Resumen

En esta Nota vamos a ver cómo usar el paquete minted. Se trata de un paquete muy potente, pero que tiene algunos pre-requisitos para poder hacer uso de él. Se explicarán las razones que han guiado la elaboración de la Nota, para a continuación describir la instalación de TeXLive, TeXStudio, Python y las aplicaciones de Python que demanda el paquete, todo ello bajo Windows, y finalmente ver ejemplos sencillos de uso.

Por lo que se verá en la Nota se recomienda el uso de TeXLive y la instalación de una versión de Python anterior a la 3.14.

Índice

1. Instalación y uso del paquete minted en TeXstudio

1.1. Introducción

El paquete minted [1]ofrece un resaltado de sintaxis mediante la biblioteca Pygments[2]. También ofrece opciones para personalizar la salida del código fuente resaltado, incluyendo funciones implementadas en Python, tales como la selección de fragmentos de código con expresiones regulares.

Antes de comenzar conviene mencionar que en versiones anteriores de minted para que funcionara debía de indicar al motor de proceso de documentos que para generar PDFs se "saliera" del programa y ejecutar comandos externos, usando -shell-escape, lo que es potencialmente peligroso[3]. Para solucionar este problema el creador del paquete, Geoffrey M. Poore[4], ha creado minted ver.3, que se supone que usa el ejecutable latexminted de Python dentro de la carpeta restricted/ (carpeta de ejecutables permitidos en modo seguro en la distribución de TeX), siendo el objetivo no tener que incluir -shell-escape en la configuración de órdenes de PdflaTeX, XeLaTeX o LuaLaTeX, lo que habilita a TeXstudio a salir del programa para ejecutar órdenes con comandos externos.

El propio autor del paquete señaló el 6 de marzo de 2025 que la versión 3 ya había estado disponible por varios meses y que ya estaba dentro de la lista de ejecutables confiables de la TeXLive, por lo que el método -shell-escape no era necesario, tal y como se puede ver en la figura 1.

La realidad es que a fecha de la elaboración de este documento, enero de 2026, se generan problemas de creación del PDF en el momento de intentar hacer funcionar el paquete. En mi caso instalé Python 3.14.1 y obviando el uso de -shell-escape genera problemas de generación del PDF, concretamente muestra el error TypeError: ArgParser.__init__()

[1] https://ctan.org/pkg/minted
[2] https://pygments.org/
[3] https://github.com/gpoore/minted/issues/166
[4] https://github.com/gpoore/minted

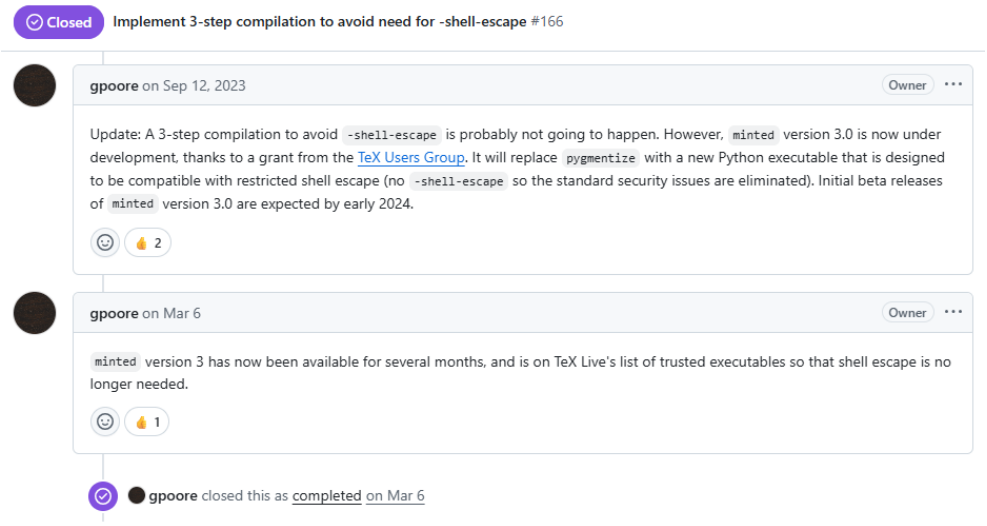

Figura 1: `minted ver.3` no necesitará más `-shell-escape`

`got an unexpected keyword argument 'color'` así como el comando `latexminted` genera un `TypeError`[5].

Como se comenta en el foro de Github se trata de un problema que se genera con las versiones de Python 3.14 en adelante, por lo que he optado por instalar una versión anterior, Python 3.11.9, así como cambiar la distribución de TeX de MikTex a TeXLive.

[5] `https://github.com/gpoore/minted/issues/463`

1.2. Instalación

1.2.1. Pre-requisito. Instalación de TeXLive

Los pasos a seguir son los siguientes:

Buscamos la página de TeXLive[6] y como podemos ver en la figura 2 buscamos **Ways to acquire TeXLive**, donde podemos elegir hacer una descarga de un archivo de instalación (para Windows en este caso), bajar una imagen ISO para crear un DVD (debido a su tamaño), solicitar un DVD ya creado, u otros métodos.

