DigitAll

Formación en
Competencias
Digitales

3
Creación de contenidos digitales

DigitAll

Formación en
Competencias
Digitales

Creación de contenidos digitales

Nivel C1

ÍNDICE

DigitAll

Creación de
contenidos digitales

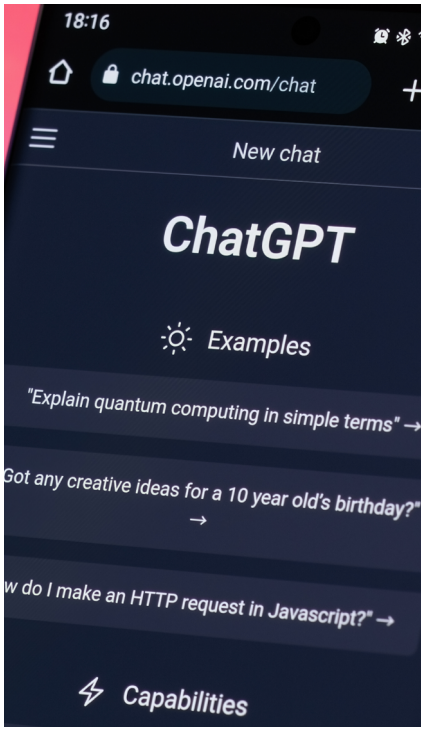
3.1
DESARROLLO DE CONTENIDOS

Creación de
contenidos digitales

Nivel C2 3.1 Desarrollo de contenidos

Generación automática de texto mediante GPT

Generación automática de texto mediante GPT



Introducción

La generación de texto mediante GPT es una herramienta muy valiosa en la creación de contenidos digitales con multitud de beneficios según el campo en el que se aplique. Así, GPT es capaz de producir texto coherente y convincente en una amplia variedad de temas, lo que nos va a permitir ahorrar tiempo y esfuerzo al no tener que redactar cada palabra desde cero. Además, esta tecnología permite a los usuarios explorar ideas nuevas y creativas, ya que puede sugerir diferentes enfoques y perspectivas. Por último, puede ser empleada para muchas de las cuestiones que hemos trabajado en éste y niveles anteriores, como por ejemplo la creación de entradas en nuestro sitio web.

GPT facilita la personalización del contenido, adaptándose a las necesidades y preferencias específicas de cada proyecto. En resumen, la generación de texto mediante GPT es una herramienta poderosa que optimiza la creación de contenidos digitales al ahorrar tiempo, mejorar la calidad y fomentar la creatividad.

Es habitual que nos encontremos descripciones de GPT como “una inteligencia artificial que está entrenada para mantener conversaciones” y, aunque esto es cierto, esta percepción puede ser un lastre a la hora de obtener información a través de este sistema. Si lo pensamos detenidamente, resulta algo obvio: la respuesta a una misma pregunta va a ser diferente dependiendo del contexto: quién nos la haga, en qué momento y lugar, etc.

Por tanto, va a ser muy importante tener en cuenta la información que vamos a proporcionar a ChatGPT, u otros modelos de generación de texto, y que va a definir el contexto en el que queremos que el sistema nos proporcione respuestas. En este texto plateamos qué información debemos proporcionar a ChatGPT para obtener un texto que se ajuste a nuestras necesidades específicas.

¿Qué información debemos proporcionar al sistema?

Vamos a describir una serie de apartados generales que debemos definir y que se van a adaptar a la mayoría de las peticiones de generación de texto que hagamos.

Propósito y tema

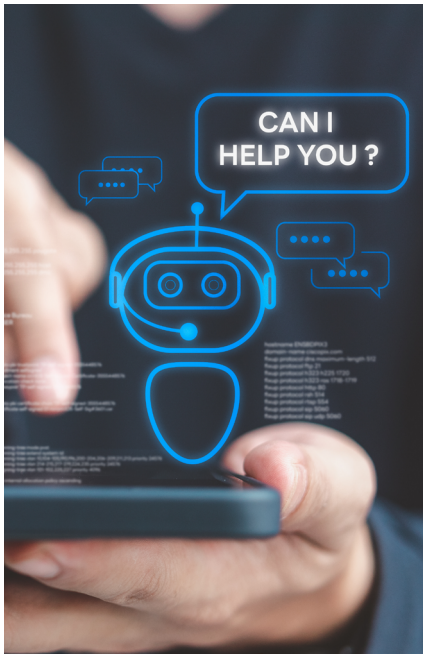
Empezamos por lo más obvio: debemos especificar el tema principal sobre el que queremos que trate el texto y también el propósito con el que lo necesitamos: informativo, persuasivo, descriptivo, divulgativo, etc.

Estructura

Debemos indicar si hay una estructura específica que deseamos que el sistema siga a la hora de generar nuestro texto. Esto va a ayudar al sistema a organizar la información que nos va a proporcionar de manera más efectiva. Por ejemplo, podemos pedir el esquema clásico de un cuento: introducción, nudo y desenlace. Pero también podemos proporcionar indicaciones para generar secciones a nuestro gusto, por ejemplo, una sección específica con ejemplos prácticos sobre un tema.

Información relevante

Debemos suministrar los detalles y datos necesarios que deben incluirse en el texto. Por ejemplo, podemos pedir que nos incluya datos estadísticos, casos prácticos, sucesos históricos, etc.



Estilo y tono

En nuestro *prompt* debemos indicar el estilo y tono que deseamos que tenga nuestro texto. Por ejemplo, formal, informal, técnico, persuasivo, neutro, etc. Además, podemos indicar si tenemos alguna preferencia en cuanto a la longitud de las oraciones o el uso de vocabulario. Por ejemplo, frases cortas o vocabulario técnico.

Audiencia objetivo

¿A quién va dirigido nuestro texto? Esto ayudará a adaptar el lenguaje y el enfoque para que sea apropiado y efectivo para

el público al que se dirige. Por ejemplo, podemos pedir que nos escriba un cuento para niños o para adultos.

Requerimientos adicionales

Si hay requisitos específicos, como la inclusión de referencias (por ejemplo, libros o webs confiables), el uso de lenguaje inclusivo o la extensión del texto, podemos también mencionarlo en nuestra petición.

Revisión y ajustes

Es un paso fundamental que muchas veces se olvida al usar este tipo de herramientas y es que el proceso no termina cuando el sistema nos ha generado un texto en base a nuestro *prompt*. Por tanto, este paso lo haremos después de recibir nuestro texto: lo revisamos y, si hay partes que no cumplen con nuestras expectativas, generaremos una nueva petición con las modificaciones oportunas para que el sistema se ajuste mejor a nuestras preferencias. Por ejemplo, podemos pedir que nos rescriba un párrafo determinado incluyendo ejemplos para hacerlo más comprensible.

Conclusión

Por tanto, al ser un modelo de lenguaje, ChatGPT se basa en patrones y datos existentes para generar respuestas coherentes. Sin embargo, sin información adicional, el modelo puede no comprender por completo el contexto o las especificaciones exactas del texto que deseamos. Al proporcionar una descripción detallada, podemos maximizar el valor y la utilidad de la generación automática de texto y obtener un texto adaptado a nuestras necesidades.

Saber más

En el artículo "Sparks of Artificial General Intelligence: Early experiments with GPT-4" (arXiv:2303.12712) podemos encontrar una investigación exhaustiva sobre cómo se entrenó GPT-4 y cómo se comparó con su predecesor, GPT-3. Aunque se discuten posibles aplicaciones y limitaciones de GPT-4 lo más interesante en relación con este texto es la multitud de diferentes *prompts* que se ensayan y los resultados que devuelve el sistema.

Creación de
contenidos digitales

Nivel C2 3.1 Desarrollo de contenidos

Uso avanzado de rutas para selección, recorte y definición de áreas

Uso avanzado de rutas para selección, recorte y definición de áreas

Introducción

Entre las numerosas herramientas avanzadas de edición y manipulación de imágenes disponibles, las rutas desempeñan un papel fundamental en la selección, recorte y definición precisa de áreas en imágenes. En este texto, nos centraremos en el uso avanzado de rutas para lograr resultados precisos y detallados en estas tareas completando los conocimientos adquiridos en el nivel anterior. Continuaremos trabajando con el programa GIMP como ejemplo.

Controla tus rutas a través del panel de rutas

Ya hemos visto que, a diferencia de las herramientas de selección básicas, las rutas permiten delinear contornos complejos y curvas suaves con mayor precisión. Así, podemos trazar una ruta alrededor de un objeto en una imagen y luego convertir esa ruta en una selección activa y aplicar diferentes efectos o modificaciones específicas a esa área en particular.

Podremos manipular nuestras rutas a través del panel de rutas, una herramienta que permite gestionar y manipular las rutas creadas. Este panel proporciona una vista detallada de todas las rutas presentes en un proyecto y ofrece opciones para editar, organizar, mostrar (o no) y, en definitiva, utilizar esas rutas de manera eficiente. Como vemos en la Figura 1, aparece junto al panel de capas y funciona de una manera muy similar.

Ejemplo práctico

Las posibilidades de las rutas en el diseño son enormes, dependiendo de cuál sea el objetivo final que nos hayamos marcado para nuestra composición, podremos emplear las rutas de un modo u otro. Ahora que sabemos los conceptos más básicos, vamos a describir paso por paso cómo hacer un diseño empleando rutas. El resultado final de aplicar estos pasos en GIMP se muestra en la Figura 2.

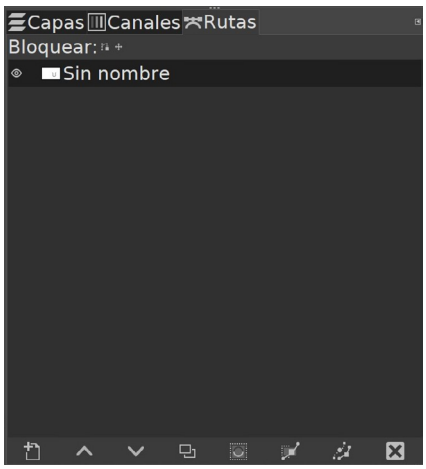


Figura 1 (A). Vista general de una ruta en el panel de rutas.

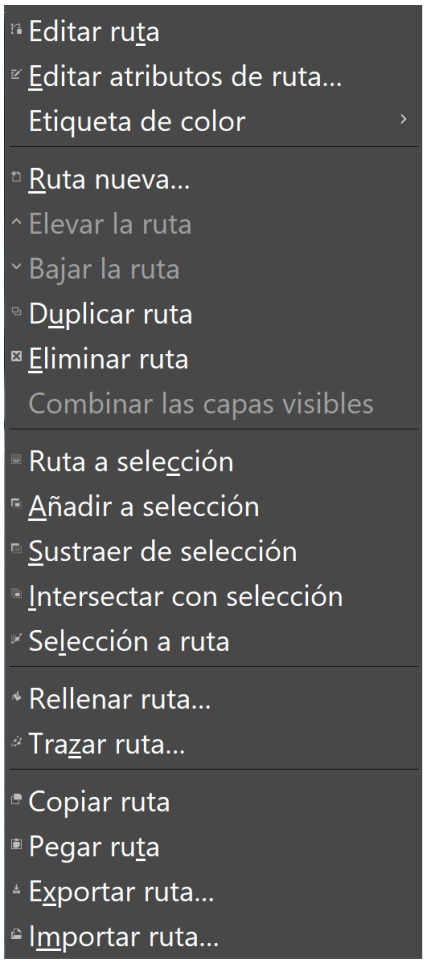


Figura 1 (B). Desplegable que se abre al pulsar el segundo botón del ratón sobre una ruta.

- Descargamos la imagen original (1920×1293) desde: ***e.digitall.org.es/practica-rutas***
- Abrimos la imagen en GIMP.
- Seleccionamos la "herramienta de rutas" y dibujamos una ruta alrededor de la rama y el pájaro.
- Editamos las veces que sean necesarias hasta que consigamos un buen resultado.
- Creamos una selección clicando en "Crear selección a partir de una ruta" en el menú de la herramienta de rutas.
- Copiamos la selección pulsando ***CONTROL + C***.
- Creamos una nueva imagen (Archivo > Nuevo) donde crearemos nuestra nueva composición.
- Pegamos nuestra imagen pulsando ***CONTROL + V***. Aparecerá en el menú de capas como "Selección flotante" por lo que pulsamos el segundo botón y seleccionamos del menú "A una nueva capa".
- Volvemos a la imagen original y en el panel de rutas pulsamos "Copiar ruta".
- Nos vamos al panel de rutas de nuestra nueva imagen y pulsamos "Pegar ruta".
- Ya tenemos en nuestra nueva composición la parte de nuestra imagen que nos interesaba y la ruta que creamos para recortarla.
- Una opción es generar una silueta sobre nuestra imagen usando nuestra ruta como referencia, para ello creamos una nueva capa transparente (entre nuestra capa y el fondo).
- Vamos al panel de rutas, seleccionamos nuestra ruta y pulsamos "Pintar a lo largo de la ruta" (abajo a la derecha).
- En el panel de opciones que nos aparece seleccionamos Trazar línea > Color sólido (anti-alias debe estar seleccionado) y elegimos las opciones del trazo que vamos a aplicar a nuestro gusto.
- Al pulsar en "Trazar" vemos que nos aparece un trazo de las características indicadas con la forma de nuestra ruta en nuestra nueva capa.
- Vamos a añadir un segundo efecto en el que vamos a crear una silueta rellena de color con la forma de nuestra ruta. Para ello duplicamos nuestra ruta y creamos una nueva capa (entre las capas que ya teníamos y el fondo).
- Ahora vamos a la herramienta de rutas y le vamos a decir que queremos "Mover" nuestra ruta y la movemos ligeramente en la dirección deseada.

IMAGEN ORIGINAL EJEMPLO PRÁCTICO
e.digitall.org.es/practica-rutas

Ctrl + C = **COPIAR**

Ctrl + V = **PEGAR**

- Ahora indicamos que queremos “Rellenar ruta” con un “Color sólido” y, de nuevo, indicamos las características del color de relleno (anti-alias debe estar seleccionado). Vemos que nos ha rellenado nuestra ruta con el color que le hemos indicado.
- Finalmente, podemos introducir efectos que ya conocemos para mejorar la composición:
  - Podemos por ejemplo en la capa del fondo añadir color con la “Herramienta de degradado”.
  - Modificar los colores de nuestra imagen recortada desde el panel “Colores”.
  - O cambiar el modo de la capa de relleno que hemos creado.

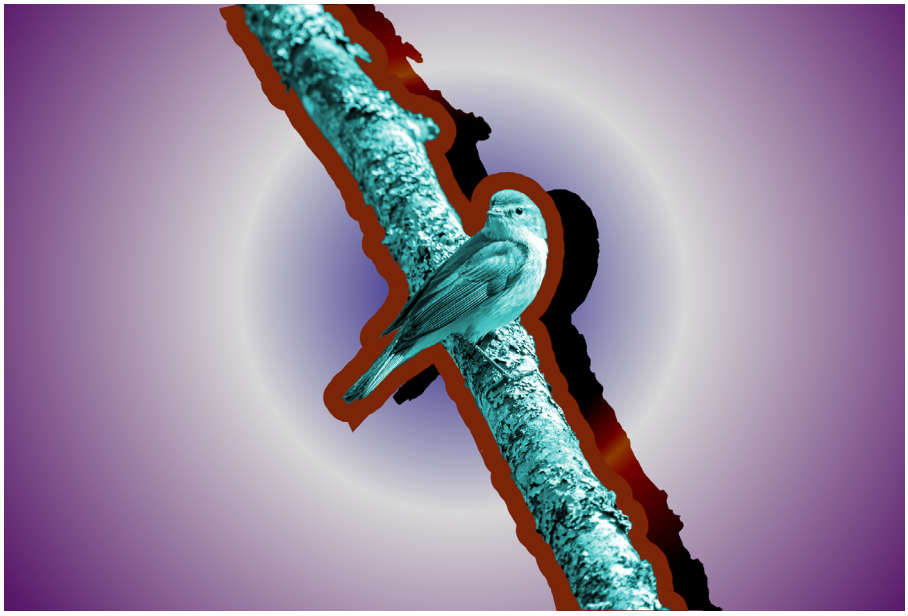


Figura 2. Resultado de aplicar paso por paso las instrucciones descritas en el apartado “Ejemplo práctico”.

