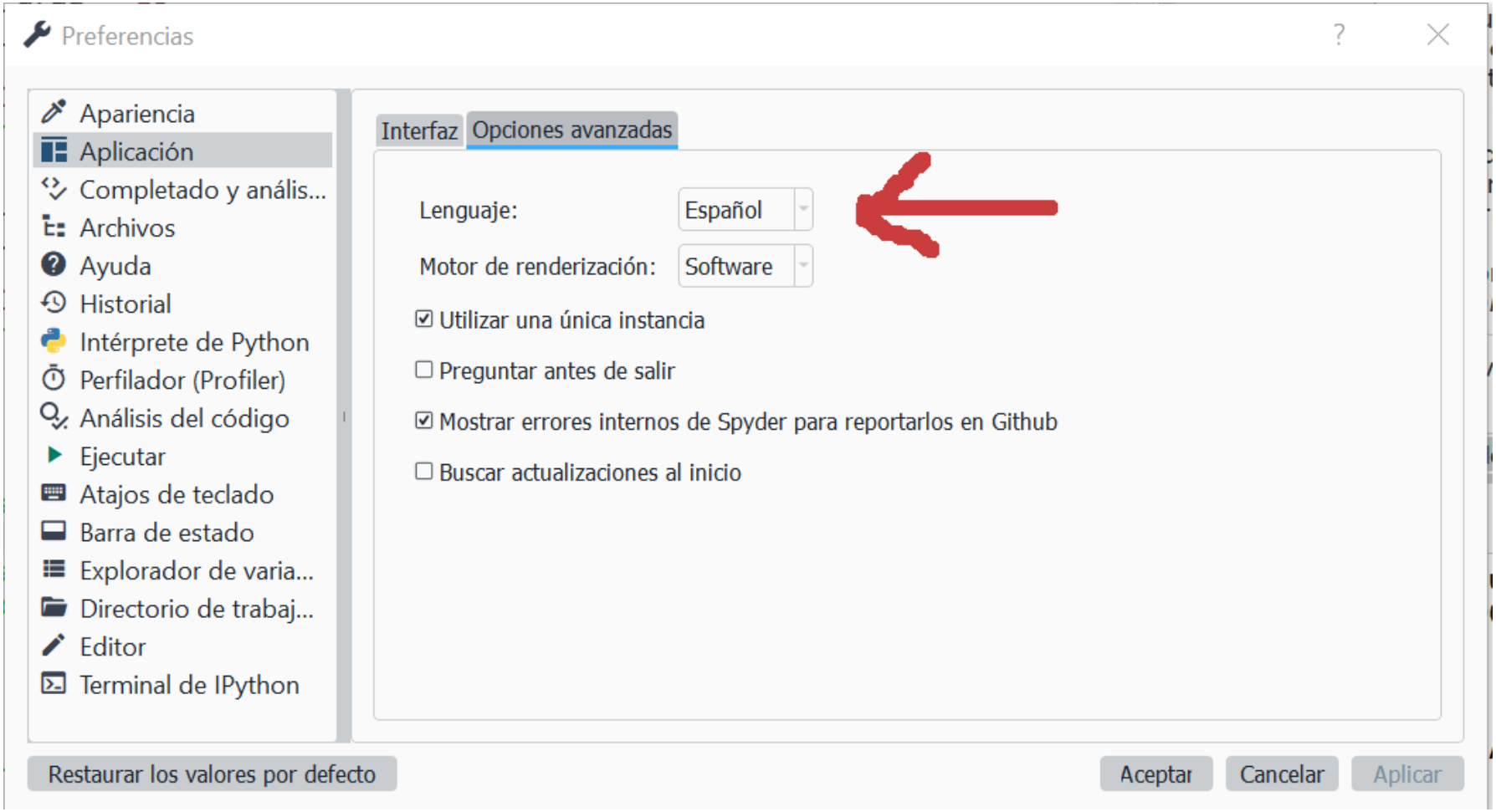


Otras opciones recomendadas