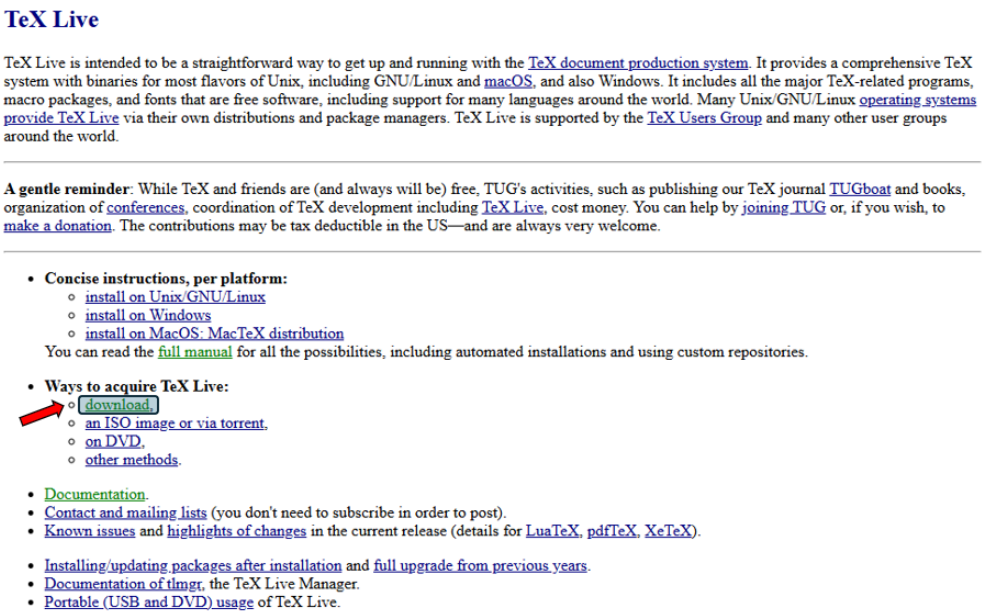

Figura 2: Página de TeXLive

Como indica la página, si se dispone de una conexión de Internet razonable, se recomienda la primera opción. Hacemos por tanto clic en `download`, lo que abre otra ventana, correspondiente con la figura 3.

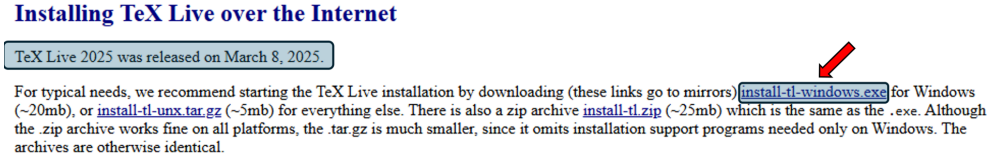

Figura 3: Opción de descarga de archivo de instalación (para Windows)

Hacemos clic en el archivo `.exe` y descargamos el ejecutable más actual, en este caso se corresponde con la versión del 8 de marzo de 2025[7], y procedemos a ejecutarlo, dando comienzo a la instalación de TeXLive. (Ver figura 4)

1.2.2. Pre-requisito. Instalación de TeXStudio

Buscamos en la página del programa TeXStudio[8] la versión más actual del programa para bajárnosla. El uso de este entorno de escritura no es obligatorio, pero se muestra por el ser el que utilizo de manera habitual.

[6] `https://www.tug.org/texlive/`

[7] Para ver las diferencias, pros y contras de MikTeX y TeXLive se puede ver lo comentado en `TEX.stackexchange.com` al respecto (`https://tex.stackexchange.com/questions/20036/what-are-the-advantages-of-tex-live-over-miktex`)

[8] `https://www.texstudio.org/#home`

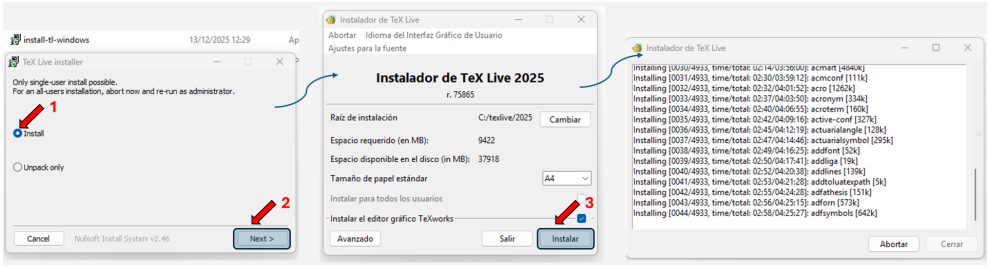

Figura 4: Proceso de instalacion de TEXLive

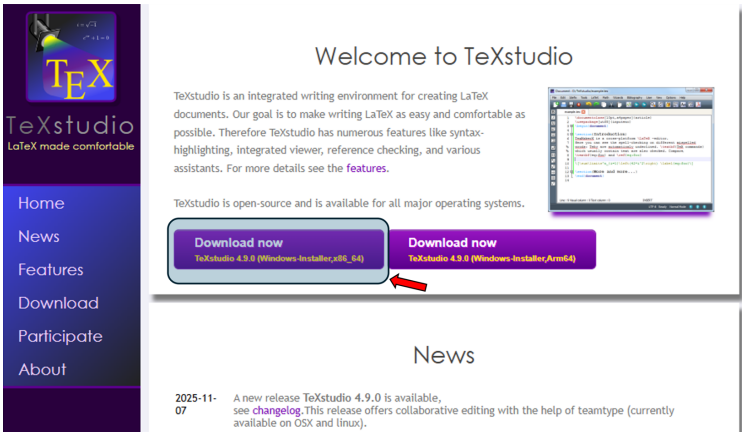

Figura 5: Página web de TexStudio y opción de bajar la última versión disponible

Una vez bajado el archivo procedemos a ejecutarlo y así dar comienzo a la instalación. Cuando se indique que el proceso se ha finalizado se puede hacer clic en el botón `Cerrar`, como indica el paso 3 de la figura 6.