Saber más

Las herramientas de dibujo de rutas ofrecen un nivel avanzado de precisión y flexibilidad en la selección, recorte y definición de áreas en imágenes. En todos los programas de diseño nos vamos a encontrar con herramientas similares, en el programa Photoshop, en vez de rutas se denominan trazados. Podemos acceder a información adicional acerca de cómo se organizan estas herramientas en el popular programa de diseño a través del siguiente enlace.

e.digitall.org.es/rutas-photoshop

Creación de
contenidos digitales

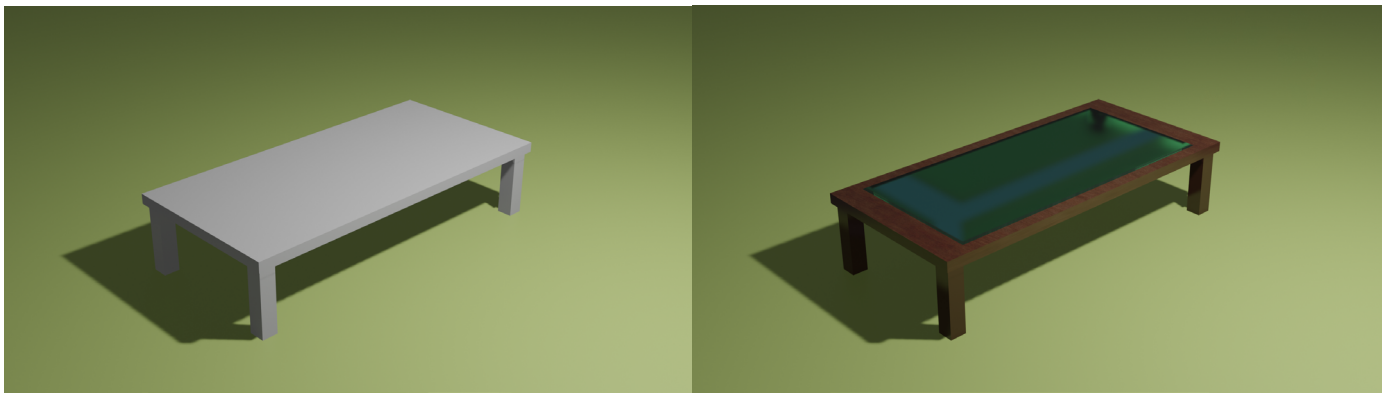
Nivel C2 3.1 Desarrollo de contenidos

Aplicación de materiales y texturizado

Aplicación de materiales y texturizado

Introducción

Definir la geometría de un objeto en el mundo de la informática gráfica es solo el primer paso en la creación de modelos 3D realistas y visualmente atractivos. Una vez que el modelo ha sido diseñado y modelado, es esencial asignar materiales adecuados y, en algunos casos, aplicar texturas para lograr una apariencia realista y coherente. La asignación de materiales determina cómo interactúa el objeto con la luz y cómo se reflejan, refractan o absorben los rayos luminosos en su superficie. Además, el texturizado UV Mapping es una técnica esencial para aplicar imágenes o patrones en la superficie de un objeto 3D con precisión y realismo. En este documento, exploraremos la importancia de asignar materiales y texturas en la creación de gráficos 3D.



A la izquierda tenemos una mesa creada sin asignar una textura predeterminada, mientras que a la derecha tenemos una mesa a la que se le han aplicado diferentes texturas. Nótese la diferencia en cómo actúa la luz en cada una de ellas, consiguiendo una apariencia mucho más realista.

Asignación de materiales y comportamiento de la luz

Asignación de materiales

La asignación de materiales en un objeto 3D es un proceso vital para simular cómo interactúa con la luz. Cada material tiene propiedades específicas que determinan cómo la luz incide y se refleja en su superficie. Algunos materiales pueden ser opacos y reflejar la luz en una sola dirección, mientras que otros pueden ser transparentes o refractar la luz cuando pasa a través de ellos. La configuración de los atributos del material,

como la rugosidad, la reflexión y el índice de refracción, influirá en la apariencia final del objeto y su realismo en la escena.

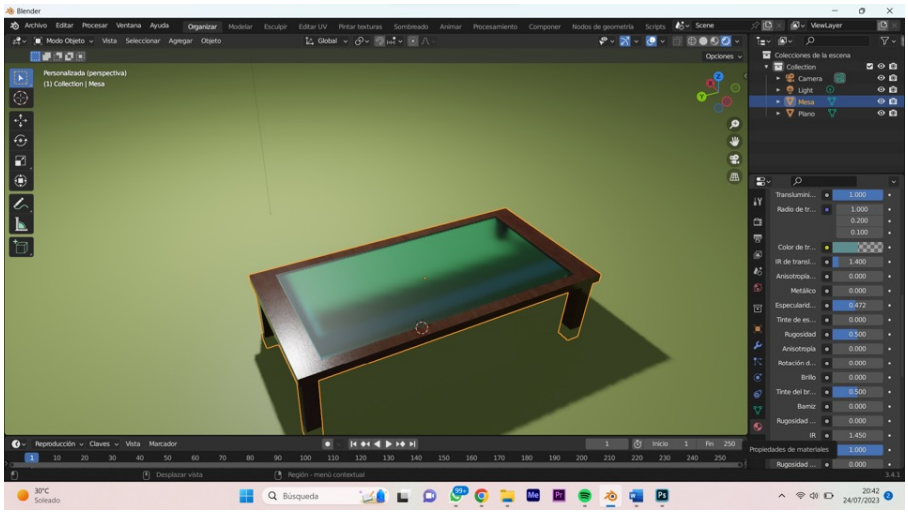
En muchos programas de modelado y renderizado 3D, los materiales se definen utilizando el modelo de iluminación de Phong o el modelo de sombreado de Lambert. Estos modelos permiten simular la forma en que la luz interactúa con las superficies del objeto, y cómo se crea el efecto de sombreado y brillo en función de la posición de la fuente de luz y el punto de vista de la cámara.

Comportamiento de la luz, propiedades

Los materiales definen cómo interactúa el objeto con la luz y cómo se reflejan, refractan o absorben los rayos luminosos en su superficie. Cada material tiene propiedades específicas que determinan cómo se comportará la luz en relación con el objeto. Estas propiedades se definen siempre que se asignen un material. Algunas de las propiedades más importantes son:

- **Color:** el color es el atributo más básico de un material y define cómo se verá el objeto bajo la luz blanca. Puedes elegir un color sólido o una textura de color para obtener un resultado específico.
- **Brillo (especularidad):** este atributo controla la cantidad y el tamaño de los reflejos en la superficie del objeto. Un valor alto de brillo creará reflejos intensos, mientras que un valor bajo dará un aspecto más mate.
- **Rugosidad (roughness):** la rugosidad define qué tan suave o áspera es la superficie del material. Una superficie muy rugosa dispersará la luz y dará un aspecto difuso, mientras que una superficie lisa reflejará la luz en una dirección más precisa.
- **Transparencia:** este atributo controla qué tan transparente es el material. Puedes hacer que el objeto sea completamente transparente o semi-transparente para simular vidrio u otros materiales transparentes.
- **Índice de refracción:** el índice de refracción determina cómo la luz se dobla al pasar a través del material transparente. Esto es especialmente importante para simular materiales como el vidrio o el agua.

- **Texturas:** además de los atributos mencionados anteriormente, puedes aplicar texturas para agregar detalles más complejos y realistas a la superficie del objeto. Las texturas pueden incluir patrones, imágenes o cualquier tipo de diseño que desees incorporar en el modelo.



En Blender podemos definir estas propiedades seleccionando un objeto y pulsando en la ventana de materiales situada abajo a la derecha de la interfaz. En esta captura podemos observar las propiedades usadas para el material creado para simular el cristal.

Es importante tener en cuenta que la asignación de materiales no solo afecta la apariencia del objeto en un renderizado estático, sino que también tiene un impacto en cómo se verá el objeto en diferentes condiciones de iluminación y en movimiento en una animación.

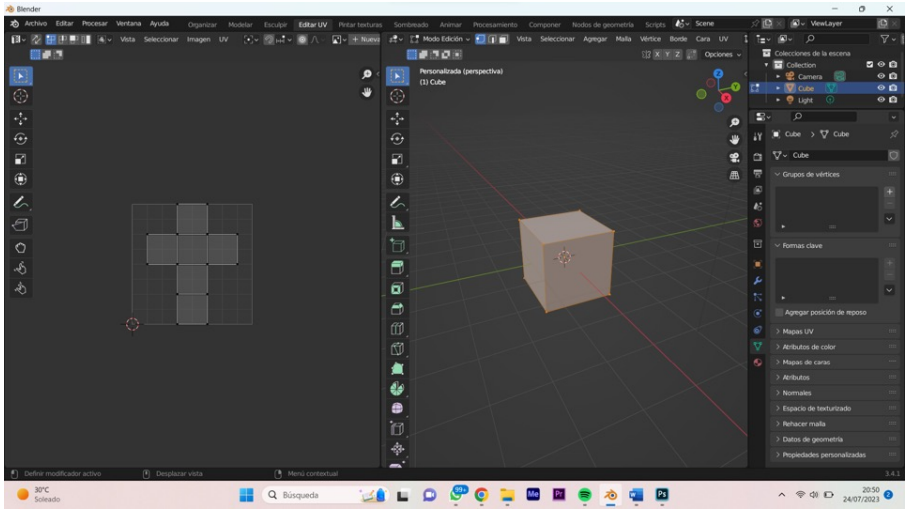
En muchos programas de modelado y renderizado 3D, como Blender, Maya, 3ds Max y otros, encontrarás una interfaz intuitiva para asignar materiales a tus modelos 3D. Estos programas ofrecen bibliotecas de materiales predefinidos y también te permiten crear y personalizar tus propios materiales, ajustando los atributos mencionados anteriormente para obtener el resultado deseado.

Técnica de Texturizado UV Mapping

Asignar una textura a un objeto es sencillo. Podemos añadir una textura directamente a un objeto (entiéndase como textura una imagen que representa “la piel” del objeto) y será el propio programa de diseño 3D quien la distribuya automáticamente y

de forma uniforme a lo largo de toda la superficie del objeto. El problema es que en ocasiones esta distribución automática no permite un ajuste adecuado de la textura. Existen otras técnicas más avanzadas que permiten un ajuste más manual y preciso como la técnica de texturizado de UV Mapping.

El texturizado UV Mapping es una técnica que permite aplicar texturas o imágenes en la superficie de un modelo 3D. Para hacerlo, se deben mapear coordenadas 2D en la superficie del objeto 3D, conocidas como coordenadas UV. Esta técnica es especialmente útil cuando se desea detallar la apariencia del objeto con mayor realismo, como agregar patrones, texturas o imágenes complejas.



Desde Blender tenemos una vista propia dedicada exclusivamente al UV mapping que ya nos "Desenvuelve" el objeto con el que trabajemos.

El proceso de texturizado UV Mapping comienza desplegando las caras del modelo 3D en un plano 2D, de modo que se puedan aplicar las texturas de manera precisa. Esto puede realizarse utilizando herramientas específicas en programas de modelado 3D, o incluso de manera manual si se necesita un control más preciso.

Una vez que las coordenadas UV se han asignado correctamente, se puede aplicar la textura deseada en un programa de edición de imágenes. Luego, la textura se asigna al material del objeto 3D en el software de modelado o renderizado, donde se ajustan parámetros como el tamaño, la intensidad y la repetición de la textura para obtener el resultado deseado.

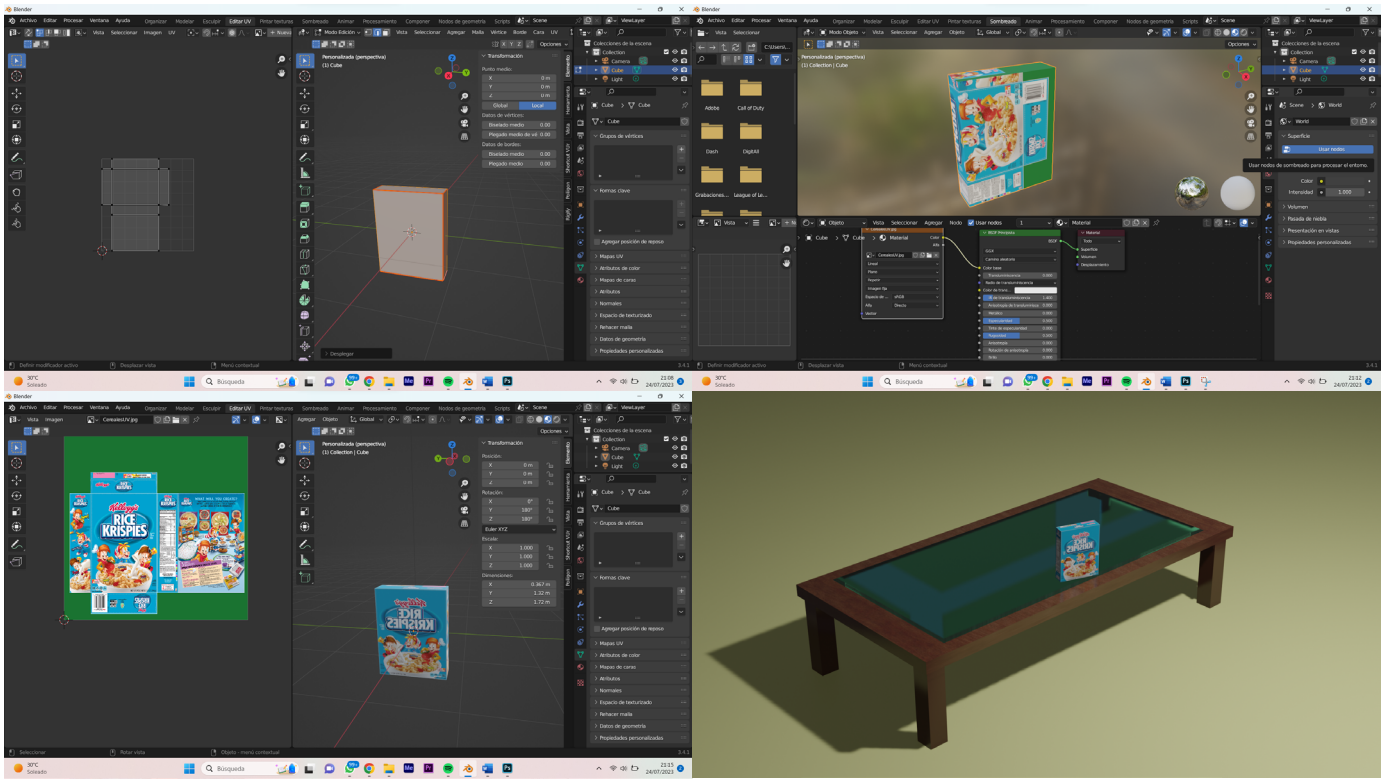
En Blender esto se traduciría únicamente como desenvolver el objeto que queramos desde el apartado UV editting, aplicarle una textura de imagen (seleccionando el mapa de textura que deseemos en png o jpeg como imagen) y volver a UV editting para ajustar esta imagen a nuestro objeto. Es importante resaltar que a más resolución tenga nuestra imagen mejor se verá nuestro render.

Como esto es más fácil decirlo que hacerlo, vamos a intentar hacernos este proceso lo más fácil posible añadiendo algunas pautas a seguir y los menús necesarios para esto. Para empezar y desenvolver el objeto, únicamente nos tenemos que situar en la vista "**UV editting**" donde ya nos aparecerá el objeto desenvuelto. Lo siguiente sería irnos al apartado "**Sombreado**". En esta vista tendremos que adjuntar la imagen (Que debe seguir el formato tipico de mapa UV, solo haría falta buscar en google lo que queramos seguido de "**Texture map**"), para esto arrastraremos la imagen de la carpeta donde la tengamos directamente a la parte inferior de la vista de sombreado, al lado del cuadro llamado "BSDF principista". Una vez tengamos la imagen añadida en este "esquema", tendremos que unir el punto "Color" de nuestra imagen con el punto "Color base" de "BSDF principista" con lo que ya tendríamos asignada esta imagen al objeto que queremos, ahora solo debemos reposicionarla para que se vea correctamente.

Para esto, nos volvemos a la vista "**UV edditing**". En este caso ahora veremos la misma figura desenvuelta junto a la imagen que hemos importado. Para que la imagen esté acorde con e resultado que queramos, debemos hacer click donde tenemos la figura desenvuelta y pulsar la **tecla a**. Con esto tendremos toda la figura desnevuelta seleccionada, ahora debemos mover ésta (con la **tecla g**) o girarla (pulsando la **tecla r** y escribiendo los grados que queramos rotarla si queremos mayor precisión) de modo que la figura desenvuelta coincida exactamente con la imagen del mapa de texturas que hemos añadido.

Cabe añadir que la imagen que usemos no ha de ser necesariamente un mapa de texturas, se puede añadir la imagen que se desee y ajustarla a placer. No obstante, los mapas de texturas están pensados especifiamente para darle forma a figuras concretas y nos pueden ser muy útiles.

A	=	SELECCIONAR FIGURA DESENVUELTA
G	=	MOVER FIGURA
R	=	GIRAR FIGURA



En el orden de las agujas del reloj, en la primera imagen tenemos un cubo que se ha escalado en forma de caja desenvuelto en el UV editting. En la segúnda imagen se nos muestra como se le ha aplicado al cubo una textura con la imagen del mapa de texturas de una caja de cereales aleatoria. En la tercera imagen se muestra como se ha ajustado el mapa de texturas a la fora uv desenvuelta del cubo. En la cuarta imagen se muestra finalmente un render de como quedaría la caja encima de la mesa previamente realizada.

Conclusión

En conclusión, la asignación de materiales y la aplicación de texturas son aspectos críticos en la creación de modelos 3D realistas y visualmente atractivos. La asignación de materiales permite definir cómo la luz interactúa con la superficie del objeto, influyendo en su apariencia y comportamiento visual en la escena. Por otro lado, el texturizado UV Mapping es una técnica valiosa para aplicar imágenes o patrones en la superficie del modelo con precisión y realismo.

Dominar estas técnicas y entender cómo afectan a la apariencia del objeto es fundamental para lograr resultados de alta calidad en proyectos de animación, videojuegos, diseño arquitectónico y otras áreas de la informática gráfica. La combinación adecuada de materiales y texturas agrega un nivel de detalle y realismo que puede marcar la diferencia entre un modelo 3D ordinario y una creación visualmente impactante. Es por ello que será necesario dedicarle muchas horas al entendimiento y uso de estas técnicas.

Creación de
contenidos digitales

Nivel C2 3.1 Desarrollo de contenidos

Renderizado para la generación de imágenes 2D y vídeo

Renderizado para la generación de imágenes 2D y vídeo

Introducción

En el mundo de la animación y los gráficos 3D, el renderizado es un proceso crucial para convertir un modelo 3D en imágenes o vídeos finales. Existen diversas opciones para realizar el renderizado, cada una con sus propios motores de rendering y configuraciones de salida. A continuación, exploraremos algunas de las opciones más populares para el renderizado, así como las configuraciones y opciones disponibles para obtener resultados de alta calidad.