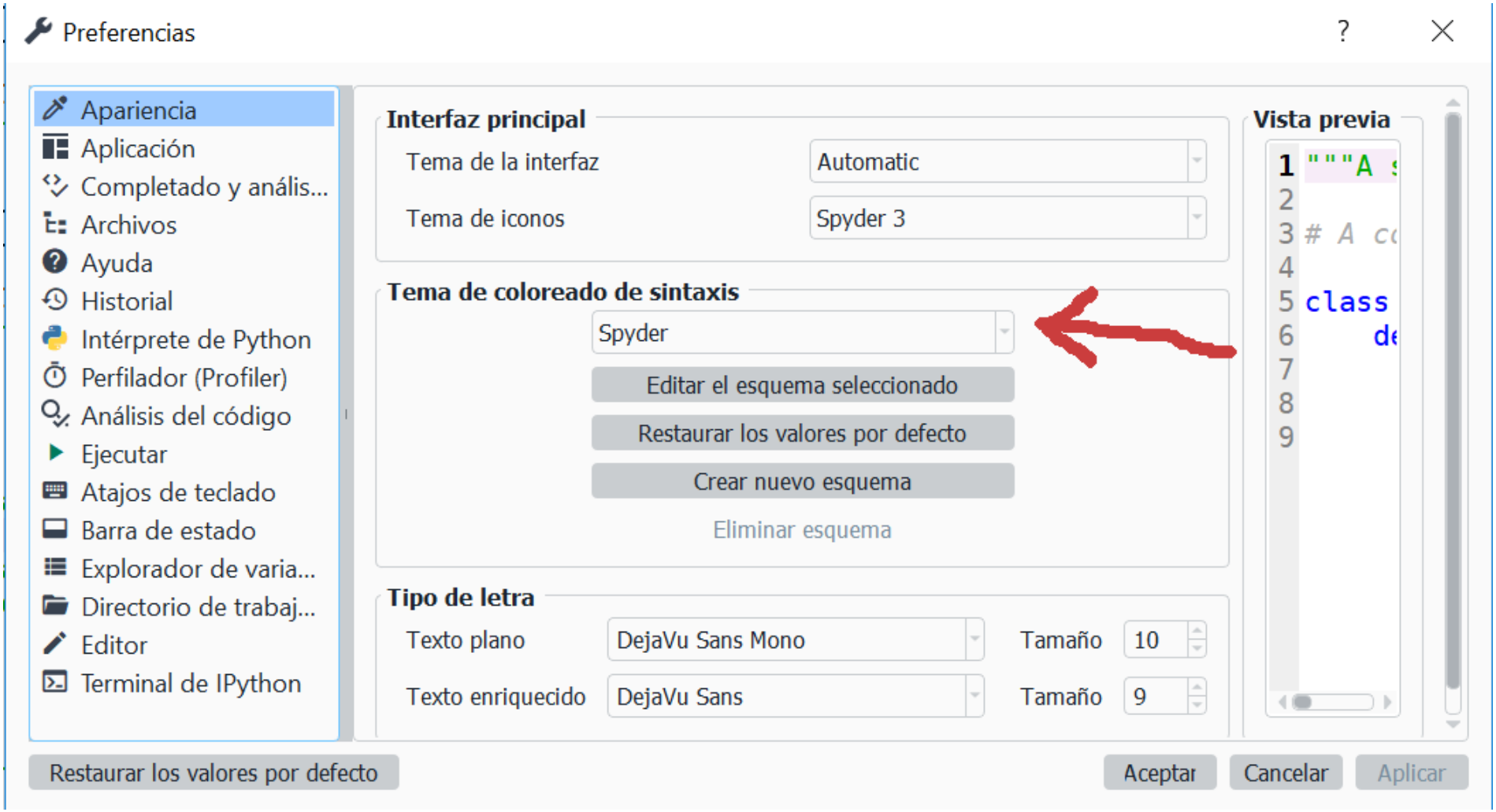
Otras opciones recomendadas (cont.)