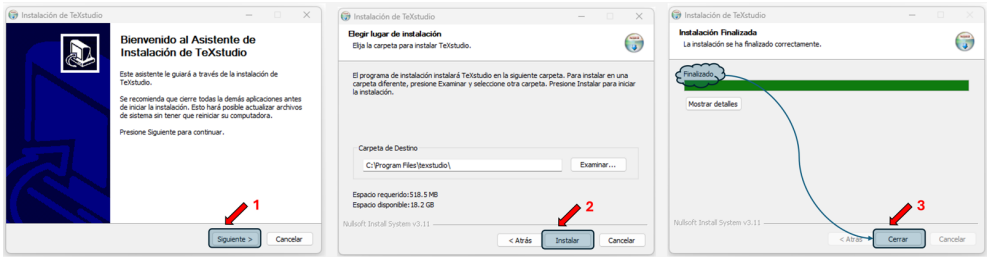

Figura 6: Instalación de TeXStudio

1.2.3. Pre-requisito. Instalación de Python

Una vez instalados tanto TeXLive como TeXStudio a continuación se indican los pasos seguidos para una correcta instalación y uso del paquete `minted` en el programa TeXstudio.

El paquete `minted` usa el *script* externo `Pygments`, por lo que se necesita instalar de manera previa Python.

Las instrucciones de instalación se han basado parcialmente en lo descrito en la página web de Rubén Sánchez[9], si bien, como ya se ha indicado en el capítulo de Introducción, vamos a intentar evitar el uso de `-shell-escape`.

Debido a la aparente incompatibilidad de `minted` con Python 3.14.1 hay que buscar una versión anterior con la que probar. Se considera que la versión 3.11[10]es suficientemente estable para entorno de ingeniería, y se busca por tanto el último Python instalable dentro de esa versión. La última versión antes de cambiar a la 3.12 es la 3.11.14, pero se indica[11] que esta es una versión no instalable, y que la última versión instalable es la 3.11.9., como se muestra en la figura 7.

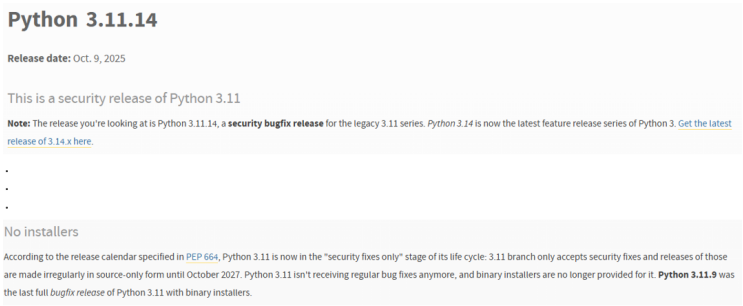

Figura 7: Versión de Python a instalar - Python 3.11.9

Por tanto vamos a la página de descargas de Python[12] y navegamos para buscar la opción `Looking for a specific release?` (¿Buscas una versión en específico?), bajaremos hasta llegar a la `Release version` Python 3.11.9 del 2 de abril de 2024, como se muestra en la figura 8.

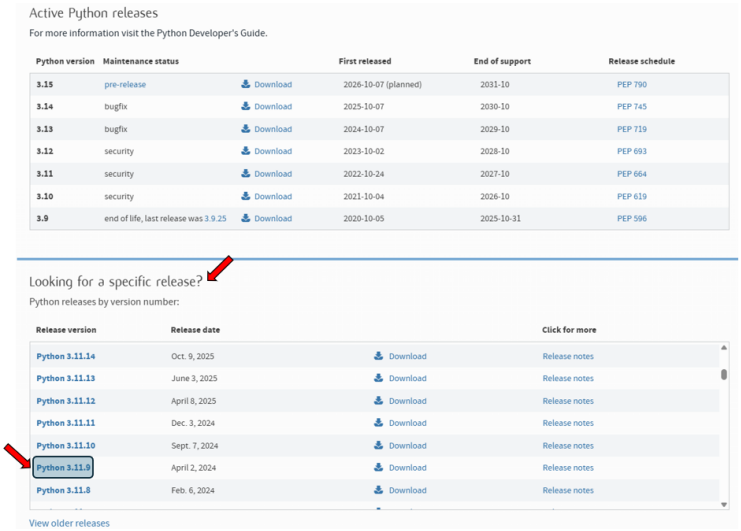

Figura 8: Búsqueda de la versión de Python 3.11.9

[9] `http://rubensm.com/?s=minted`

[10] De manera adicional se indica en la descripción del paquete `minted` por parte del autor que el archivo de configuración `latexminted_config` puede estar en formato Python literal, y requerir, por tanto, una versión de Python 3.11+ `https://ctan.org/pkg/minted`

[11] `https://www.python.org/downloads/release/python-31114/`

[12] `https://www.python.org/downloads/`

Al hacer clic sobre la versión 3.11.9 se abre una nueva ventana, correspondiente a esta versión. Navegamos por la página hacia su parte inferior, donde encontramos la zona de descarga. En este caso optamos por el archivo de Windows, como muestra la figura 9.

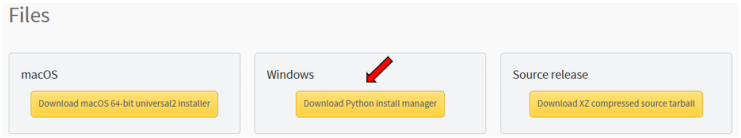

Figura 9: Obtención del archivo de instalación de Python 3.11.09

Una vez se haya descargado el archivo de instalación procedemos a ejecutarlo. Es muy importante asegurarse de añadir Python al PATH del sistema durante la instalación como muestra la figura 10.