Motores de Rendering

En este caso no nos centraremos solo en Blender, pues ya vimos en su vídeo de introducción ("***Introducción general a la interfaz de una herramienta de diseño 3D***") qué motores de rendering usaba, haremos un recorrido global de motores de rendering:

- **Cycles (Blender):** Cycles es el motor de renderizado de Blender y es ampliamente utilizado debido a su flexibilidad y realismo. Ofrece un enfoque basado en el trazado de rayos que permite simular efectos de luz y sombra de manera precisa, brindando resultados fotorrealistas. Cycles es ideal para producir imágenes de alta calidad y animaciones realistas.
- **Arnold (Autodesk Maya):** Arnold es un motor de renderizado altamente valorado y utilizado en la industria del cine y la animación. Es conocido por su capacidad para manejar complejas escenas y efectos visuales. Arnold proporciona un rendimiento óptimo en el cálculo de iluminación y sombreado, lo que lo convierte en una opción poderosa para proyectos de alta gama.
- **RenderMan (Pixar):** RenderMan es un motor de renderizado desarrollado por Pixar y es ampliamente utilizado en la creación de películas animadas. Es conocido por su capacidad para producir imágenes con detalles y complejidad excepcionales. RenderMan ofrece una amplia gama de configuraciones y opciones avanzadas para un control preciso sobre el resultado final.

INTRODUCCIÓN GENERAL A LA INTERFAZ DE UNA HERRAMIENTA DE DISEÑO 3D

e.digitall.org.es/A3C31C2V04

Configuración y opciones de salida

- **Resolución:** la resolución determina el número de píxeles en la imagen final. Una resolución más alta resultará en imágenes más detalladas, pero también aumentará el tiempo de renderizado.
- **Tamaño del Fotograma:** define el tamaño del área de visualización en la escena 3D. Puedes renderizar en formato de pantalla panorámica, cuadrado o personalizado.
- **Muestreo y Anti-aliasing:** estas configuraciones controlan la calidad de los bordes y eliminan los efectos de escalera (aliasing) en los bordes de los objetos, proporcionando imágenes más suaves.
- **Profundidad de Color:** define la cantidad de información de color por píxel. Mayores profundidades de color (como 16 o 32 bits) permiten un rango más amplio de colores y evitan la pérdida de detalles en las sombras y luces.
- **Formato de Salida:** puedes seleccionar el formato de archivo de salida, como JPEG, PNG, TIFF o EXR, entre otros. El formato elegido dependerá de tus necesidades y de si deseas conservar información de color y detalles de manera óptima.
- **Compresión:** algunos formatos de archivo permiten la compresión para reducir el tamaño del archivo resultante. Sin embargo, ten en cuenta que la compresión puede afectar la calidad de la imagen.

Generación de vídeo

Para generar una animación en vídeo en lugar de un único fotograma, se requiere combinar múltiples imágenes renderizadas en secuencia. Esto se logra mediante el proceso de renderizado de fotogramas clave o keyframes. A continuación se describen los pasos generales para generar un vídeo:

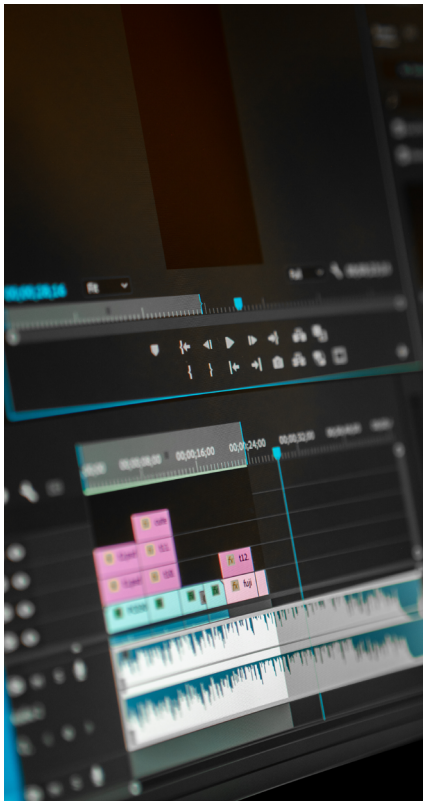
- **Animación y Keyframes:** define la animación en tu escena 3D, estableciendo posiciones clave (keyframes) para objetos, cámaras e iluminación. Los keyframes indican cómo evoluciona la escena a lo largo del tiempo.

- **Secuenciador de Vídeo o Editor de Video:** muchos programas de modelado y renderizado 3D, como Blender, proporcionan un secuenciador de vídeo interno o se integran con un editor de vídeo. Aquí, puedes cargar las imágenes renderizadas y organizarlas en una secuencia para crear el vídeo.
- **Tasa de Framerate:** define la duración de cada fotograma y la velocidad de reproducción para el vídeo. La velocidad de reproducción se mide en frames por segundo (FPS), típicamente 24 FPS para una animación fluida.

Ahora, combinando los conocimientos generales que tenemos sobre animación y los motores de rendering, seremos capaces de crear cualquier imagen. De esta forma, seremos capaces de crear también cualquier secuencia de las mismas o lo que es lo mismo, un vídeo, pues cualquier motor de los que hemos revisado antes nos permite crear una secuencia de imágenes entre *Keyframes* ajustándose a un *framerate* previamente establecido.

Conclusión

En conclusión, el renderizado es un proceso esencial en la creación de imágenes y animaciones 3D realistas y visualmente atractivas. La elección del motor de rendering y las configuraciones de salida son determinantes para lograr resultados de alta calidad. Además, la generación de vídeo implica combinar múltiples fotogramas renderizados en una secuencia animada para lograr animaciones impresionantes y fluidas. Con el dominio de estas opciones y técnicas, podrás crear proyectos de animación y gráficos 3D de alto nivel.



Saber más

Es interesante consultar los manuales que usan actualmente los animadores de empresas famosas como Disney o Pixar pues incluyen todo lo que hemos escrito en este documento aunque mucho más desarrollado.

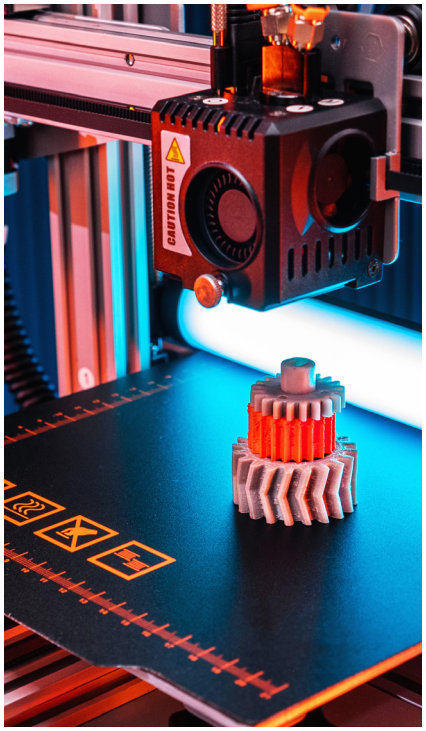
e.digitall.org.es/pixar

Creación de
contenidos digitales

Nivel C2 3.1 Desarrollo de contenidos

Fabricación de objetos 3D

Fabricación de objetos 3D



Introducción

Impresión 3D

Historia Impresión 3D

La impresión 3D ha revolucionado la forma en que fabricamos objetos, permitiendo la creación de prototipos, herramientas y piezas personalizadas de manera rápida y eficiente. Una parte fundamental del proceso de impresión 3D es la exportación de modelos 3D en formatos adecuados para su posterior impresión. En esta primera página, exploraremos los formatos de exportación admitidos por impresoras 3D, así como las aplicaciones y herramientas utilizadas para la impresión.

Tendemos a pensar que la impresión 3D es algo puramente actual, pero en realidad la impresión 3D se remonta a la década de 1980, cuando el concepto de fabricación aditiva comenzó a tomar forma. En 1984, Chuck Hull, un ingeniero estadounidense, inventó el proceso de estereolitografía (SLA), que se considera el primer método de impresión 3D. La SLA permitía crear objetos tridimensionales capa por capa, utilizando fotopolímeros sensibles a la luz ultravioleta. A medida que avanzaron los años, otras técnicas de fabricación aditiva, como la fusión por deposición de material (FDM) y la sinterización selectiva por láser (SLS), fueron desarrolladas y comercializadas.

¿Cómo imprimimos en 3D?

Cuando queramos imprimir algo en 3D habrá dos cosas que necesitaremos principalmente. Lo primero, sería el soporte hardware, es decir, una impresora 3D. Por otro lado, necesitaremos una pieza 3D compatible con la impresión.
En cuanto al hardware, es importante saber que existen principalmente dos tipos de impresora: impresoras de resina e impresoras de filamento.

- **La impresión 3D por filamento**, también conocida como FDM (Fused Deposition Modeling) o FFF (Fused Filament Fabrication), es uno de los métodos más comunes y accesibles de impresión 3D. En este proceso, un filamento termoplástico, generalmente PLA o ABS, es calentado y

extruido a través de una boquilla. El material fundido se deposita capa por capa para construir el objeto 3D. El modelo se imprime sobre una plataforma de construcción que se mueve en ejes X, Y y Z. Una vez depositada una capa, la plataforma desciende y se inicia la impresión de la siguiente capa.

- **La impresión 3D por resina**, también conocida como SLA (Stereolithography) o DLP (Digital Light Processing), es una tecnología de impresión 3D basada en resina líquida fotosensible. En este proceso, un láser o un proyector DLP expone selectivamente la resina líquida, causando que se solidifique capa por capa. A diferencia de la impresión por filamento, donde se construyen objetos mediante la deposición de material fundido, la impresión por resina crea objetos sumergiendo una plataforma en la resina y elevándola de manera controlada a medida que las capas se solidifican.

En resumen, utilizaremos una impresora de filamento cuando no queramos un gran nivel de detalle, pero si queramos una pieza grande, robusta, con un gran nivel de acabado superficial o destacar detalles pequeños de una pieza, entonces deberíamos elegir resina. También cabe destacar que el filamento es la opción más popular debido a su fácil uso.

En este documento nos centraremos sobre todo en la segunda opción, pues las impresoras 3D son un mundo muy extenso y nosotros nos vamos a dedicar a ampliar y aplicar la formación anterior que tenemos sobre editores 3D, en este caso para poder darle una forma física a lo que hemos aprendido a modelar. Aprenderemos ciertas pautas y formatos que debemos seguir si queremos que nuestro objeto 3D pase a nuestro mundo como un objeto físico similar al que nos habíamos imaginado

Proceso de Preparación e Impresión

Preparación del Modelo 3D para Impresión

La impresión 3D implica más que simplemente cargar un modelo en la impresora. En este punto profundizaremos en el proceso de preparación del modelo, los ajustes de impresión necesarios y los pasos para llevar a cabo la impresión en sí.

Antes de imprimir, el modelo 3D debe ser preparado para asegurar que esté listo para la fabricación aditiva. Esto implica la revisión del modelo, la detección y corrección de errores geométricos, y la realización de ajustes para garantizar la estabilidad del objeto durante la impresión. Asimismo, se pueden agregar estructuras de soporte para aquellos diseños que lo requieran. Estos cambios antes de la impresión los podríamos dividir en dos fases, la fase del editor y la fase del software de impresión.

Fase edición

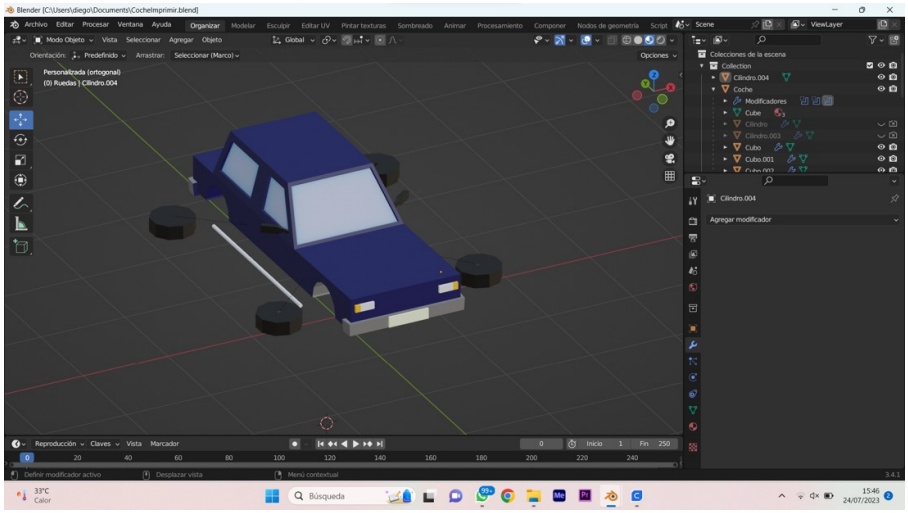
Dentro de un editor, como sería en nuestro caso Blender, debemos tener bastantes cosas en cuenta. A continuación, dejaremos unas pautas:

- **Debemos verificar que nos ajustarnos a las dimensiones y escala reales.** Si bien esto es algo que podremos corregir a posteriori como veremos en el software de impresión 3D, a la hora de diseñar una pieza que deseamos imprimir debemos tener muy en cuenta el tamaño de nuestra impresora para no pasarnos si no queremos dividir el modelo en trozos e imprimirlo en fases.
- **Debemos optimizar al máximo el número de vértices y polígonos.** Limpia y optimiza la geometría de tu modelo. Elimina geometría innecesaria y evita triángulos y ngons (polígonos de más de cuatro lados) ya que pueden causar problemas durante la impresión.
- **Debemos verificar la orientación.** Asegúrate de que la orientación del modelo sea adecuada para la impresión. La base del modelo debe estar plana y en contacto con la superficie de impresión. Evita salientes excesivamente pequeños o estructuras muy delgadas, ya que pueden ser difíciles de imprimir.
- **No debemos colocar elementos flotantes.** Si tu modelo tiene piezas móviles o elementos separados, asegúrate de que estén conectados adecuadamente para evitar problemas de impresión y ensamblaje posterior.
- **No debemos usar caras totalmente planas.** En el mundo real, no existen los objetos totalmente planos, y esto es algo que debemos tener en cuenta pues las impresoras 3D imprimen objetos sólidos, por lo que asegúrate de que tu

modelo tenga un espesor adecuado en todas las partes. Si tu modelo es solo una superficie, conviértela en un objeto 3D sólido mediante el modificador Solidify en Blender.

- **No debemos solapar piezas.** Asegúrate de que las partes del modelo no se superpongan, ya que esto puede causar problemas de impresión y dar lugar a piezas fusionadas o distorsionadas.

Otro consejo que deberíamos tener en cuenta sería exportar siempre los archivos en los formatos estándar STL y OBJ que trabajan bien con cualquier software de impresión 3D.

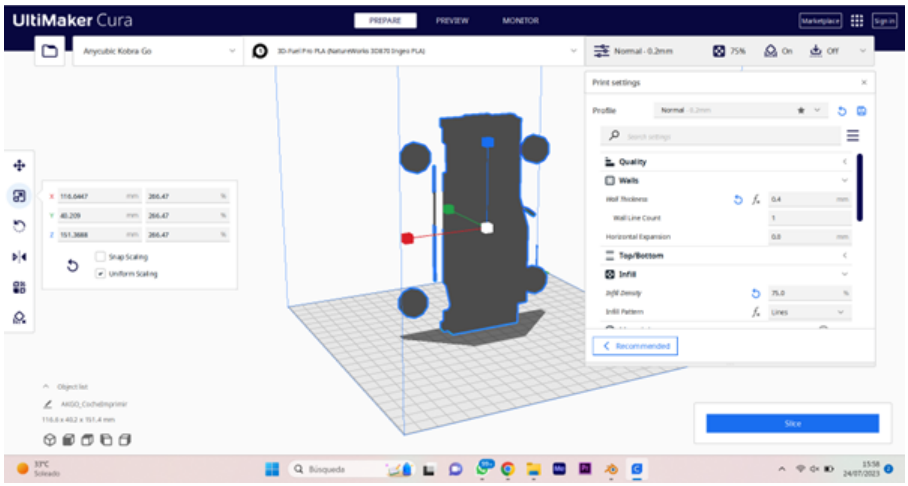


En esta captura podemos observar un modelo listo para impresión en Blender. Todas las piezas están a nivel del suelo con lo que no solo no tenemos piezas flotantes, sino que además hemos conseguido en una misma impresión más piezas. Cabe añadir que es aconsejable tener activada la vista de renderizado ya que, aunque el color no se imprimirá, puede ser interesante tenerla para que nos sirva de guía.

Fase software impresión

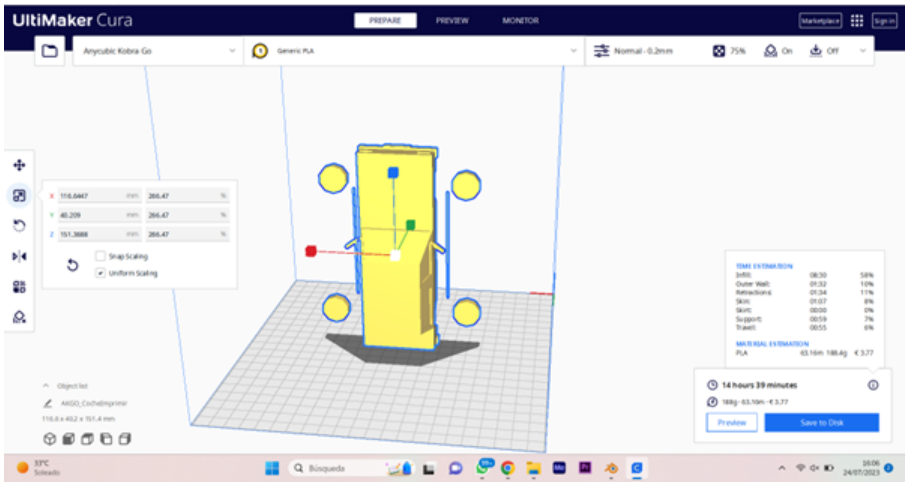
Tomaremos como software predefinido CURA, una aplicación diseñada para impresoras 3D gratuita, en la que se pueden modificar los parámetros de impresión y después transformarlos a código G, el necesario para cualquier archivo destinado a ser imprimido.