Podemos configurar el idioma español si lo deseamos



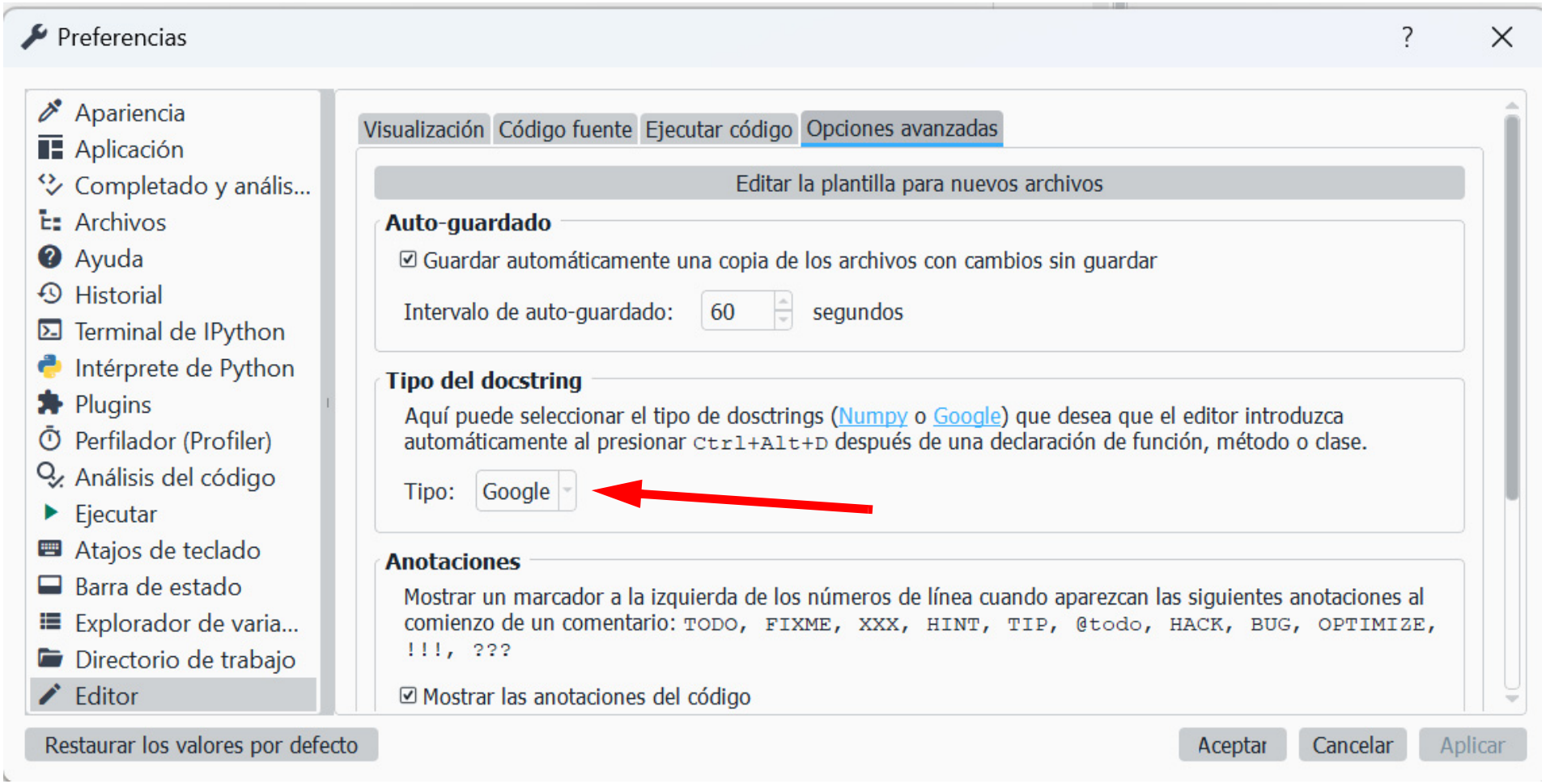
Otras opciones recomendadas (cont.)