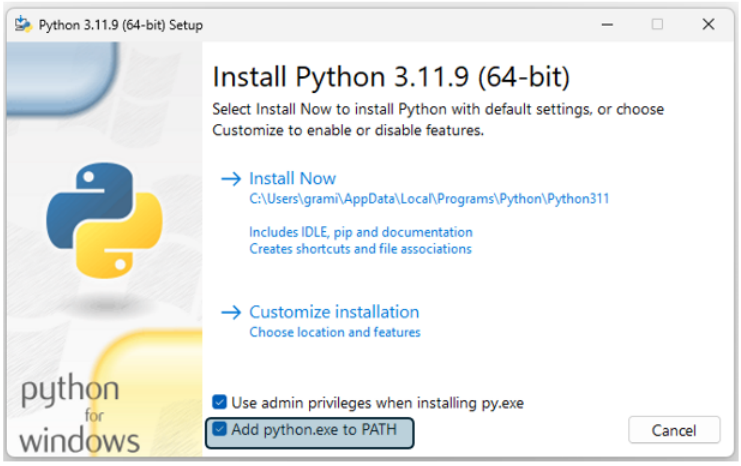

Figura 10: Añadir Python al PATH del sistema antes de instalar Python

Para asegurarse que Python se ha añadido de manera correcta al PATH del sistema se puede abrir la consola de Windows (tecleando `cmd` en la barra de búsquedas o presionando ⊞ + R, que abre el cuadro *Ejecutar*, en el que se puede teclear `cmd`, por último haz clic en *Aceptar* o presiona la tecla *Enter*). Una vez abierto procedemos a teclear `Python` y pulsamos *Enter*. Si hemos añadido bien Python al PATH debería salir un mensaje como el de la figura 11.

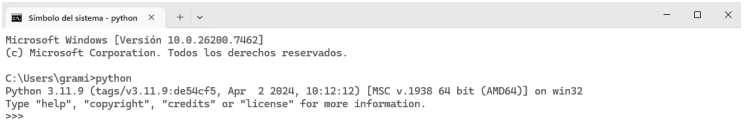

Figura 11: Comprobación de haber añadido Python al PATH

En caso de no mostrar ningún mensaje Python no está añadido al PATH y se debe de añadir de manera manual. Para ello abriremos una ventana del Explorador de Windows y sobre el icono del símbolo de "Este equipo" haremos clic derecho para abrir el menú contextual. En dicho menú buscaremos la opción de *Propiedades*. Al elegirla se nos abre la pantalla de Información del Sistema, tal y como muestra la figura 12.

Haremos clic en el botón que muestra la opción de *Configuración avanzada del sistema*, lo que a su vez nos abre otra ventana, correspondiente a las Propiedades del sistema, como se muestra a la izquierda en la figura 13.

En la parte inferior derecha de esta ventana está el botón de *Variables de entorno*, que debemos pulsar para entrar a la ventana de configuración de estas variables. En la parte