Lo primero que se debe hacer al abrir CURA sería seleccionar el perfil de impresora correspondiente y cargar tu modelo 3D en formato STL u OBJ. Luego, ir a la pestaña “Preparar” (Prepare). Tras esto tendremos muchos parámetros a modificar como la velocidad de impresión, espesor o temperatura de la cama, que de momento dejaremos en su estado predeterminado pues por defecto nos servirán para cualquier primera pieza que deseemos imprimir



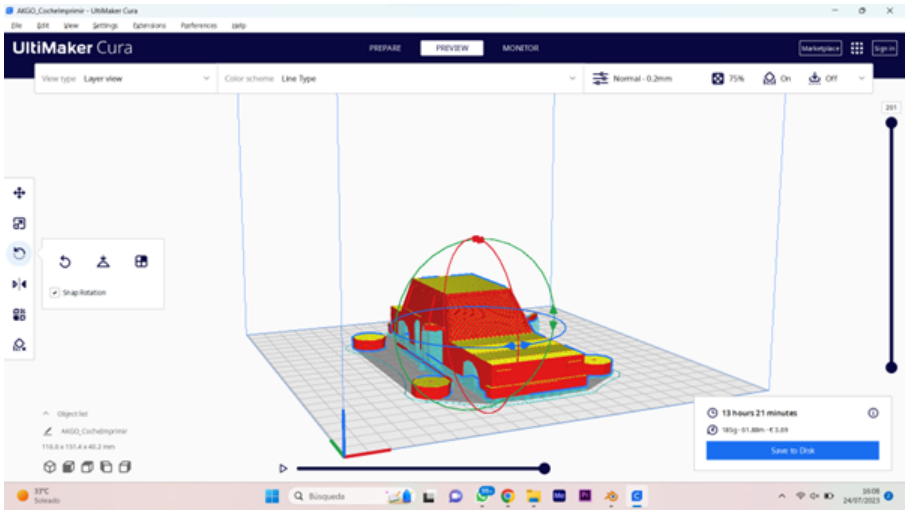
Esta sería la primera impresión que tendríamos del programa CURA una vez importado el modelo. Podemos observar que para mover el modelo tenemos las mismas herramientas que en cualquier otro editor 3D. También a la derecha podemos observar las opciones mencionadas en el párrafo anterior.

También puedes agregar estructuras de soporte desde la pestaña "Soporte" (Support) si tu modelo lo requiere porque hayamos dejado alguna pieza alejada del cuerpo principal o no hayamos hecho caso a la pauta de no dejar partes "flotantes". Revisa la vista de capas para asegurarte de que todo esté en orden. Tras esto, nos aparecerá una estimación de tiempo y si hemos ajustado el filamento o resina que vamos a usar y su precio. Si alguna de estas dos opciones no fuera correcta, podríamos modificar los parámetros de modo que estas dos bajen (tiempo y precio). Finalmente, guarda el archivo G-code. Este archivo contiene las instrucciones específicas para tu impresora y es el que utilizarás para la impresión.



Con el *slice* ahora podremos ver que (efectivamente) se nos calcula el tiempo y el precio (siempre que tengamos este precisado en preferencias). También podemos consultar una estimación precisa de cuánto tiempo ocupará cada proceso de la impresión detallado de forma sencilla.

No obstante, antes de imprimir la pieza que deseas, podrías probar a imprimir una versión más pequeña y comprobar si ésta ha salido con algún desperfecto que puedas corregir antes de imprimir la versión completa.



Comprobando defectos en el apartado de preview con el anterior modelo, hemos podido comprobar que la forma óptima de imprimir el modelo era en horizontal, creándose así solo soportes para los retrovisores y facilitando el trabajo post impresión.

Antes de terminar este punto, nos gustaría resaltar que no todo es modelar y preparar el diseño para imprimirlo, también existe un proceso importante post impresión que requerirá tiempo y dedicación para que el modelo quede exactamente como lo tenemos en el ordenador. Las impresoras nunca van a ser 100% precisas pues, como ya hemos comentado, será necesario imprimir soportes que hará falta quitar a mano y dejarán rebabas e imperfectos que se sumarán a otros causados por pequeños defectos que toda impresora tiene. Por eso es importante saber que, a más nivel de detalle que necesites, más trabajo de lijado post impresión y cuidado necesitarás.

Con todo lo visto hasta ahora, ya tendríamos generado un archivo básico para poder imprimir en 3D.

Proceso de impresión y consideraciones finales

Impresión del modelo

A continuación, abordaremos el proceso de impresión propiamente dicho y algunas consideraciones finales para obtener resultados de calidad y garantizar la seguridad en la impresión 3D.

Con el modelo 3D preparado y habiendo seguido los pasos definidos en el punto anterior, el proceso de impresión puede comenzar. Durante la impresión, la impresora deposita capas sucesivas del material seleccionado para construir el objeto en 3D. Es esencial supervisar la impresión para detectar problemas potenciales y garantizar un resultado exitoso.

Sobre todo, si utilizamos una impresora de resina, debemos tener en cuenta que la impresión 3D involucra el uso de materiales y procesos específicos que pueden presentar riesgos potenciales si no se manejan adecuadamente. Es importante seguir las pautas de seguridad proporcionadas por el fabricante de la impresora y tomar medidas para evitar lesiones o daños durante el proceso de impresión.

Conclusiones finales

La impresión 3D ha revolucionado la fabricación y ha abierto un mundo de posibilidades en diversas industrias. Con el conocimiento de los formatos de exportación, el software de modelado y las consideraciones de impresión adecuadas, se pueden crear objetos 3D personalizados y funcionales que antes eran difíciles de imaginar.

Saber más

Siempre podrás consultar el completo artículo de Adam Jorquera Ortega que nos habla más en profundidad sobre la impresión 3D y el modelado exclusivamente dedicado a este objetivo, "***Fabricación digital: Introducción al modelado e impresión 3D***"

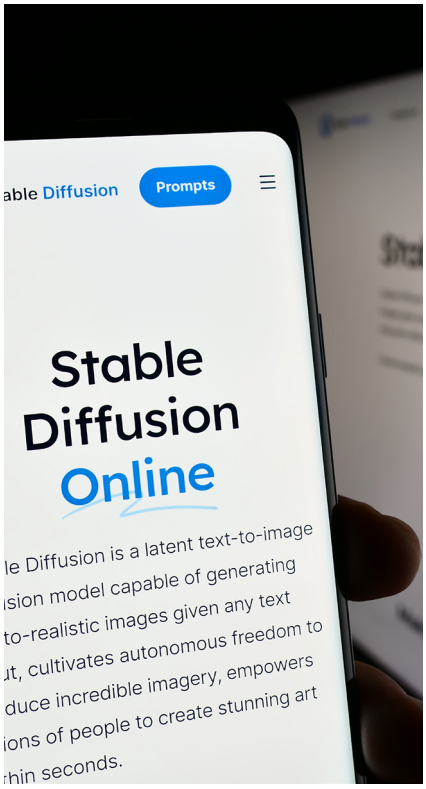
e.digitall.org.es/fabricacion-digital

Creación de
contenidos digitales

Nivel C2 3.1 Desarrollo de contenidos

Requisitos e instalación local de herramientas de inteligencia artificial para la creación de vídeo y audio

Requisitos e instalación local de herramientas de inteligencia artificial para la creación de vídeo y audio



Evaluación de necesidades

En general, la decisión de instalar una IA como *Stable Diffusion* en local dependerá de nuestras necesidades específicas, requisitos de privacidad y seguridad, así como los recursos y capacidades técnicas disponibles. Debemos tener en cuenta que este proceso va a tener unos requisitos de hardware, que comentaremos posteriormente, pero, además, trabajar en local nos hace a los usuarios responsables del mantenimiento del sistema, así como de sus actualizaciones y pudiera ocurrir que, si no tenemos una máquina demasiado potente tengamos problemas de escalabilidad, es decir, que no tengamos tanta capacidad para manejar datos como nos proporciona la misma herramienta en la nube.

Sin embargo, este proceso tiene también algunas claras ventajas, entre las que destacamos:

- Vamos a tener un **control total sobre nuestros datos**. Esto va a ser especialmente importante en casos donde la privacidad y la confidencialidad sean fundamentales, ya que los datos no salen de nuestro entorno local.
- Dispondremos de una mayor velocidad y latencia reducida. Al ejecutar la IA en local, evitamos la dependencia de una conexión a internet para el procesamiento de datos, así como posibles fallos por caídas en la red o limitaciones en el ancho de banda que tengamos contratado. Por tanto, la IA puede responder de manera más rápida, lo que es beneficioso en aplicaciones que requieren tiempos de respuesta rápidos.
- Mayores grados de personalización y ajuste. Nos vamos a encontrar con una mayor flexibilidad a la hora de personalizar y ajustar el modelo a nuestras necesidades específicas. Podremos adaptar parámetros del sistema o entrenar a la IA con nuestros propios datos, lo que nos proporciona una mayor adaptabilidad y un mejor rendimiento en función de nuestros objetivos. De hecho,

si tenemos los suficientes conocimientos, vamos a poder modificar el código fuente de nuestra versión local de *Stable Diffusion* para adaptarlo a nuestros requisitos particulares.

Requisitos de instalación

La instalación de *Stable Diffusion* en local presenta unos requisitos en términos de *hardware*, así como de recursos computacionales.

Por un lado, no tenemos requisitos de procesador, pero vamos a necesitar un disco duro SSD (recomendado) de más de 256 Gb, con al menos 25 Gb libres, así como un mínimo de 8 Gb de memoria RAM. Por otro lado, se recomienda una tarjeta gráfica dedicada NVIDIA con, al menos, 2 Gb de VRAM para mostrar resultados de manera más ágil y precisa con capacidad suficiente para poder usar el ordenador de forma normal.

Lógicamente, cuanto más potente sea la máquina en la que trabajemos, más rápido obtendremos nuestros resultados. Se calcula que *Stable Diffusion* instalado en local solo necesita 5 segundos para generar una imagen de 512x512 pixeles usando una NVIDIA 3060 de 12GB.

Tutorial de instalación

En la actualidad, se ha simplificado bastante el proceso de instalación local de *Stable Diffusion*. Debemos empezar por ir a la página de GitHub de *Stable Diffusion UI*, donde indicaremos en qué sistema operativo estamos trabajando para bajarnos un programa de instalación convencional (Figura 1).

PÁGINA DE GITHUB DE STABLE DIFFUSION UI
e.digitall.org.es/diffusion

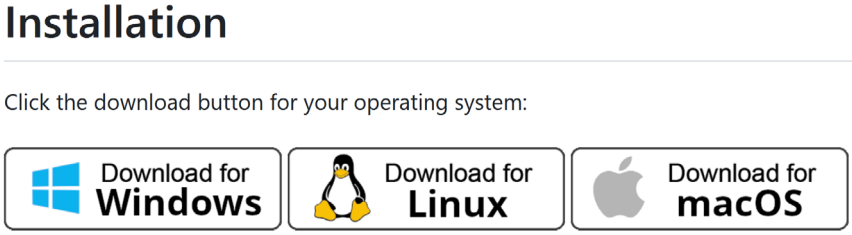


Figura 1. Opciones de selección de sistema operativo para la instalación local de *Stable Diffusion*.

Después solo tendremos que ejecutar el instalador y aceptar el acuerdo de licencia para instalar *Stable Diffusion* en nuestro PC. La ruta de enlace sugerida en la instalación es: ***C:\EasyDiffusion*** (Figura 2).

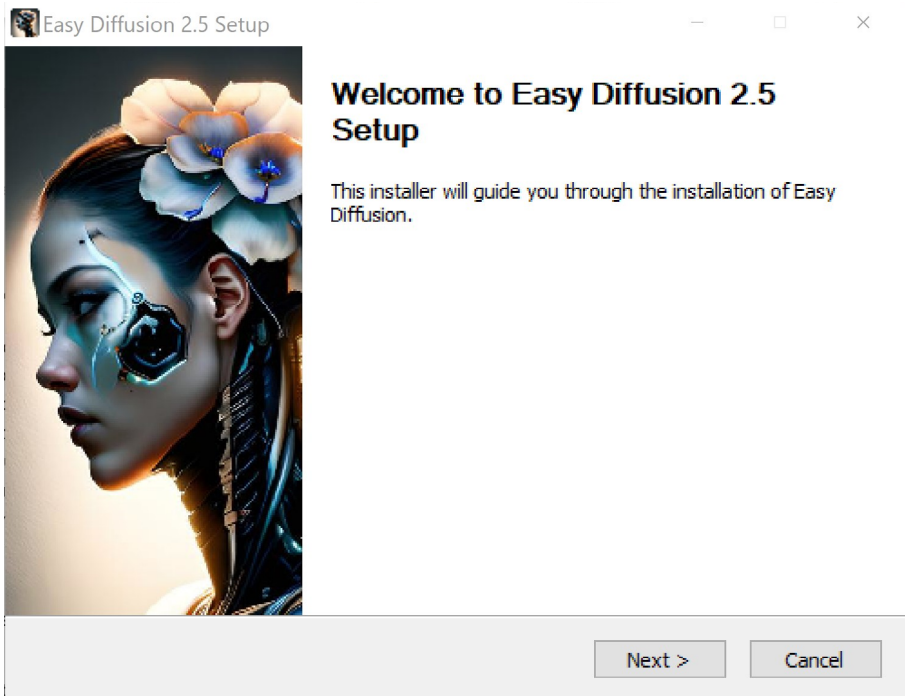


Figura 2. Visión del programa de auto instalación de *Stable Diffusion*.

Una vez finalizada la instalación podremos ejecutar el programa (Start Stable Diffusion UI.cmd) y se nos abrirá una ventana de CMD, en la que veremos que se siguen descargando componentes necesarios para el correcto funcionamiento del sistema.

Una vez descargados e instalados el resto de los archivos necesarios, se nos abrirá la interfaz de la aplicación en nuestro navegador predeterminado en la dirección ***http://localhost:9000/***. Si estás en Windows, probablemente salte un aviso del Firewall (Figura 3).

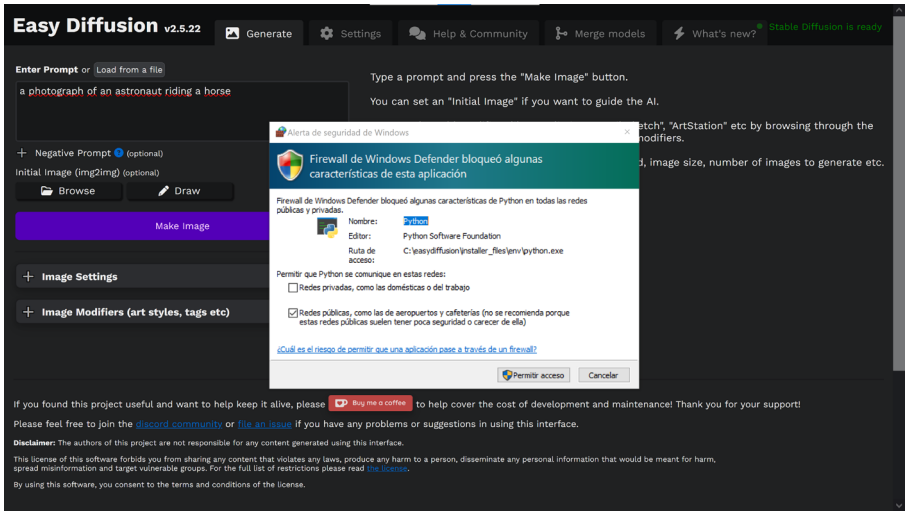


Figura 3. Aspecto del navegador en Windows la primera vez que se ejecuta la instalación local de *Stable Diffusion*.