Podemos elegir el fondo blanco o negro



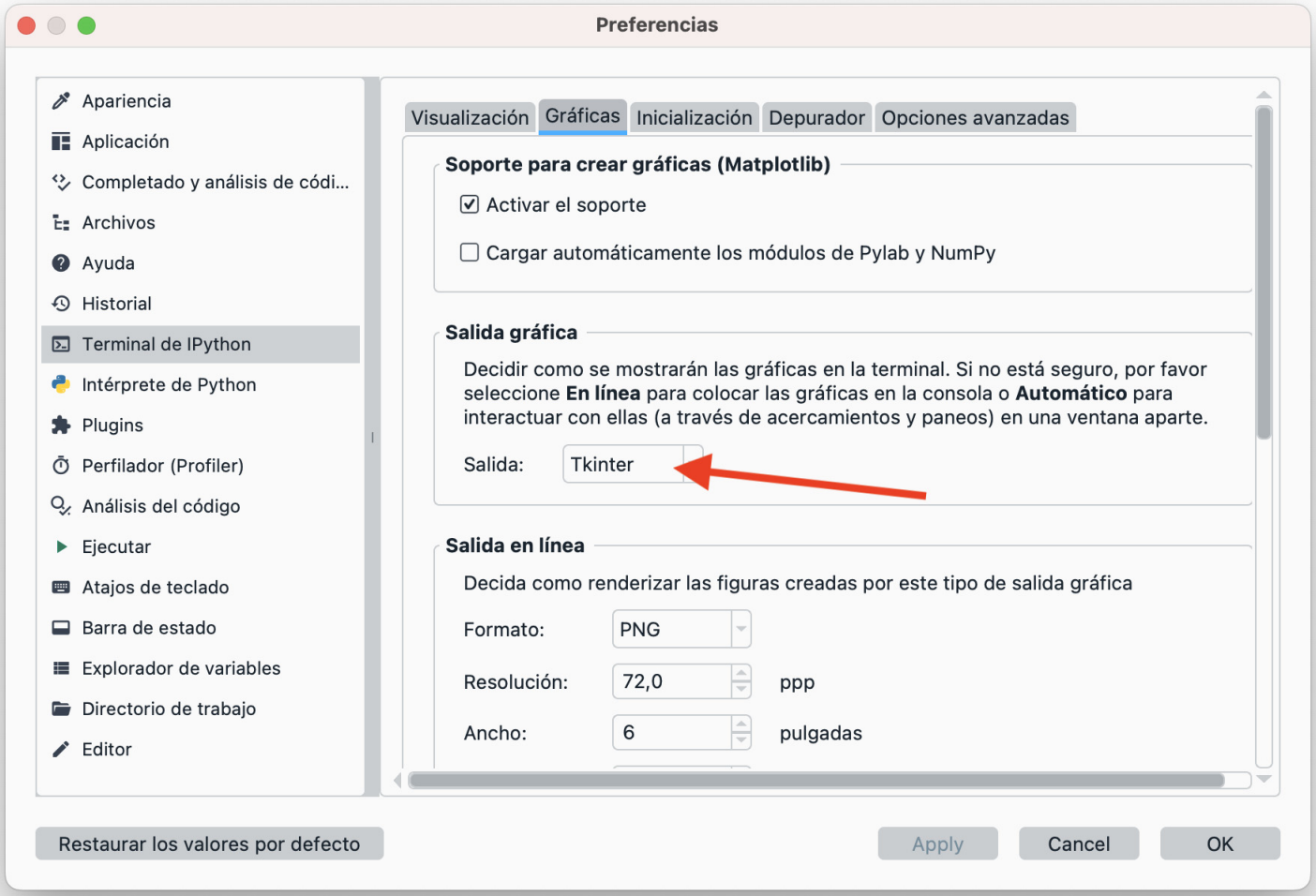
Otras opciones recomendadas (cont.)

Elegir el estilo de docstring



Otras opciones recomendadas (cont.)

Para sistemas Mac y Linux, configurar las gráficas



Otras opciones recomendadas (cont.)

Instalar `mypy` para detección de errores de tipificación

- Ver penúltima página del apéndice en el capítulo 2

Para sistemas Mac y Linux es recomendable instalar la integración de `mypy` y `Spyder`, con objeto de ver los errores de tipificación sobre la marcha

- Ver última página del apéndice en el capítulo 2

Cargar y ejecutar

Análisis avanzado del código: Pulsar F8

- es muy importante acordarse de usarlo

Para cargar el programa en el intérprete: Pulsar F5 o el botón 

- las instrucciones sueltas se ejecutan al cargar
- pero las funciones o clases no

Para ejecutar un `main()`, teclear desde el intérprete:

```
main()
```

Para ejecutar una función, teclear desde el intérprete:

```
funcion(parámetros)
```

Ejercicio 2: simulación de un muelle

- Utilizando el navegador de archivos, meter en la carpeta del proyecto el ejemplo de clase `muelle.py` que se suministra
- Analizarlo, cargarlo y ejecutarlo

10.4. La depuración

Es el proceso de prueba del programa para localizar errores

La depuración se puede realizar:

- manualmente: insertando instrucciones de salida (`print`) que muestren el flujo de control del programa y el valor de las variables de interés
- mediante un depurador de alto nivel

El depurador de alto nivel permite:

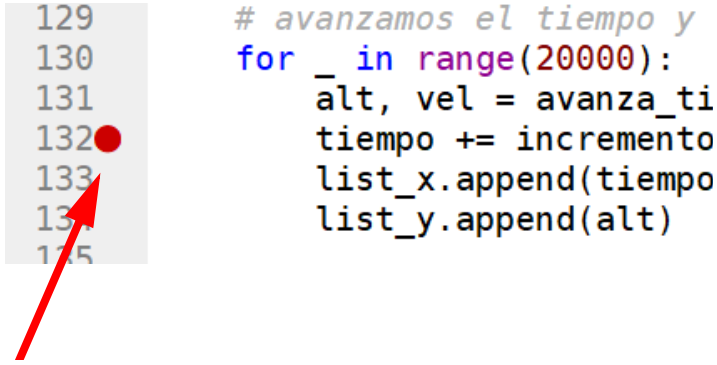
- parar el programa en los puntos deseados: *puntos de ruptura* o *breakpoints*
- ejecutar paso a paso
- visualizar el contenido de las variables
- visualizar secuencia de llamadas a funciones y sus argumentos

Depurador de Spyder

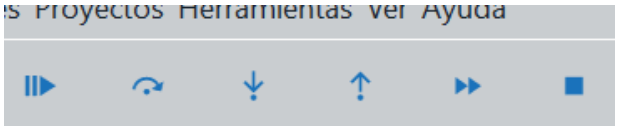
Es un depurador para programas Python con interfaz gráfica

Permite introducir puntos de ruptura (*breakpoints*)

- "click" en la columna a la derecha número de línea (aparece un ●)
- se quita también con "click"



En el menú superior se dispone de los siguientes botones de depuración



Depuración: Comienzo

Para comenzar la depuración el programa debe tener instrucciones que ejecuten el `main()`

- si no, el depurador no se ejecutará

Para ello, en la ventana del editor poner una invocación al `main()` al *final del módulo*. Por ejemplo:

```
def main():
    """
    Hola Mundo
    """
    print("Hola, ¿qué tal?")

main()
```

Añadir para depurar.
Luego quitar,
o proteger como se indica en el capítulo 4

Botones de depuración

	Comenzar la depuración. Se detiene en la primera instrucción o en el primer punto de ruptura
	Ejecutar línea seleccionada[a]
	Entrar en la función de la línea actual[b]
	Ejecutar hasta que la función actual termine[c]
	Continuar la ejecución hasta el próximo punto de ruptura[d]
	Finalizar la depuración[e]

a. Equivale al comando `n` o `next` en la consola del depurador `(ipdb>)`; para repetir el comando, presionar '*enter*'
b. Equivale al comando `s` o `step`
c. Equivale al comando `r` o `return`
d. Equivale al comando `c` o `continue`
e. Equivale al comando `q`, `quit` o `exit`

Visualizar información

Variables: en la ventana multiusos, pestaña "*Explorador de variables*"

Con el programa detenido en una función se pueden teclear comandos en el terminal, tras el símbolo "`IPdb>`":

- Secuencia de llamadas a funciones:
  - `w` o `where`: muestra la secuencia, lo más antiguo arriba
- Argumentos y variables de funciones
  - `args`: muestra los argumentos
  - `up`: cambia a la llamada anterior (más antigua)
  - `down`: cambia a la llamada posterior (más nueva)

Ejercicio 3: depuración

Depurar el programa `muelle.py`

- ejecutar paso a paso
- añadir un punto de ruptura en el bucle
- continuar la ejecución hasta el punto de ruptura varias veces
  - observar cómo varían los valores de las variables y cómo crecen las listas
- quitar el punto de ruptura y continuar la ejecución hasta el final

10.5. Generación de documentos

Para usar un módulo con sus funciones y clases, lo único que se necesita conocer de él es la interfaz pública:

- clases: sus nombres, atributos y métodos
- funciones y métodos: sus cabeceras y descripción de lo que hacen

Se puede extraer esta información de manera automática, por medio de herramientas de documentación

- por ejemplo, `pydoc`

Ejecutar pydoc

Para ejecutar `pydoc` lo hacemos desde una shell del sistema operativo:

- En Windows, usar como shell `"Anaconda Prompt"`
- En Linux o Mac un `terminal`

El primer paso es situarse en la carpeta del proyecto

```
cd ruta_carpeta_del_proyecto
```

- Ejemplo: `cd C:\Users\pepe\Documents\tmp\Practica18`

El segundo paso es ejecutar `pydoc`

```
pydoc -w nombre_modulo
```

Observar que no se pone la extensión `".py"`

El resultado es un fichero `.html`. Se puede ver en un navegador web