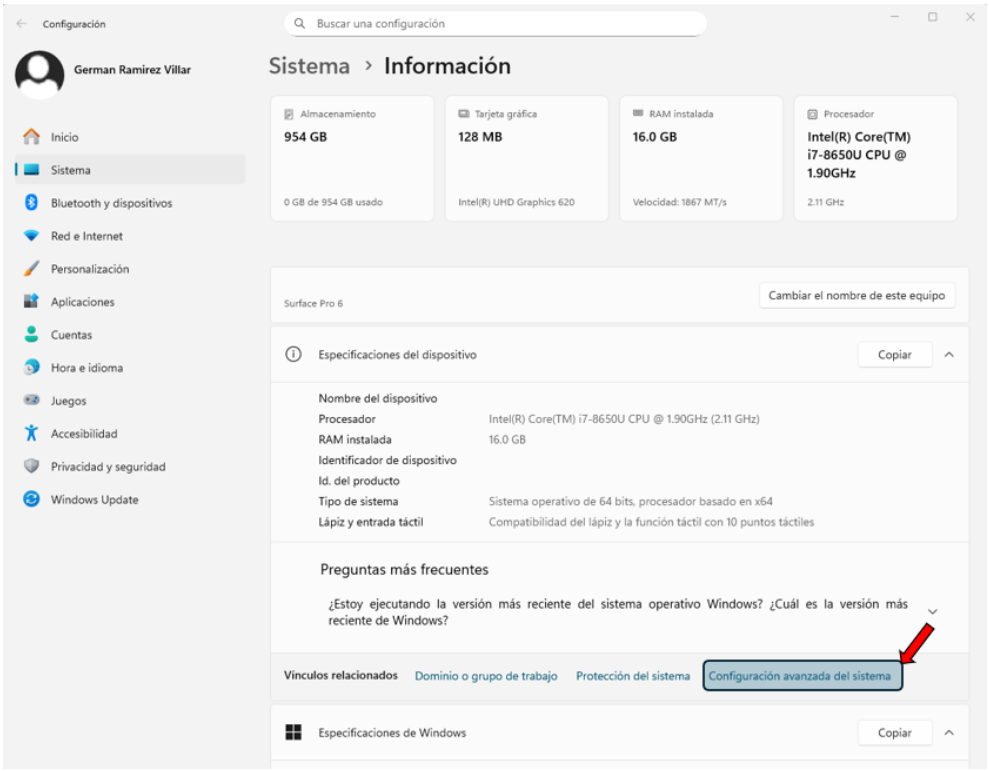

Figura 12: Información de Sistema y Configuración avanzada del sistema

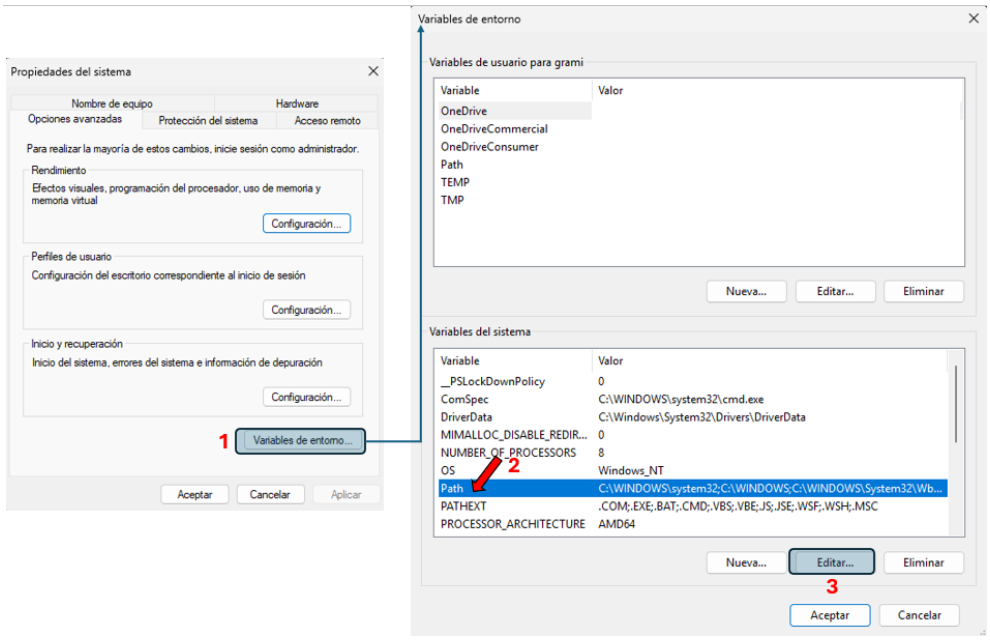

Figura 13: Variables del entorno

inferior de la ventana se encuentra el apartado de *Variables del sistema*, donde debemos buscar la variable *Path*. Una vez seleccionada pulsamos en el botón de *Editar...*.

En la figura 14 se puede observar la nueva ventana emergente, *Editar variable de entorno*, en la que debemos hacer clic sobre el botón *Nuevo*, lo que nos permitirá introducir la ruta de ubicación de nuestro archivo ejecutable de Python *python.exe*. De igual manera se debe crear un nuevo PATH para la carpeta *Scripts*. Una vez acabado el proceso de adición de nuevas rutas de acceso se hará clic sobre el botón *Aceptar*

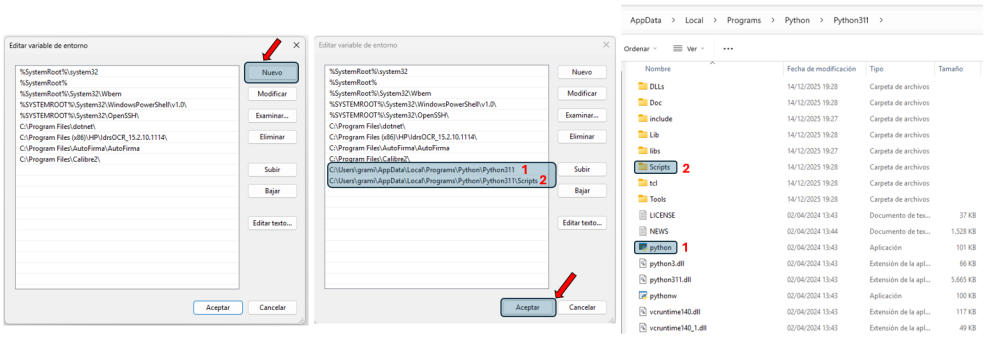

Figura 14: Adición de rutas en PATH desde las variables de entorno

1.2.4. Pre-requisito. Instalación de PIP

PIP[13] es sistema de gestión de paquetes estándar de Python y se utiliza para instalar, gestionar y desinstalar bibliotecas y módulos de Python que no formen parte de la biblioteca estándar, buscando los paquetes en Índice de Paquetes de Python (PyPI).

Para comprobar si tienes instalado PIP puedes abrir la consola de Windows, tal y como hemos visto antes, y teclear el siguiente texto, pulsando *Enter* al terminar:

```
pip --version
```

En caso de ya tener instalado PIP se mostrará un mensaje indicando la versión instalada.

Figura 15: Comprobación de versión de PIP

Por el contrario, si lo que se muestra es un mensaje de error, se puede instalar mediante un *script* de instalación manual. Se abre de nuevo la consola de Windows, y se escribe el texto siguiente, pulsando *Enter* al terminar:

```
py -m ensurepip --upgrade
```

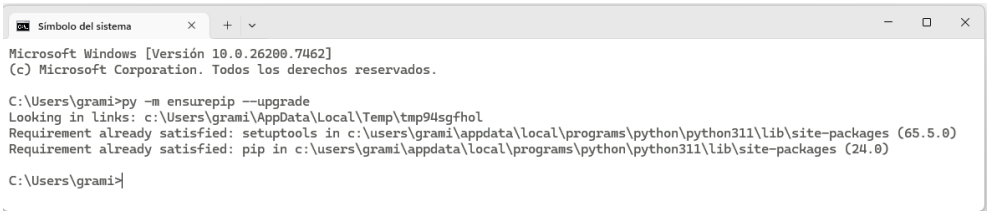

Figura 16: Instalación de PIP

Una vez instalada, es una buena práctica mantener PIP actualizado a su versión más reciente ejecutando:

```
py -m pip install --upgrade pip
```