Una vez establecidos los permisos del firewall, podremos operar con nuestra instalación local de *Stable Diffusion* sin problemas. Podemos ver ejemplos de uso de esta instalación local en el vídeo:

USO LOCAL DE HERRAMIENTAS DE INTELIGENCIA ARTIFICIAL PARA LA CREACIÓN DE IMÁGENES A PARTIR DE INFORMACIÓN TEXTUAL

e.digitall.org.es/A3C31C2V08

Saber más

Si deseas saber más acerca de esta instalación de Stable Diffusion en tu propio entorno local, te invitamos a explorar su página oficial en GitHub. Además, al tratarse de un software de código abierto, existen en la actualidad otras versiones en GitHub para instalar Stable Diffusion de forma local. Por último y también por este mismo motivo, puede que incluso cuando leas esto todas estas versiones que ahora mismo están disponibles ya hayan sido sustituidas por programas más actuales.

e.digitall.org.es/diffusion

DigitAll

Creación de contenidos digitales

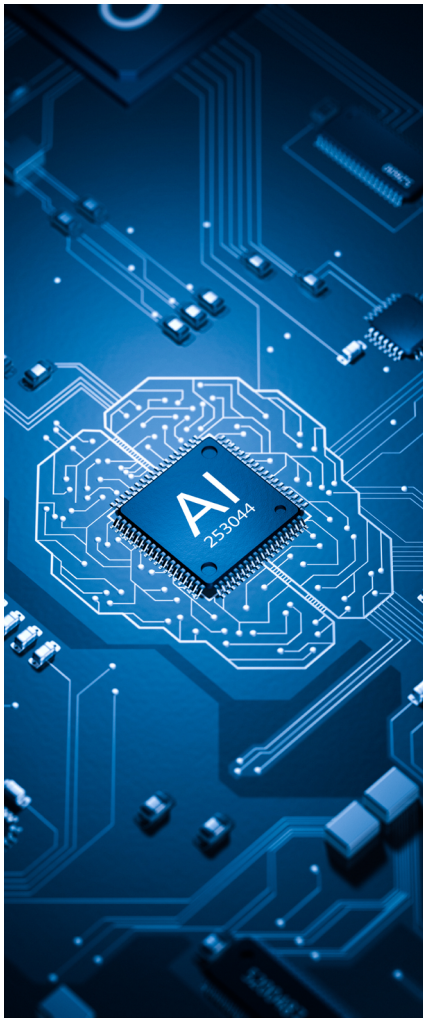
3.2 INTEGRACIÓN Y REELABORACIÓN DE CONTENIDO DIGITAL

Creación de
contenidos digitales

Nivel C2 3.2 Integración y reelaboración de contenido digital

Herramientas de inteligencia artificial para la toma de decisiones

Herramientas de inteligencia artificial para la toma de decisiones



El **aprendizaje automático** es un tipo de inteligencia artificial basada en datos e incluye un conjunto de técnicas que permiten que los ordenadores y los microcontroladores aprendan a realizar unas tareas determinadas. La integración de estas técnicas en sistemas *hardware-software* enriquece las reglas informáticas y libera a los programadores de escribir la lógica de un programa que haga que a una determinada entrada **x** le corresponda la respuesta **y**. Estos métodos actúan como **cajas negras** que reciben una entrada **x** y devuelven una salida **y**. Estos métodos pueden guiar algunas de las acciones del sistema al introducir una cierta *inteligencia* en él.

Suponemos que nuestro sistema hardware-software tiene un vector de variables de entrada **x**, por ejemplo, recogidos por un conjunto de sensores, y el sistema tiene que decidir una variable respuesta **y**. Esta variable puede recoger la respuesta para un conjunto de actuadores o un dato interno necesario para seguir procesando. La clasificación de los problemas de aprendizaje automático se basa en dos criterios: I) la posibilidad o no de conocer la variable respuesta (**y**), y II) la tipología de esta variable. Se distingue la siguiente tipología de las variables:

- **Variables cuantitativas**. Los datos toman valores numéricos. A su vez se distingue entre **variables continuas** que toman valores en un intervalo de números reales o **variables discretas**, que toman un número finito de valores numéricos.
- **Variables ordinales**. Los datos expresan relación de orden entre las observaciones. Por ejemplo, podemos considerar un *ranking*.
- **Variables cualitativas**. En este caso se expresa una cualidad de un objeto. Estas variables solo pueden tomar un conjunto de valores que no miden ninguna magnitud determinada. Un ejemplo son las variables **binarias** en las que solo se pueden tomar dos valores y expresan la presencia/ausencia de una característica.

Una taxonomía de los algoritmos de aprendizaje automático se basa en la naturaleza de las variables del sistema **x** e **y**. Se distingue entre **aprendizaje supervisado** en el cual partimos de un conjunto de datos etiquetado previamente, es decir, conocemos un conjunto de datos (**x**i,**y**i). El aprendizaje **no supervisado** parte de datos no etiquetados previamente, esto es, exclusivamente se dispone de la información (**x**i). Dentro del aprendizaje no supervisado aparecen los problemas de ***clustering***, **reducción de la dimensionalidad** y **detección de *outliers***. El primero determina patrones del sistema. El segundo reduce el tamaño de los datos **x** generados, intentando perder el mínimo de información. Por ejemplo, si los sensores realizan mediciones en tiempo real pueden generar grandes cantidades datos que tiene que ser transmitidos. Un ejemplo notable es la transmisión de imágenes por los satélites. Si los dispositivos tienen una capacidad de comunicación reducida es entonces recomendable aplicar estas técnicas. La detección de *outliers* permiten reconocer que el estado del sistema *software-hardware* está en un estado (patrón) diferente a los a priori planificados para su funcionamiento y por tanto algún tipo de alarma o notificación se debe enviar.

En aprendizaje automático aparecen tres tipos de problemas básicos:

1 | **Clustering**. En este problema se buscan patrones dentro de los datos. Esto es, encontrar subconjuntos de observaciones que son similares entre sí. Matemáticamente se formularía como determinar las etiquetas y de los datos de modo que si dos observaciones i y j son similares le asociemos la misma etiqueta, Yi=Yj. Una dificultad de este problema es que no se parte de un conjunto previo de etiquetas del cual aprender. Un ejemplo de este tipo de problemas es la agrupación de datos. Esto permite identificar un conjunto finito de patrones en el que se puede encontrar el sistema.

2 | **Regresión**. En este problema se predice el valor de una variable continua **y** conocida el valor de la variable **x**. En los contextos de regresión la variable **x** se le denomina regresor o variable explicativa. Estos sistemas estiman funciones de regresión **y**=f(**x**) y permitirían por ejemplo dar el valor **y** de cierto actuador para el estado actual del sistema **x**.

3 | Clasificación. En este problema se debe predecir el valor de la variable cualitativa **y** a partir de la variable **x**. En clasificación la variable **x** se denomina **características** o **atributos** mientras que **y** se denomina **etiqueta** o **clase**. Por ejemplo, en el coche autónomo se basa en tres pilares: *IoT*, técnicas de aprendizaje automático aplicado a *Big Data* y conexión a internet en tiempo real. En la construcción de los *ojos* del vehículo se requiere resolver el problema de clasificar las imágenes que rodean al vehículo y poder así determinar qué tipo de objeto está delante, detrás o a los lados del vehículo.

La Figura 1 resume esta clasificación de los modelos de aprendizaje automático. Para cada tipo de problemas se han propuesto multitud de algoritmos. No existe un método que sea mejor que otro para todas las circunstancias. Por lo tanto, es necesario disponer de una *caja de herramientas* con las que poder ensayar diferentes soluciones. Por ejemplo, el algoritmo *K-means* es rápido y funciona bien para problemas de *clustering* en los que los patrones se pueden *separar linealmente* (mediante hiperplanos). En otras situaciones este algoritmo puede no funcionar bien y por tanto habría que recurrir a técnicas alternativas como algoritmos basados en densidad (DBSCAN). Otro ejemplo son las **redes neuronales profundas** que muestran un alto rendimiento en todos estos problemas. Sin embargo, para ciertos dispositivos, pese a su eficacia, el coste computacional las puede hacer inaplicables.

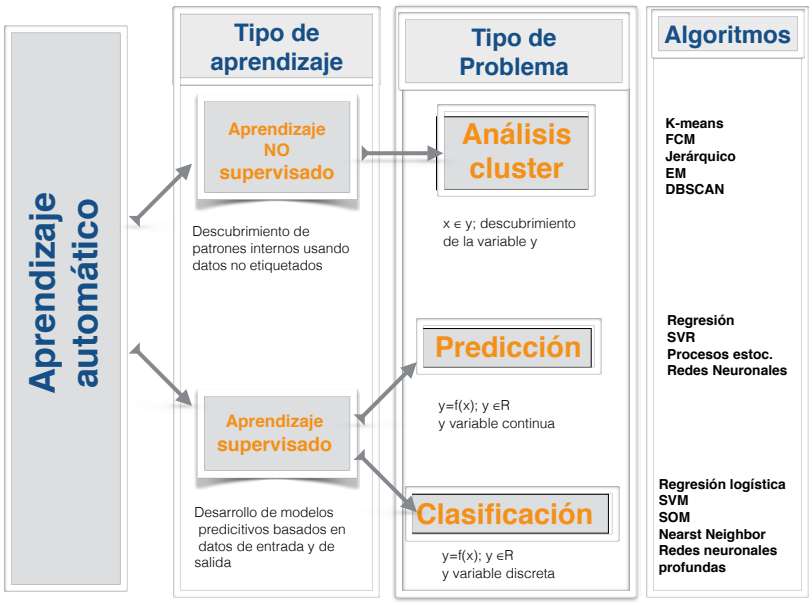


Figura 1. Clasificación de problemas de aprendizaje automático.

Estas técnicas emplean un **aprendizaje inductivo**, ya que a partir de la observación y el análisis de ejemplos concretos se desarrollan modelos que explican dichos datos y que permiten llevar a cabo una generalización. El modelador debe **entrenar** los modelos sobre un conjunto de datos disponibles y los modelos entrenados son introducidos en los sistemas.

La Tabla 1 muestra algunas de las herramientas más populares para implementar modelos de aprendizaje automático. La elección de la herramienta depende de las necesidades y preferencias del proyecto.

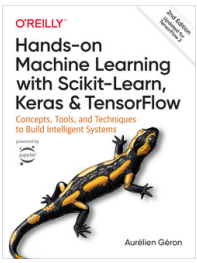
Tabla 1. HERRAMIENTAS PARA CONSTRUCCIÓN DE MODELOS DE APRENDIZAJE AUTOMÁTICO

Nombre	Características	Desarrollador
TensorFlow	Colección herramientas para la creación, entrenamiento e implementación de modelos de aprendizaje automático. Emplea como lenguaje de programación Python y C++.	Google
scikit-learn	Es una biblioteca de aprendizaje automático de código abierto para Python.	
PyTorch	Framework popular de aprendizaje profundo. Ofrece una interfaz más flexible que TensorFlow.	Facebook
Keras	Keras es una biblioteca de Redes Neuronales de código abierto escrita en Python. Es capaz de ejecutarse sobre TensorFlow.	
Microsoft Azure ML	Plataforma en la nube para el ciclo completo de preparación de datos, implementación modelos de aprendizaje automático y monitorización en producción.	Microsoft

Saber más

Un libro excelente para integrar en Python estas técnicas es ***Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow***.

e.digitall.org.es/hands-on



Creación de
contenidos digitales

Nivel C2 3.2 Integración y reelaboración de contenido digital

Herramientas de desarrollo de robots programables

Herramientas de desarrollo de robots programables

Los robots programables han ganado un gran protagonismo en los últimos años, gracias a su capacidad para automatizar tareas y mejorar la eficiencia en diferentes ámbitos, como la industria, la educación o el hogar. Diseñar y programar robots no es tarea fácil, ya que requiere de conocimientos especializados en diferentes áreas, como electrónica, mecánica o informática.

Sin embargo, existen diversas herramientas y plataformas que facilitan este proceso y permiten a cualquier persona, independientemente de su nivel de experiencia, desarrollar robots de manera sencilla y rápida.

La principal herramienta sobre la que se trabaja son las denominadas **placas programables**, como **Arduino** o **Raspberry Pi**. Estas placas son pequeños dispositivos que cuentan con un microprocesador de bajo consumo conectado a diferentes componentes, como módulos de memoria RAM, conexión para almacenamiento de datos a través de una tarjeta de memoria, componentes de entrada-salida (puertos USB, puertos Ethernet, módulos Wi-Fi y Bluetooth, pines de conexión GPIO, etc.) e incluso pequeñas tarjetas gráficas o procesadores de imagen. Estas placas pueden conectarse a diferentes sensores y actuadores y controlar su funcionamiento a través de programas escritos en diferentes lenguajes de programación.

Una vez se tiene la placa programable deseada, otra herramienta fundamental es el **lenguaje de programación**. Algunos de los lenguajes más populares son **C, C++, Python** y **Java**. Estos lenguajes permiten escribir código que indique al robot cómo debe funcionar y cómo debe interactuar con el mundo exterior. Cada placa soportará un conjunto diferente de lenguajes de programación:

- Por ejemplo, las placas Arduino utilizan un lenguaje de programación propio basado en C++ y denominado Arduino. No obstante, existen herramientas de programación que permiten el uso de otros lenguajes

alternativos como por ejemplo C, C# o Python, que posteriormente se convierten en Arduino antes de ser enviados a la placa.

- Por su parte, la placa Raspberry Pi dispone de un sistema operativo propio basado en Linux denominado Raspbian. Por ello, la placa soporta numerosos lenguajes de programación como C, C++, Python, Java, Ruby, Perl, etc.

Saber más

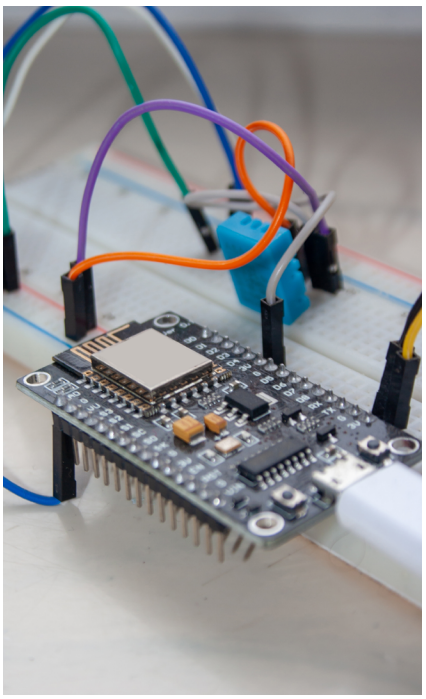
Aunque Raspbian es el sistema operativo más conocido y utilizado para Raspberry Pi, existen otra gran variedad de sistemas operativos compatibles. Por ejemplo: Windows 10 IoT Core, Ubuntu Core/Desktop/Server, Raspbian, OSMC, Kali Linux, etc

Además de los lenguajes de programación, existen diferentes **aplicaciones de desarrollo y simuladores** que permiten diseñar y simular el comportamiento de los robots antes de construirlos. Estas herramientas complementan a los lenguajes de programación incrementando sus posibilidades y añaden capacidades de simulación. Los **simuladores** son especialmente útiles para probar diferentes escenarios en un entorno virtual y optimizar el funcionamiento del robot sin tener que esperar a construirlo físicamente y probarlo en el mundo real. Gracias a estas aplicaciones se logra evitar daños o errores durante el desarrollo del robot, los cuales pueden ser muy costosos, ya que conllevaría volver a la etapa de diseño, programación y construcción del robot.

Saber más

Un simulador sencillo y gratuito para Arduino es **SimulIDE**, disponible en: *simulide.com*. Este simulador está disponible para Windows, MacOS y Linux.

El diseño y la programación de robots requieren de una alta combinación de conocimientos y habilidades específicos, pero gracias a la aparición de ciertas herramientas de desarrollo para un público general, **cualquier persona puede aprender a crear su propio robot programable**. Además, algunas de estas herramientas permiten programar los robots sin necesidad de escribir código o saber programar. Por ejemplo, para Arduino existe la herramienta ArduBlock.



ArduBlock es una herramienta que permite programar en Arduino utilizando bloques gráficos, como si de piezas de puzles se tratara.

Esta herramienta permite elaborar pequeños programas sin necesidad de tener experiencia previa en programación. Actualmente, esta herramienta se encuentra accesible en: ***ardublock.ru/en***. La Figura 1 muestra un ejemplo de un proyecto elaborado en ArduBlock.

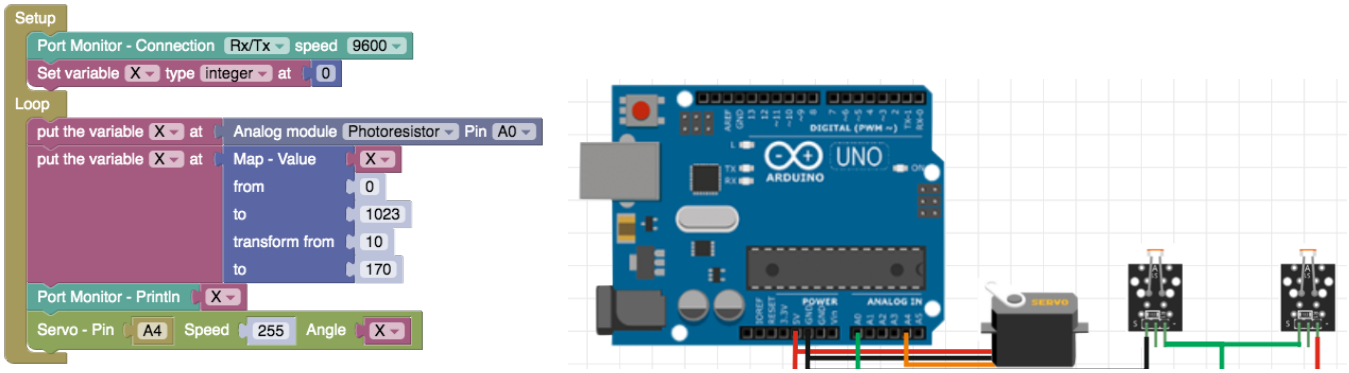


Figura 1: Ejemplo de un proyecto en ArduBlock para gestionar un servomotor que sigue la trayectoria del sol mediante el uso de dos fotorresistores.
Fuente: ardublock.ru

Saber más

Scratch es una herramienta de programación mediante bloques desarrollada por el MIT y orientada a la iniciación a la programación. Esta herramienta está enfocada principalmente en la programación de videojuegos. Existe una modificación de Scratch denominada **S4A** (*s4a.cat*), la cual permite programar en Arduino utilizando el lenguaje Scratch.

También existen los denominados **kits de desarrollo** o de robotización, que son pequeños kits comerciales que incluyen todos los componentes necesarios para construir y programar un robot. Algunos kits incluyen las propias placas programables, un amplio conjunto de sensores y actuadores y todo el cableado necesario, así como herramientas de programación específicas. Estos kits suelen venir con guías y tutoriales que facilitan el proceso de montaje y programación.