[13] `https://pip.pypa.io/en/stable/installation/`

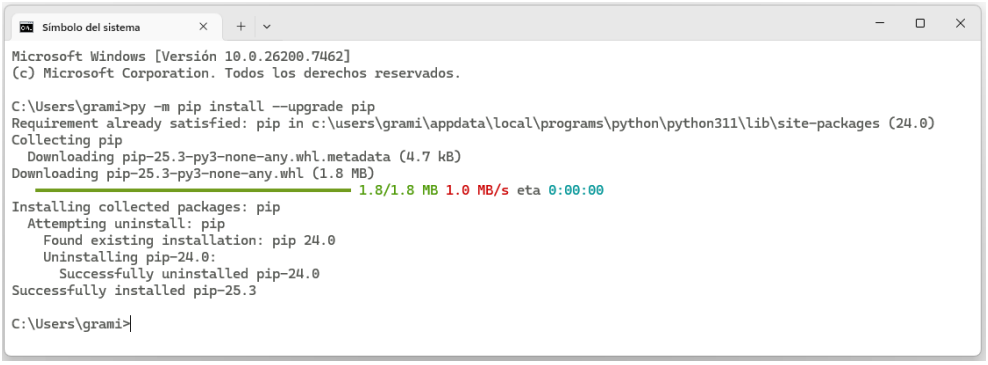

Figura 17: Actualización de PIP

1.2.5. Instalación de `Pygments`

Para instalar `Pygments` abriremos la consola de Windows y escribiremos el siguiente texto, pulsando *Enter* al terminar:

```
pip install Pygments
```

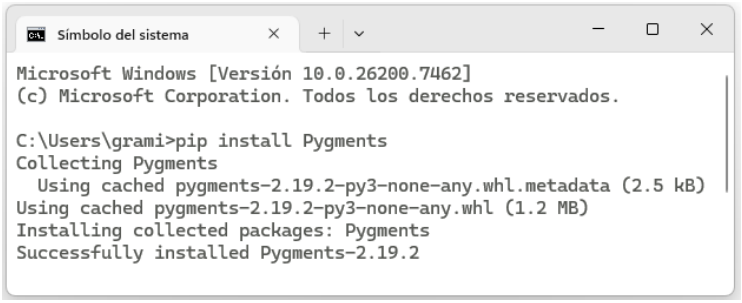

Figura 18: Instalación de Pygments

Se puede comprobar si se ha instalado correctamente `Pygments` escribiendo en la consola de Windows el texto siguiente, pulsando *Enter* al terminar, lo que nos mostrará la versión del paquete:

```
pygmentize -V
```

1.2.6. Instalación de `latexminted`

Para la instalación del paquete `latexminted` abriremos la consola de sistema y tecleare-mos el siguiente texto, pulsando *Enter* al terminar:

```
pip install latexminted
```

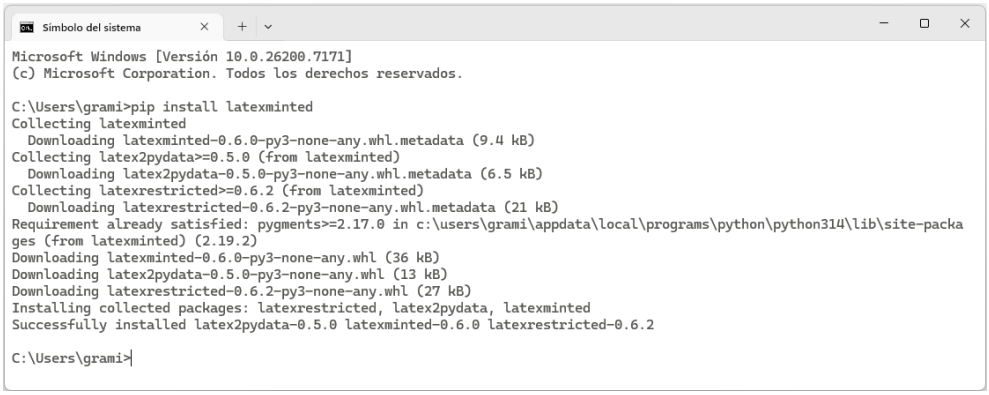

Figura 19: Instalación desde la consola de `latexminted`

Si quisiéramos comprobar si tenemos el paquete `latexminted` instalado abriremos la consola del sistema y teclearemos el siguiente texto, pulsando `Enter` al terminar:

```
latexminted --version
```

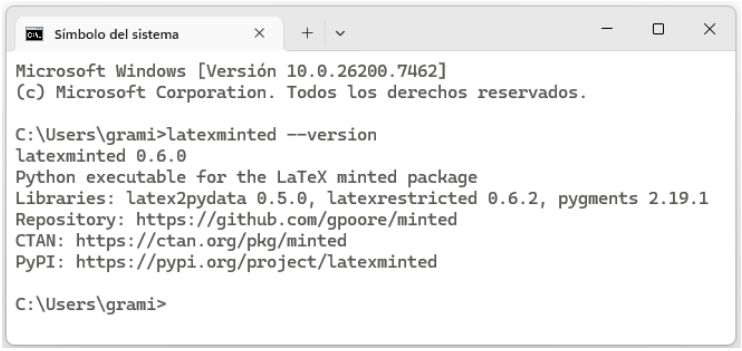

Figura 20: Comprobación de la versión de `latexminted`

1.2.7. Reinicio del sistema

Una vez se han realizado todas las instalaciones se debe de reiniciar el equipo, de otra manera al compilar sin reiniciar dará error.

A la hora de compilar es posible que haya que hacerlo dos veces para salga el texto de manera correcta.