La propia web del proyecto Arduino ofrece un kit de iniciación denominado **Arduino Starter Kit** (*e.digitall.org.es/arduino*). No obstante, existen multitud de kits para Arduino en múltiples tiendas de electrónica. Desde kits para niños, como kits para

entusiastas de la tecnología o gente que quiere iniciarse. También hay una gran variedad de kits para Raspberry Pi, algunos de los cuales permiten construir tu propio ordenador personal. Por ejemplo, el **Raspberry Pi 400 Personal Computer Kit** (*e.digitall.org.es/raspberry*).

Saber más

Otra herramienta importante son los **foros y comunidades en línea**, donde se pueden encontrar tutoriales, ejemplos y consejos de otros usuarios que también estén interesados en el desarrollo de robots. Estas comunidades suelen ser una fuente inagotable de inspiración y aprendizaje.

En resumen, existen diferentes herramientas y paradigmas que facilitan el desarrollo de robots programables. Desde los lenguajes de programación de propósito general aplicados a la programación de estas placas, las aplicaciones de desarrollo, los simuladores y los kits de desarrollo. Cada uno de estos enfoques tiene sus propias ventajas y desventajas, y es importante elegir la herramienta adecuada en función de las necesidades del proyecto.

DigitAll

Creación de contenidos digitales

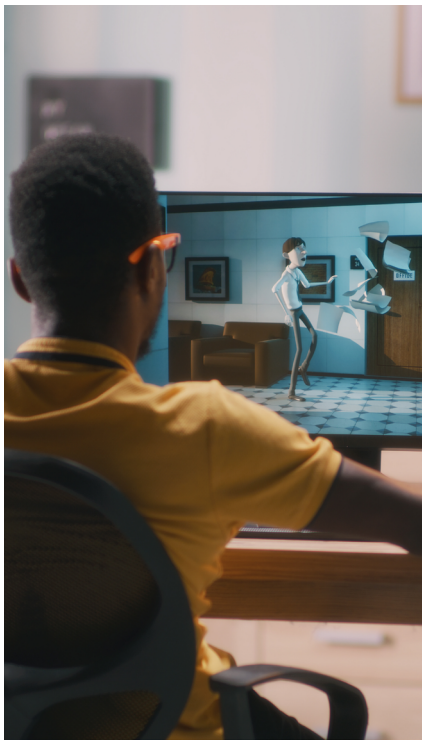
3.3 DERECHOS DE AUTOR Y LICENCIAS DE PROPIEDAD INTELECTUAL

Creación de
contenidos digitales

Nivel C2 3.3 Derechos de autor y licencias de propiedad intelectual

Registrando el copyright y explotando una obra creada con ayuda de la IA

Registrando el copyright y explotando una obra creada con ayuda de la IA



El contenido de este documento te hará reflexionar sobre un tema de suma actualidad.

Al día de hoy, los creadores disponen en internet de herramientas revolucionarias que hacen uso de la inteligencia artificial para la creación de todo tipo de contenido artístico.

Estas herramientas permiten a artistas, músicos, escritores y otros creadores explorar nuevas fronteras y encontrar inspiración en este tipo de herramientas.

En el ámbito musical, permiten componer melodías complejas, armonías bastante atractivas y ritmos hasta ahora no escuchados. Por otro lado, el campo de la pintura puede generar reproducciones fieles de los grandes maestros o nuevas interpretaciones artísticas.

Todos conocemos ya el éxito de la herramienta ChatGPT para generar textos originales. Otros ejemplos son la herramienta DALL·E 2 que genera imágenes a partir de descripciones que se le proporcionan, AIVA, que compone pistas de música originales otro ejemplo es la herramienta AI Time Machine, que permite a los usuarios crear imágenes de una persona en diferentes periodos de tiempo a lo largo de la historia.

Pero qué pasa con todas las obras que se creen usando estas herramientas ¿Es posible obtener el copyright de una obra creada con la ayuda de la "**inteligencia artificial**"?

En la mayoría de los países, **se pueden proteger por medio del Derecho de autor o Copyright las obras originales creadas por un ser humano**.

En las décadas pasadas, algunos creadores necesitaban de la computadora y la utilizaban como un instrumento para crear sus obras, del mismo modo que un escritor utilizaba el bolígrafo y el papel, otro utilizaba un programa de ordenador para escribir su obra, el pintor hacia uso del pincel y el lienzo y el creador de música electrónica necesitaba de un sintetizador. En todos estos casos primaba siempre la creatividad del creador.

Sin embargo, la revolución tecnológica que está teniendo lugar en los últimos años, impulsada, en parte, por el desarrollo del "**software de aprendizaje automático**", nos obliga a repensar la interacción entre las computadoras y el proceso creativo cuando interviene el mencionado software.

Saber más

El software de aprendizaje automático, una de las formas de la IA, es un programa informático que puede aprender a partir de los datos que se le van introduciendo al mismo software. Con la introducción de nuevos datos evoluciona y toma nuevas decisiones que pueden ser dirigidas o autónomas.

Cuando un creador, por ejemplo, un pintor, escritor o compositor, utiliza un software de aprendizaje automático para desarrollar su obra, en el proceso creativo el propio software va aprendiendo a partir de la información que el creador o programador va introduciendo y a partir de esos datos, comienza a tomar decisiones independientes que dan como resultado una nueva obra de arte. Lo que ocurre en realidad en estos casos es que, **si bien el creador define algunos parámetros en el proceso creativo, la obra es generada por el programa informático**.

En casos como el anterior, podríamos concluir que el programa informático ya no es una herramienta que utiliza el creador, como en los casos que se mencionaron anteriormente, sino que toma decisiones asociadas al proceso creativo sin que intervenga el creador, por lo que se podría entender que el creador es el propio programa y no el ser humano. Teniendo en cuenta la legislación española, una obra creada utilizando un **software de aprendizaje automático**, que es un subgrupo de la Inteligencia Artificial no podría registrarse por Derecho de Autor.

En el caso de que las obras creadas con la intervención de la Inteligencia Artificial no se pudieran proteger mediante el Derecho de Autor por considerarse que no han sido creadas por el ser humano cualquier persona podría utilizarlas libremente sin cometer plagio, lo que sería un problema muy serio para empresas del sector que venden estas obras. Por ejemplo, pensemos en la productora de películas que invierte mucho dinero en desarrollar un sistema que genere música para sus películas y posteriormente la ley no le permitiera protegerlas,

cualquier persona en el mundo podría utilizarla libremente, ocasionando un gran perjuicio económico a la empresa.

En la actualidad es un problema que no está resuelto, no obstante, hay indicios de que la legislación de numerosos países no es favorable al derecho de autor que no se aplica al ser humano, como ya se han pronunciado en Estados Unidos y Australia y muchos países europeos. Sin embargo, en otros países como Hong Kong, la India, Irlanda, Nueva Zelandia y Reino Unido estarían dispuestos a concederle la autoría a la persona que realiza los arreglos necesarios para la creación de la obra".

En un futuro cercano, con el avance de la informática, cuando el uso de la Inteligencia Artificial esté más generalizada y la mayoría de los creadores hagan uso del software de aprendizaje automático, los ordenadores producirán obras creativas cada vez mejores y podría ser difícil distinguir entre una obra de arte hecha por un ser humano y la realizada por la máquina y entonces habrá que decidir qué tipo de protección se le debería conceder a las obras creadas con poca o ninguna intervención humana.

Desde el punto de vista de la explotación comercial de la obra, lo sensato sería conceder el derecho de autor a la persona que hace posible el funcionamiento del software de aprendizaje automático, de esta forma, por un lado, se garantizaría la continuidad de la industria creativa como la sostenibilidad del artista que vive de la explotación de sus obras, y por otro, que las empresas sigan invirtiendo en la tecnología, con la seguridad de saber que obtendrán rendimientos de su inversión.

Saber más

En este artículo encontrarás información sobre Inteligencia Artificial (IA) explicada de forma muy sencilla y clara:

e.digitall.org.es/inteligencia-artificial

DigitAll

Creación de
contenidos digitales

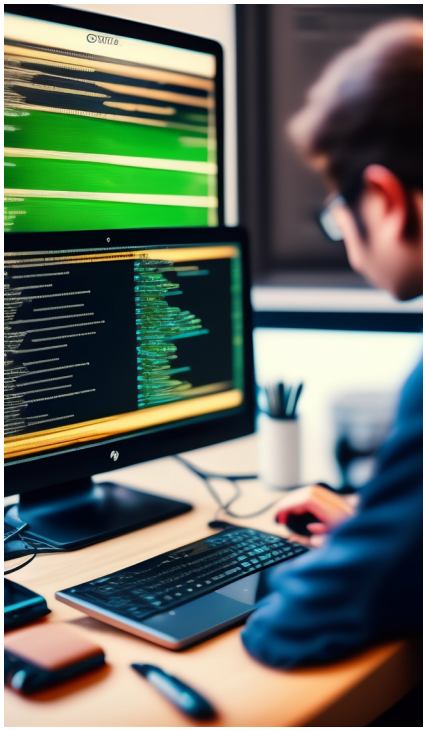
3.4
PROGRAMACIÓN

Creación de
contenidos digitales

Nivel C2 3.4 Programación

Paradigmas de programación. Principios generales

Paradigmas de Programación. Principios Generales



En la actualidad existen distintos estilos que definen cómo abordar el diseño y la escritura de un programa, lo que se conoce como *paradigmas de programación*. Cada uno de ellos posee características (reglas, técnicas y principios) específicas que lo diferencia de los demás, haciéndolo más o menos apropiado para resolver distintos tipos de problemas en función de sus necesidades. Los paradigmas de programación más utilizados son el imperativo, el orientado a objetos y el funcional. Es importante conocer las ventajas y los inconvenientes de cada uno de ellos con el fin de elegir el más adecuado para abordar un problema, ya que puede hacer que el código sea más legible, mantenible y escalable. En este sentido, los lenguajes de programación juegan también un papel significativo, ya que están diseñados para soportar uno o más paradigmas de programación. Esto significa que ciertos paradigmas pueden ser más fáciles de implementar o más efectivos en ciertos lenguajes de programación que en otros. Por ejemplo, la programación orientada a objetos es ampliamente utilizada en lenguajes como Java y C++, mientras que para la programación funcional se usan lenguajes como Haskell y Lisp. Además, algunos lenguajes de programación son específicos de un único paradigma, como los lenguajes Pascal y C, que se utilizan principalmente para la programación imperativa. Otros lenguajes son multiparadigma, lo que significa que permiten el uso de varios paradigmas diferentes en el mismo programa. Por ejemplo, Python es un lenguaje de programación multiparadigma que admite programación imperativa, orientada a objetos y funcional. La capacidad de un lenguaje de programación para soportar múltiples paradigmas puede brindar a los desarrolladores la flexibilidad y el poder para elegir el enfoque de programación más adecuado para su proyecto.

A continuación, se describen los tres paradigmas de programación más comunes, junto con el tipo de aplicaciones en el que son más indicados:

- La **programación imperativa** se caracteriza por poner el foco en "cómo" se resuelve el problema, por lo que los programas consisten en la descripción precisa y detallada de la secuencia de pasos que hay que realizar para resolver un problema. La idea es hacer uso de las instrucciones para ir ***modificando el estado del programa***. Para ello, se cuenta con las variables, que son los elementos que permiten almacenar los datos (tanto de entrada como de salida), y con los bucles y los condicionales, que controlan el orden de ejecución de las instrucciones. Además, los subprogramas (procedimientos o funciones) se usan para modularizar y reutilizar el código. La gran ventaja de este paradigma es que el modelo bajo el que se basa es muy intuitivo, ya que es muy parecido a un "manual de instrucciones paso a paso". Por eso, es el que se suele utilizar para enseñar y aprender a programar. Sin embargo, a la hora de resolver problemas más complejos, el código puede hacerse demasiado largo, lo que puede dificultar su mantenimiento.
- La **programación orientada a objetos (POO)** considera un programa como una colección de objetos que interactúan entre sí, de forma más parecida a como ocurre en la vida real. Un **objeto** constituye un ejemplar de una **clase**, que es la "plantilla" o modelo que define la representación, las propiedades y el comportamiento comunes a un conjunto de objetos. La representación del objeto se realiza por medio de variables que se denominan **atributos**. Por ejemplo, para definir las propiedades comunes de los y las estudiantes de la asignatura de programación, como su nombre, apellidos y calificación alfanumérica ("suspenso", "aprobado", "notable" y "sobresaliente") se podría definir una clase, llamada `Estudiante`, con tres variables (atributos) denominados `Nombre`, `Apellidos` y `Calificación`, respectivamente.

  El comportamiento de los objetos lo definen los **métodos**, que son los procedimientos o funciones que implementan las operaciones para manipularlos y para interactuar con otros objetos. Por ejemplo, la clase `Estudiante` debería

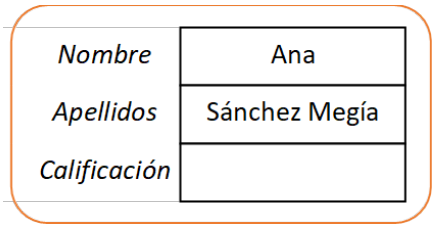


Figura 1. Ejemplo de objeto de la clase *Estudiante*.

contener, al menos, métodos para conocer el nombre, los apellidos y la calificación de un o una estudiante. Existe un método particular, llamado ***constructor***, que es el que se invoca para ***crear un objeto***. Así, para crear el objeto correspondiente a la alumna Ana Sánchez Megía podría invocarse al constructor con el nombre y los apellidos como parámetros. El resultado implica la creación de una "variable" similar a un registro con tres campos como ilustra la Figura 1.

Una vez definida una clase, se pueden crear tantos objetos o instancias como se necesiten.

La POO se fundamenta en los tres pilares siguientes:

- **Encapsulación** se usa para ocultar la representación concreta de los objetos, lo que los protege de usos indebidos. Por ejemplo, que no se pueda asignar una calificación inexistente o que dos calificaciones no se puedan sumar.
- **Herencia** se utiliza para crear subclases, es decir, clases derivadas de otra, que heredan sus propiedades y métodos, pero que pueden tener otros adicionales. Por ejemplo, se podría crear la clase `EstudianteRepetidor`, como subclase de `Estudiante`, para representar los estudiantes que ya cursaron otro año la asignatura, con el atributo específico `AñoRealización`, además de los que hereda (Nombre, Apellidos, Calificación).
- **Polimorfismo**, que permite que los objetos de diferentes clases se comporten de manera similar en función del contexto. Por ejemplo, los objetos de la clase `EstudianteRepetidor` también son objetos de la clase `Estudiante`, por lo que pueden comportarse de ambas formas. Así, pueden formar parte de la lista de todos los estudiantes de la asignatura, formada por objetos de la clase `Estudiante`.

  El polimorfismo dota de gran flexibilidad a los programas, lo que constituye una ventaja de este paradigma; la herencia facilita la reutilización de código, y el encapsulamiento ayuda a prevenir errores. Además, el diseño de clases permite abordar el programa de forma modular, facilitando, además, la

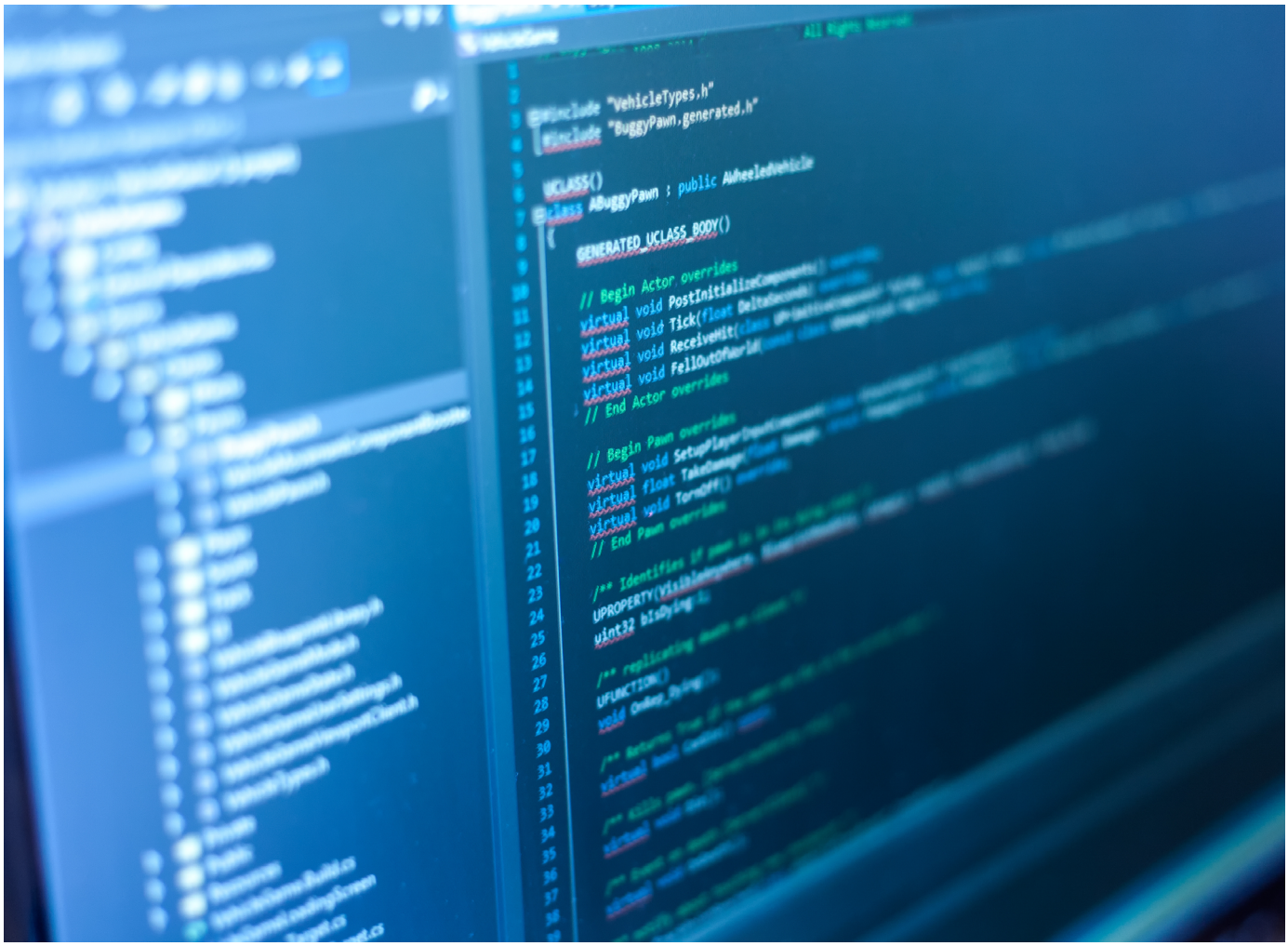
inserción de nuevos objetos y la modificación de los existentes. Por tanto, es un paradigma idóneo para el desarrollo de grandes aplicaciones en equipo. Sin embargo, hay que tener en cuenta que la ejecución de los programas es más lenta y que el diseño de clases puede ser demasiado complicado.