1.3. Uso

1.3.1. Carga de minted en el documento para su posterior uso

El paquete minted[14] se debe de cargar en el preámbulo del documento.

```
\documentclass{article}
...
\usepackage{minted}
...
\begin{document}
...
\end{document}
```

1.3.2. Sintaxis básica de minted. Lenguajes disponibles

La sintaxis básica de este paquete es la siguiente:

```
\begin{minted}[<opción1>,<opción2>,...]{<lenguaje>}
        <código>
\end{document}
```

Existen multitud de lenguajes de programación asociados a este paquete, se puede revisar el listado de lenguajes disponibles en `https://pygments.org/languages/`

Como se puede ver en la figura 21 se puede buscar el lenguaje de programación deseado, como por ejemplo Python o LaTeX, y se accede a la información del tipo de *Lexer*.[15]

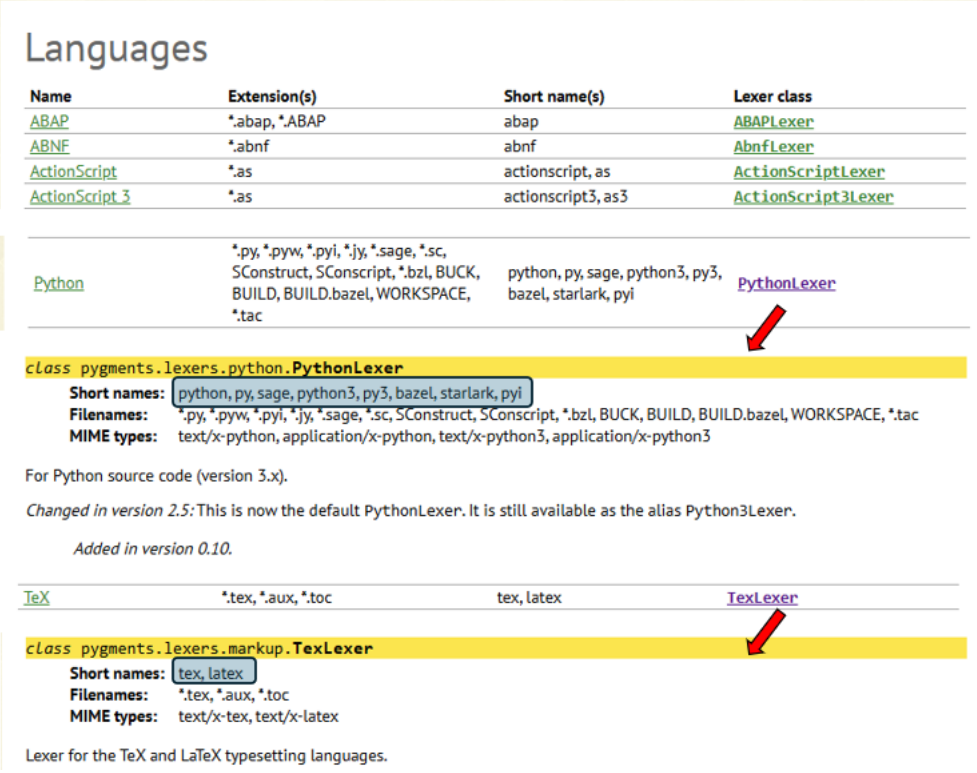

Figura 21: Lenguajes de programación soportados por Pygments

Por tanto, para usar minted con el lenguaje de programación Python usaremos los "Short names" `python, py, ...`.

Para el caso de documentar documentos de LaTeX se usará `tex` o `latex`.

[14] `https://ctan.org/pkg/minted`

[15] Un *Lexer* o analizador léxico es un programa componente que realiza la primera fase del procesamiento de un código fuente o texto. Convierte una secuencia de caracteres (texto plano "sin significado") en una secuencia de *tokens* (unidades con significado léxico). En el contexto de Pygments el lexer es el encargado de identificar qué parte del código es un comentario, qué parte es una función o qué parte es un operador para poder aplicarles colores diferentes según el estilo visual elegido.

1.3.3. Sintaxis básica de minted. Opciones del paquete

A continuación se muestran algunas de las opciones más interesantes del paquete minted, con las que podremos:

- Numerar y destacar (*highlight*) algunas líneas de código.
- Enmarcar y poner un fondo de otro color a las líneas de código.
- Colocar descripción (*caption*) de un fragmento de líneas de código.
- Tener en cuenta el número de capítulo en las descripciones.
- Hacer un listado de las descripciones de líneas de códigos.
- Cargar directamente el texto del archivo.

Para la descripción de estas opciones de una manera más visual usaremos el ambiente `example`[16], que nos permite ver el texto tal cual lo escribimos en LaTeX a la izquierda y a la derecha la "ejecución" del código, tal y como se vería en pantalla.

Numerar y destacar. Para numerar las lineas de código usaremos la opción `linenos`. Ejemplo tomado del manual de minted, al que se le ha añadido la opción de numerar las líneas de código.

```
\begin{minted}[linenos]{ruby}
 class Foo
  def init
   pi = Math::PI
   @var = "Pi = #{pi}..."
  end
 end
\end{minted}
```

```ruby
class Foo
 def init
  pi = Math::PI
  @var = "Pi = #{pi}..."
 end
end
```

Para destacar (*highlight*) una parte del texto usaremos de manera previa al *enviroment* `minted` el comando `\setminted{escapeinside=||}` de manera que podamos ejecutar parte del código sin que se interprete como texto propio del lenguaje de programación, sino como código de LaTeX que se debe de ejecutar. Esto nos permite crear una caja coloreada del color que elijamos mediante el comando `\colorbox{<color>}{<Texto>}`.

```
\setminted{escapeinside=||}
\begin{minted}[linenos]{ruby}
 class Foo
  def init
  |\colorbox{red}{pi = Math::PI}|
   @var = "Pi = #{pi}..."
  end
 end
\end{minted}
```

```ruby
class Foo
 def init
  pi = Math::PI
  @var = "Pi = #{pi}..."
 end
end
```