- La **programación funcional** se encuadra en el paradigma de programación declarativa, mediante el que los programas especifican qué quieren conseguir sin describir cómo hacerlo. En el paradigma funcional, los elementos con los que se escriben los programas son las **funciones puras**, es decir, aquellas que para una misma entrada siempre producen el mismo resultado. Al contrario que en el paradigma imperativo o en el orientado a objetos, no interesa modificar el estado del programa, por lo que los datos permanecen **inmutables**: durante la ejecución del programa los datos no se modifican, sino que se van creando otros nuevos. Para ello, es imprescindible que las funciones estén correctamente parametrizadas y que la invocación a las mismas se haga con los valores adecuados. Hay que tener en cuenta que las funciones pueden ser usadas también como parámetros de otra función. Además, las funciones no producen efectos colaterales indeseados. Por otro lado, en lugar de bucles se usa la recursión. La **recursión** ocurre cuando una función se llama a sí misma, por lo que se crea una repetición en el que los valores antiguos permanecen inalterables. Un ejemplo de función recursiva puede ser la que calcula la suma desde 1 hasta n, siendo n su argumento, y que se podría definir así:

```
Suma(1)=1;
Suma(n)=Suma(n-1)+n, n>1
```

De esta forma, `Suma(3)=Suma(2)+3;` pero `Suma(2)=Suma(1)+2`. Como `Suma(1)=1`, ahora se van "deshaciendo" las llamadas, es decir, ahora ya se puede calcular `Suma(2)=1+2=3`; y después, `Suma(3)=3+3=6`. El valor obtenido es el resultado de `1+2+3`.

Las ventajas de la programación funcional son varias: el código suele ser más conciso y expresivo, ya que las funciones suelen ser pequeñas e independientes entre sí, lo que facilita el mantenimiento de los programas. Además, la inmutabilidad de los datos evita la existencia de efectos colaterales, por lo que es más fácil detectar errores y facilita la concurrencia. Teniendo en cuenta que la complejidad de las aplicaciones es cada vez mayor, especialmente por la cantidad de datos que procesan, se hace necesario desarrollar programas que puedan ser ejecutados en varias máquinas a la vez, soportando concurrencia y paralelismo. Por eso, este paradigma está resurgiendo con fuerza en el mundo empresarial e industrial. No obstante, la programación funcional no es sencilla, ya que la recursividad es una técnica compleja, que puede dar a errores graves si no se domina. Por otra parte, el mantenimiento de los programas es complicado y el código es difícilmente reutilizable.



Creación de
contenidos digitales

Nivel C2 3.4 Programación

Depuración en Python. Aspectos generales

Depuración en Python. Aspectos Generales

Introducción

En programación, la necesidad de la creación de código que funcione correctamente y sin errores siempre está presente como objetivo. En este contexto, la depuración se convierte en un aspecto esencial del proceso de programación. La depuración es un procedimiento sistemático que ayuda a detectar, aislar y rectificar errores o "bugs" en un programa, asegurando así tanto su correcto funcionamiento como la generación de un código libre de errores.

La habilidad para depurar programas en Python, al igual que en cualquier otro lenguaje de programación, es una competencia crucial para todo desarrollador. Un código correctamente depurado minimiza la existencia de errores sin detectar, previene fallos o comportamientos imprevistos del programa en su ejecución, y permite a los desarrolladores entender mejor la lógica y el flujo de su código, facilitando así el mantenimiento del software.

Entre las diversas herramientas disponibles para facilitar la depuración en Python, una de las más destacadas es pdb. Pdb, acrónimo de *Python DeBugger*, es el depurador integrado en el lenguaje Python. Este depurador brinda una serie de funcionalidades valiosas que permiten a los programadores recorrer su código paso a paso, inspeccionar variables y evaluar expresiones.

Tipos de errores

Al depurar programas en Python, es probable que nos encontremos con varios tipos de errores. Los errores en Python se clasifican generalmente en tres categorías: errores de sintaxis, excepciones y errores lógicos.

1 | **Errores de sintaxis**: estos ocurren cuando el código viola las reglas de sintaxis del lenguaje Python. Ejemplos de este tipo de error podría ser desde olvidar poner dos puntos (:) al final de una declaración de función o clase, hasta errores de sangría o incluso omitir paréntesis o llaves. Cuando

Python encuentra un error de sintaxis, interrumpe el proceso de interpretación de código y muestra un mensaje de error que indica la línea y la naturaleza del error.

2 | **Excepciones**: estos son errores que ocurren durante la ejecución del programa. Aunque la sintaxis del código puede ser correcta, el programa puede generar un error cuando intenta ejecutar una instrucción. Las excepciones incluyen situaciones como intentar dividir un número por cero, abrir un archivo que no existe, o acceder a una variable no definida, entre otras. Python es muy explícito cuando se producen excepciones y proporciona un rastreo de pila detallado, incluyendo la línea en la que ocurrió la excepción y el tipo de excepción.

⚠ ATENCIÓN

Un rastreo de pila en Python, también conocido como 'stack trace', es una representación de la secuencia de las llamadas de funciones que ha realizado tu programa hasta llegar a un punto determinado. Es como una huella digital de cómo tu programa ha llegado a donde está, generalmente presentado cuando ocurre un error. Si tu programa falla (un error), Python te mostrará la 'ruta' que siguió, indicando las funciones y métodos que se llamaron y en qué orden, para ayudarte a diagnosticar dónde pudo haber surgido el problema.

3 | **Errores lógicos**: estos son los errores más difíciles de detectar y corregir. Los errores lógicos ocurren cuando la lógica o el diseño de un programa es incorrecto, pero el programa se ejecuta sin generar errores de sintaxis o excepciones. Un ejemplo común de un error lógico es un bucle infinito. Aunque el bucle puede ser sintácticamente correcto, si la condición de terminación del bucle nunca se cumple, el programa continuará ejecutándose indefinidamente. Estos errores no son capturados por el sistema de excepciones de Python, por lo que requieren una cuidadosa revisión del código y una comprensión sólida de la lógica del programa para su resolución.

Depurador pdb

El depurador pdb es una herramienta vital para el desarrollo de código en Python. Ofrece una gama de funcionalidades que facilitan el proceso de depuración, permitiendo a los desarrolladores rastrear su código, establecer puntos de ruptura, avanzar paso a paso, y mucho más. Aquí, revisaremos algunos de los conceptos clave para comenzar a depurar con pdb.

1 | **Puntos de ruptura:** los puntos de ruptura son marcadores que puedes establecer en una línea de código específica donde quieres que la ejecución del programa se detenga. Esto es útil cuando sabes que un error ocurre en una sección específica del código, pero no estás seguro de exactamente dónde o por qué. Con pdb, puedes usar la función "`set_trace()`" para establecer un punto de ruptura.

2 | **Ejecución línea por línea:** una vez que la ejecución del programa se ha detenido en un punto de ruptura, puedes usar la función "`step`" (o "`s`" en su forma abreviada) para avanzar línea por línea en el código.

3 | **Inspección de variables:** al detenerse en un punto de ruptura o al avanzar línea por línea, puedes querer ver el valor de una variable específica. Con pdb, puedes hacerlo simplemente ingresando el nombre de la variable en la consola de pdb. También puedes usar la función "`args`" para ver los argumentos de la función actual.

4 | **Continuar la ejecución:** si has detenido la ejecución en un punto de ruptura y has determinado que todo está funcionando correctamente hasta ese punto, puedes usar la función "`continue`" (o "`c`" en su forma abreviada) para reanudar la ejecución hasta el próximo punto de ruptura o hasta el final del programa.

5 | **Salir de pdb:** si necesitas detener la depuración por cualquier motivo, puedes usar la función "`quit`" (o "`q`" en su forma abreviada) para salir de pdb y terminar la ejecución del programa.

Ejemplo de depuración de un programa

Partimos del siguiente código de ejemplo que divide un número por un denominador y luego decrementa el denominador hasta que llega a 0, lo que ocasionará una excepción del tipo ZeroDivisionError.

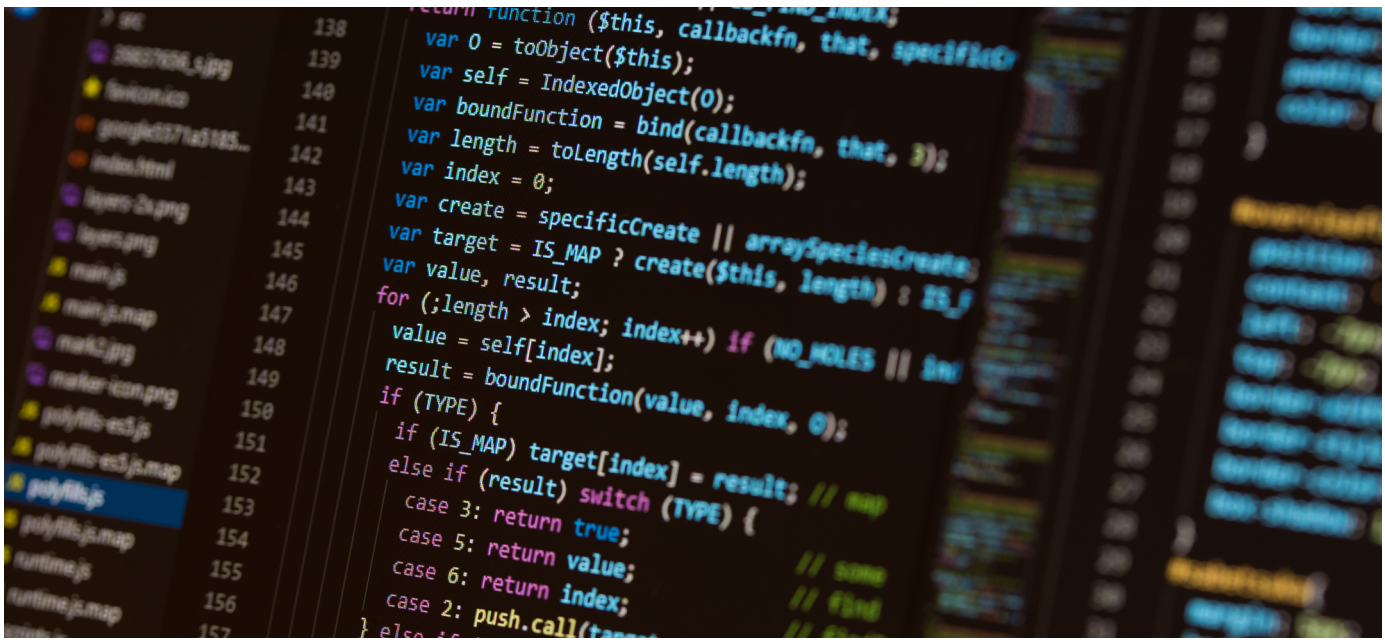
```
def division_decreciente(numerador, denominador):
    while denominador >= 0:
    resultado = numerador / denominador
        print(f"El resultado de dividir {numerador} entre {denominador} es
        {resultado}")
        denominador -= 1

division_decreciente(10, 3)
```

Para depurar el código, vamos a insertar un punto de ruptura en la función antes de la línea que creemos que está causando el problema:

```
import pdb
def division_decreciente(numerador, denominador):
    while denominador >= 0:
        pdb.set_trace()
        resultado = numerador / denominador
        print(f"El resultado de dividir {numerador} entre {denominador} es
        {resultado}")
        denominador -= 1

division_decreciente(10, 3)
```



Una posible simulación de la depuración en la consola de Python podría ser la siguiente:

```
> python3 mi_programa.py
> /ruta/a/mi_programa.py(6)division_decreciente()
-> resultado = numerador / denominador
(Pdb) p denominador
3
(Pdb) c
El resultado de dividir 10 entre 3 es 3.3333333333333335
> /ruta/a/mi_programa.py(6)division_decreciente()
-> resultado = numerador / denominador
(Pdb) p denominador
2
(Pdb) c
El resultado de dividir 10 entre 2 es 5.0
> /ruta/a/mi_programa.py(6)division_decreciente()
-> resultado = numerador / denominador
(Pdb) p denominador
1
(Pdb) c
El resultado de dividir 10 entre 1 es 10.0
> /ruta/a/mi_programa.py(6)division_decreciente()
-> resultado = numerador / denominador
(Pdb) p denominador
0
(Pdb) s
ZeroDivisionError: division by zero
```

En esta simulación, utilizamos el comando "p" para inspeccionar el valor del denominador en cada iteración del bucle. También utilizamos el comando "c" para continuar la ejecución hasta el próximo punto de ruptura, que está dentro del bucle, por lo que se detiene en cada iteración. Finalmente, cuando vemos que el denominador es 0, utilizamos el comando "s" para avanzar a la línea siguiente, lo que provoca el `ZeroDivisionError`. Mediante la depuración, el programador podría percatarse de la línea en la que se está produciendo el error y justo la situación que lo provoca (valores concretos de numerador y denominador).

Saber más

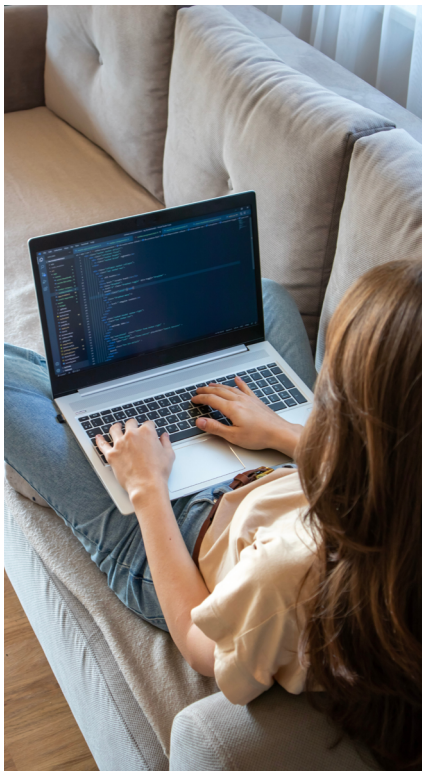
Puedes aplicar tus conocimientos en depuración en Python con pdb a través de la documentación oficial de Python en castellano, en concreto, en *e.digitall.org.es/depuracion*

Creación de
contenidos digitales

Nivel C2 3.4 Programación

Diseño de código en Python. Buenas prácticas

Diseño de código en Python. Buenas prácticas



Introducción

El diseño de código efectivo y eficiente es uno de los principales retos a los que se enfrenta cualquier programador. En este sentido, Python, con su enfoque en la simplicidad y legibilidad, ofrece un medio ideal para desarrollar código que no solo sea funcional, sino que también sea fácil de entender y mantener. La adopción de buenas prácticas de diseño de código es crucial, no solo para mejorar la calidad del software, sino también para facilitar la colaboración y el mantenimiento a largo plazo del código. De hecho, seguir buenas prácticas puede ser considerado una inversión a largo plazo. Puede requerir más tiempo y esfuerzo inicialmente, pero definitivamente vale la pena a largo plazo ya que minimiza los problemas y aumenta la eficiencia.

La aplicación de buenas prácticas reduce la complejidad del código, aumenta su mantenibilidad y facilita la detección y corrección de errores, mejorando la calidad del software.

Buenas prácticas para el diseño de código en Python

A continuación, se discuten algunas de las buenas prácticas más habituales que deberías considerar en el diseño y codificación de programas.

Nombrado descriptivo

Los nombres de las variables, funciones y clases deben ser lo suficientemente descriptivos para indicar su propósito o uso en el programa. Los nombres deben ser entendibles y proporcionar una idea de lo que hace la función o lo que representa la variable. Esta práctica facilita la lectura y comprensión del código, no solo por el propio programador, sino también por otros desarrolladores que puedan trabajar o revisar el código. Por ejemplo, supongamos que estás escribiendo una función para calcular el área de un círculo. En lugar de nombrar la

función de manera genérica como `func1` y `usar x` para la variable del radio, puedes usar nombres más descriptivos:

```
#Nombramiento poco descriptivo

def func1(x):
   return 3.1416 * (x**2)
```

```
#Mismo ejemplo con nombramiento de variables descriptivo

def calcular_area_circulo(radio):
   return 3.1416 * (radio**2)
```

Uso apropiado de comentarios

Los comentarios son una herramienta útil para explicar el propósito o la funcionalidad de una sección de código, especialmente si es compleja o no está claramente indicada por el código en sí. Deberían utilizarse para agregar detalles que no son inmediatamente obvios solo con leer el código. Sin embargo, también es importante no sobrecargar el código con comentarios innecesarios, ya que esto puede hacer que el código sea más difícil de leer.