[16] Tomado del archivo `minted.dtx` de la página de Github de Geoffrey M. Poore. `https://github.com/gpoore/minted`

Enmarcar y poner fondo. Para enmarcar unas líneas de código usaremos la opción `frame`, que a su vez tiene las opciones (`none|leftline|topline|bottomline|lines|single`). La opción por defecto es `none`, por lo que si queremos algún tipo de enmarcación de nuestras líneas de código habrá que activar la opción.

```
\begin{minted}[linenos,
frame=single]{ruby}
class Foo
 def init
  pi = Math::PI
  @var = "Pi = #{pi}..."
 end
end
\end{minted}
```

```ruby
class Foo
 def init
  pi = Math::PI
  @var = "Pi = #{pi}..."
 end
end
```

Si quisiéramos incluir un color de fondo para resaltar las líneas de código del resto del texto usaremos la opción `bgcolor=<color>`, pudiendo definir de manera previa nuestro color para los fondos, como se aprecia en el ejemplo adjunto.

```
\definecolor{f_a}{RGB}{255,250,205}
\begin{minted}[linenos,
bgcolor=f_a]{ruby}
class Foo
 def init
  pi = Math::PI
  @var = "Pi = #{pi}..."
 end
end
\end{minted}
```

```ruby
class Foo
 def init
  pi = Math::PI
  @var = "Pi = #{pi}..."
 end
end
```

Una opción interesante es la de poder poner tanto lineas superiores como inferiores para delimitar el fragmento de código, así como la posibilidad de colocar sobre ellas un texto de inicio y de fin, como se puede ver en el siguiente ejemplo, siguiendo la nomenclatura para este tipo de anotaciones del paquete `fancyvrv`, en concreto `frame=lines, label={[<Texto superior>]Texto inferior}`

```
\begin{minted}[linenos,
frame=lines,
label={[Inicio]Final}]{ruby}
class Foo
 def init
  pi = Math::PI
  @var = "Pi = #{pi}..."
 end
end
\end{minted}
```

Inicio

```ruby
class Foo
 def init
  pi = Math::PI
  @var = "Pi = #{pi}..."
 end
end
```

Final

Por último vamos a ver como modificar el grosor de las líneas mediante el comando `framerule=<valor de espesor>`, siendo el valor por defecto de 0.4pt, así como cambiar el color de las líneas mediante `rulecolor=<color>`.

```
\begin{minted}[linenos,
frame=topline,
label=\textcolor{black}{Inicio},
framerule=1.5pt,
rulecolor=red]{ruby}
 class Foo
  def init
  pi = Math::PI
  @var = "Pi = #{pi}..."
 end
end
\end{minted}
```

Inicio

```ruby
class Foo
 def init
 pi = Math::PI
 @var = "Pi = #{pi}..."
 end
end
```

Colocación de descripción (*caption*) de un fragmento de líneas de código. Si en nuestro documento aparecen fragmentos de código y deseamos que éstos tengan su correspondiente descripción, así como poder hacer referencia a este fragmento en otra parte del texto se debe de seguir lo que se indica a continuación.

Dentro del paquete `minted` tenemos el ambiente `listing`, que permite la colocación tanto de descripción `\caption{<descripción>}`, como de etiquetas `\label{lst_etiqueta}`. Si no queremos que la descripción sea nombrada por defecto en inglés (`Listing`) se puede cambiar en el preámbulo por la palabra que queramos utilizar usando el comando `\renewcommand{\listingscaption}{Fragmento de código}`, y así mostrará la descripción en español `Fragmento de código`

Para generar el ejemplo no puedo usar el método en dos cajas separadas, ya que me da error. En su lugar usaré el ambiente `listing` para describir el texto que de escribir en LaTeX y a continuación irá la ejecución del código dentro de un ambiente `minted`

```
\begin{listing}
\definecolor{f_a}{RGB}{255,250,205}
\begin{minted}[linenos,bgcolor=f_a]{ruby}
  class Foo
   def init
    pi = Math::PI
    @var = "Pi = #{pi}..."
    end
   end
\end{minted}
\caption{Código ejemplo.}
\label{ref_codigo1}
\end{listing}
Tomemos como ejemplo el fragmento de código \ref{ref_codigo1} para
    ver una referencia a un listado o fragmento de código.
```

```ruby
class Foo
 def init
 pi = Math::PI
 @var = "Pi = #{pi}..."
 end
end
```

Fragmento de código 1: Código ejemplo.

Tomemos como ejemplo el listado 1 para ver una referencia a un listado o fragmento de código.

Para tener en cuenta el número de capitulo en las descripciones (*caption*). Para que en la descripción del fragmento de código o *listing* se tenga en cuenta el número del capítulo se debe de poner la opción `[chapter]`a la hora de de cargar el paquete `minted`, esto es:

```
\usepackage{minted}[chapter]
```

Hacer listado de las descripciones de las líneas de código. En el caso de que quisiéramos disponer de un listado de los distintos fragmentos de código debemos de ejecutar la macro:

```
\listoflistings
```

.

Para que el listado no aparezca con el nombre en inglés debemos de ejecutar el siguiente comando en el preámbulo del documento, poniendo el nombre que se desee en español:

```
\renewcommand{\listoflistingscaption}{Lista de Fragmentos de Código}
```

Uso de archivos externos para carga de código. Supongamos que queremos transcribir un archivo extenso o que pueda modificarse a lo largo del tiempo, dentro del paquete `minted` podemos cargar dicho archivo directamente invocándolo con el comando:

```
\inputminted[<opciones>]{<lenguaje>}{<nombre del archivo>}
```