Ejemplo: Supongamos que estás implementando un algoritmo que ordena una lista de números utilizando el método de la burbuja. Aquí es cómo podrías comentar correctamente ese código:

```
def ordenamiento_burbuja(lista):
   # Iterar sobre cada elemento en la lista
   for i in range(len(lista)):
      # Comparamos el elemento actual con el siguiente
      for j in range(0, len(lista) - i - 1):
         # Si el elemento actual es mayor que el siguiente,
         los intercambiamos
         if lista[j] > lista[j + 1] :
            lista[j], lista[j + 1] = lista[j + 1], lista[j]
```

Aplicación del principio de responsabilidad única

Este principio sugiere que cada función, módulo o clase debe tener una única responsabilidad. En términos más sencillos, deberían hacer solo una cosa, pero hacerla bien. El cumplimiento de este principio hace que el código sea más manejable, fácil de mantener y menos propenso a errores, ya que, si surge un problema, sabes exactamente dónde buscar.

Ejemplo: Supongamos que tienes una aplicación que procesa pedidos de un e-commerce.

```
def manejar_pedido(cliente, producto, cantidad):
    # Verificar disponibilidad del producto
    ...
    # Actualizar inventario
    ...
    # Procesar el pago
    ...
    # Generar factura
    ...
    # Enviar correo de confirmación al cliente
    ...
```

En lugar de tener una función que maneje todo el proceso de pedido, sería mejor tener funciones separadas para cada paso del proceso.

```
def verificar_disponibilidad(producto, cantidad):
    ...
def actualizar_inventario(producto, cantidad):
    ...
def procesar_pago(cliente, monto):
    ...
def generar_factura(cliente, detalles_pedido):
    ...
def enviar_confirmacion_por_correo(cliente, detalles_pedido):
    ...
```

Uso de funciones y abstracción

La abstracción es un método esencial en la programación que se centra en distanciar los detalles técnicos de una parte específica del código de su aplicación práctica. En el lenguaje Python, se pueden emplear las funciones para simplificar operaciones y englobar secciones de código que cumplen con un propósito definido. El uso de la abstracción mediante funciones optimiza la legibilidad del código, su posibilidad de reutilización y facilita su mantenimiento.

Por ejemplo, si estás creando un programa que realiza cálculos matemáticos complejos varias veces, en lugar de escribir la misma lógica de cálculo una y otra vez, podrías abstraer esa lógica en una función. Entonces, en lugar de repetir el mismo código, simplemente llamarías a la función cada vez que necesites realizar ese cálculo. Así, si necesitas cambiar la forma en que se realiza el cálculo, solo tendrías que hacerlo en un lugar

(dentro de la función), en lugar de buscar y cambiar múltiples instancias del mismo código.

> **NOTA**
>
> Cuando se utilizan funciones en Python, un buen consejo es seguir el principio de que "menos es más". Las funciones deberían ser pequeñas y hacer solo una cosa. Si encuentras que una función está creciendo demasiado o está empezando a hacer demasiadas cosas, probablemente sea el momento de dividirla en varias funciones más pequeñas.

Manejo de errores y excepciones

Un buen programa es aquel que es robusto y puede manejar errores y situaciones inesperadas. El manejo de errores y excepciones en Python permite a los programas manejar errores y continuar ejecutándose, incluso si algo inesperado sucede. Los programadores pueden definir lo que debería suceder si ocurre una excepción a un tipo de error particular.

Ejemplo: Imagina que tienes una función que divide dos números. En lugar de dejar que el programa falle si intentas dividir por cero, puedes capturar y manejar esa situación de la siguiente manera:

```
def dividir(numerador, denominador):
    try:
        return numerador / denominador
    except ZeroDivisionError:
        print("Error: No se puede dividir por cero.")
        return None
```

En este ejemplo, si intentas dividir por cero, el programa captura la excepción `ZeroDivisionError`, imprime un mensaje de error y devuelve `None`. De esta manera, el programa no se detiene y puede continuar ejecutándose, a pesar de que se intentó realizar una operación inválida. Esto hace que tu programa sea más robusto y amigable para el usuario.

Evitar código redundante

Este es un principio de desarrollo de software fundamental que sugiere que cualquier funcionalidad debería ser implementada en un solo lugar. Si encuentras que estás escribiendo el mismo código más de una vez, puede ser un signo de que deberías encapsular esa funcionalidad en una función o clase y reutilizarla.

Uso de las convenciones de estilo de Python (PEP 8)

La guía de estilo *Python Enhancement Proposal 8*, comúnmente conocida como PEP 8, es un conjunto de recomendaciones sobre cómo formatear el código Python. Seguir estas convenciones ayuda a mantener la consistencia y mejora la legibilidad del código. Aunque algunas de estas reglas pueden parecer arbitrarias, adherirse a ellas puede hacer que sea más fácil para otros (y para ti mismo) leer y entender tu código.

Puedes acceder a la guía en castellano en:
e.digitall.org.es/pep8

Prueba siempre que tu código funciona correctamente

Las pruebas unitarias son un componente esencial de la programación saludable. Permiten al programador verificar que las piezas individuales de código (o "unidades") estén funcionando correctamente bajo diferentes circunstancias y entradas. En Python, el módulo **unittest** es una herramienta poderosa para realizar estas pruebas.

Escribir pruebas unitarias para tu código puede ayudarte a identificar y corregir errores más rápido, mejorar la calidad del código, y también puede facilitar la modificación y extensión del código en el futuro.

Ejemplo: Supón que tienes la siguiente función que calcula el factorial de un número:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

Podrías escribir la siguiente prueba unitaria para asegurarte de que la función funciona correctamente:

```
import unittest
class TestFactorial(unittest.TestCase):
    def test_factorial(self):
        self.assertEqual(factorial(0), 1)
        self.assertEqual(factorial(1), 1)
        self.assertEqual(factorial(2), 2)
        self.assertEqual(factorial(3), 6)
        self.assertEqual(factorial(4), 24)
        self.assertEqual(factorial(5), 120)
if __name__ == '__main__':
    unittest.main()
```

La clase `TestFactorial` contiene un método llamado `test_factorial` que realiza varios tests sobre la función factorial. Se comprueba que el resultado de la función sea correcto para varios valores de entrada. Si alguno de estos tests falla, la función `unittest.main()` nos informará sobre el fallo y podríamos entonces corregir el problema en la función **factorial.**

Uso de patrones de diseño

Los patrones de diseño son soluciones probadas a problemas comunes en el diseño de software. Ofrecen un marco para enfrentar situaciones comunes, permitiendo una mejor organización del código y facilitando su mantenimiento y expansión en el futuro. Utilizar patrones de diseño puede ayudarte a escribir código más limpio, modular y eficiente.

Saber más

Para saber más sobre Patrones de diseño, se recomienda la lectura de Gamma, E. (2002). Patrones de diseño. España: Pearson Educación.

Documenta el código

Además de los comentarios y los docstrings en el código, también es importante proporcionar una documentación más amplia y completa. Esta puede incluir la documentación de la API para bibliotecas o módulos, los manuales del usuario para programas, y las guías de instalación o configuración para el software. La documentación es crucial para el mantenimiento y la escalabilidad de un proyecto, así como para la colaboración con otros desarrolladores.

Una buena documentación del código puede ser en forma de archivos README, wikis en el repositorio de código, o incluso páginas web dedicadas con la documentación completa del proyecto.

Saber más

Además de las buenas prácticas existen muchas más. Por ello, te recomendamos la lectura de los siguientes libros:

"Clean Code: A Handbook of Agile Software Craftsmanship" por *Robert C. Martin*: Aunque este libro no está escrito específicamente para Python, las lecciones y principios que enseña son aplicables a cualquier lenguaje de programación. El libro trata sobre cómo escribir código de alta calidad que sea fácil de leer, mantener y extender. Te ayudará a entender cómo los profesionales piensan sobre el diseño del software y por qué hacen las cosas de la manera que las hacen.

"Fluent Python: Clear, Concise, and Effective Programming" por *Luciano Ramalho*: Este libro es un excelente recurso para programadores intermedios y avanzados de Python. No solo cubre las características del lenguaje, sino que también ofrece perspectivas sobre las "formas pythonicas" de hacer las cosas. A través de este libro, puedes aprender a escribir código Python que es más idiomático, eficiente y claro.

Creación de
contenidos digitales

Nivel C2 3.4 Programación

Manejo de excepciones en Python

Manejo de excepciones en Python

Excepciones en Python

El lenguaje de programación Python distingue dos tipos de errores principales: i) los errores de sintaxis y ii) las excepciones. Los primeros ocurren debido a construcciones incorrectas a la hora de utilizar la sintaxis del lenguaje. El siguiente listado muestra un sencillo ejemplo, donde se puede apreciar cómo el intérprete de Python *señala* una posición cercana al potencial error (en este caso, la falta del carácter `:` después de '`while True`'). Las segundas, como ya se ha introducido previamente, responden a **eventos que tienen lugar durante la ejecución de un programa**, alterando de forma no deseada su flujo de ejecución *estándar*.

```
>>> while True print("Aprendiendo Python")
  File "<stdin>", line 1
    while True print("Aprendiendo Python")
               ^
SyntaxError: invalid syntax
```

Cuando se produce una excepción en Python, y no se controla, entonces el intérprete muestra el nombre de la excepción que se ha lanzado. Por ejemplo, el siguiente listado muestra la excepción `ZeroDivisionError`, definida por el propio lenguaje cuando se intenta dividir por 0.

```
>>> 7 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

Afortunadamente, es posible escribir programas en Python que manejen excepciones de forma adecuada. Para ello, es posible utilizar la **sentencia** `try`. Esta funciona de la siguiente forma:

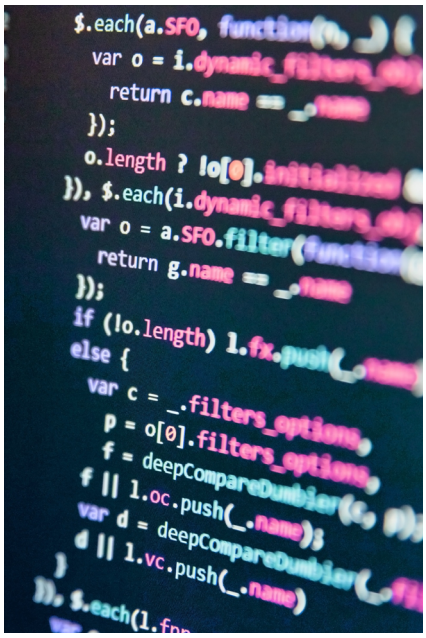
1 | En primer lugar, se ejecuta la cláusula try (las sentencias entre las palabras clave `try` y `except`).
2 | Si no se produce ninguna excepción, la cláusula `except` se salta y finaliza la ejecución de la sentencia `try`.
3 | Si se produce una excepción durante la ejecución de la cláusula `try`, se salta el resto de la cláusula. Entonces, si su tipo coincide con la excepción nombrada después de la palabra clave `except`, se ejecuta la cláusula `except` y la ejecución continúa después del bloque `try`/`except`.

4 | Si ocurre una excepción que no coincide con la excepción nombrada en la cláusula `except`, se pasa a las sentencias try externas. Si no se encuentra un manejador para la excepción, entonces se trata de una excepción no manejada y la ejecución se detiene con un mensaje como el del listado anterior.

Una sentencia `try` puede tener más de una cláusula `except`, con el objetivo de especificar manejadores para diferentes excepciones. Por otro lado, la sentencia `try` tiene una cláusula `else` opcional que, cuando está presente, debe seguir a todas las cláusulas `except`. Esta cláusula `else` es útil para incluir el código que debe ejecutarse si la cláusula `try` no lanza una excepción. El siguiente fragmento de código muestra un ejemplo en el que se trata de abrir un archivo, cuyo nombre se almacena en `mi_archivo`. Si el archivo no existe, entonces se lanzaría una excepción del tipo `OSError`, capturada en la cláusula `except`. Si el archivo existe, entonces, en este ejemplo, se ejecutaría el código de la cláusula `else`, donde se imprime el número de líneas que tiene el archivo.

```
try:
   f = open(mi_archivo, "r")
except OSError:
   print("No se pudo abrir", mi_archivo)
else:
   print(mi_archivo, "tiene", len(f.readlines()), "líneas.")
   f.close()
```

Es posible definir **excepciones de usuario**, es decir, excepciones cuya semántica está asociada al código que representa la solución a un problema concreto. Un ejemplo de excepción definida por el usuario, ya introducida anteriormente, podría contemplar el intento de almacenamiento de un número de teléfono que no tenga exactamente 9 dígitos. El siguiente listado muestra el código representativo de este ejemplo, en el que se puede apreciar la creación de la excepción `FormatoDeNumeroNoValidoException`, definida como una nueva clase en Python. Esta clase heredaría de la clase genérica `Exception`.



```
class FormatoDeNumeroNoValidoException(Exception):
    pass

numero = input("Introduzca un número de teléfono de 9 dígitos...")
if len(numero) != 9:
    raise FormatoDeNumeroNoValidoException
```

El ejemplo anterior también muestra cómo utilizar la sentencia raise para lanzar una excepción. En este caso, se podría suponer que la capa superior de código incluiría la lógica necesaria para capturar, y tratar, la excepción lanzada.

Para concluir esta sección, mencionaremos la cláusula finally. En esencia, si existe una cláusula finally, esta se ejecutará como última tarea antes de que finalice la sentencia try. La cláusula finally se ejecuta tanto si la sentencia try produce una excepción como si no. El siguiente ejemplo muestra un fragmento de código más completo.

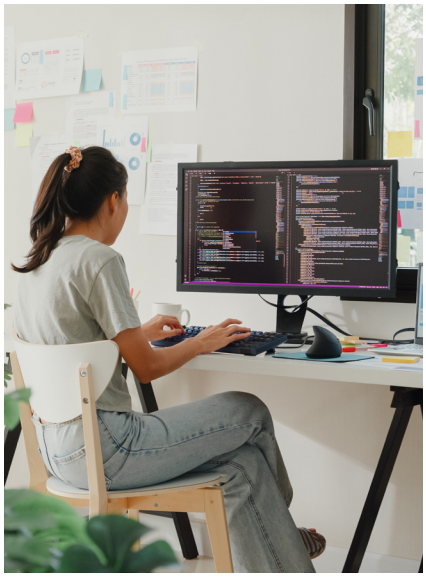
```
def dividir(a, b):
    try:
        resultado = a / b
    except ZeroDivisionError:
        print("División por cero.")
    else:
        print("El resultado es", resultado)
    finally:
        print("Ejecutando la cláusula finally")
```

Creación de
contenidos digitales

Nivel C2 3.4 Programación

Pruebas de código en Python

Pruebas de código en Python



Introducción

Las pruebas son una parte esencial en el proceso de desarrollo de aplicaciones. Gracias a ellas es posible verificar la correcta funcionalidad del código y garantizar su calidad. Un claro ejemplo de su importancia es el caso del Desarrollo Guiado por las Pruebas (TDD).

En este documento, se explorará cómo escribir pruebas unitarias, en concreto para aplicaciones desarrolladas en Python. Antes de continuar, es aconsejable que revises la documentación ***A3C34C1D06***.

PRUEBAS DE CÓDIGO. ASPECTOS FUNDAMENTALES

Documento introductorio a las pruebas de código y el enfoque TDD.

Documento referenciado: ***A3C34C1D06***

Tipos de pruebas. Pruebas unitarias

Las pruebas utilizadas durante el desarrollo de aplicaciones son de diferentes tipos, el uso de unas u otras depende de lo que se quiera verificar con ellas. Algunas de las más utilizadas son: las pruebas unitarias, pruebas de integración, pruebas funcionales y pruebas de regresión. Este documento se centra en las primeras, las pruebas unitarias.

Las pruebas unitarias son un tipo de pruebas focalizadas en verificar el correcto funcionamiento de unidades individuales de código. Es decir, asegurar que funciones, métodos o clases funcionen correctamente de manera aislada.

En Python, existen diferentes herramientas y frameworks que facilitan la creación y ejecución de pruebas unitarias. Algunos de las librerías más utilizados en Python son unittest, pytest y nose.

Pruebas en Python: unittest

Unittest es el framework de pruebas integrado en la biblioteca estándar de Python. Proporciona una serie de módulos y herramientas que ayudan a comprobar las funciones del código y facilitar el desarrollo de las aplicaciones.

Ejemplo práctico

En esta sección se verá un ejemplo práctico de pruebas unitarias con Python. En concreto, se ha desarrollado una aplicación que funciona a modo de calculadora. Se trata de una clase en la que se han implementado los métodos de suma, resta, multiplicación y división:

```
class Calculadora:
   def suma(self, a, b):
      return a + b
   def resta(self, a, b):
      return a - b
   def multiplicacion(self, a, b):
      return a * b
   def division (self, a, b):
      if b == 0:
         raise ValueError("Error: no se puede dividir entre cero")
      return a / b
```

Antes de implementar las pruebas, hay que analizar los diferentes escenarios a probar. Por ejemplo, en el caso de la calculadora será necesario comprobar la suma de dos números positivos, la resta de dos números negativos, la división entre cero, etc.

Una vez estudiados los casos a implementar, se procede a su codificación. En este caso, con unnitest, se pueden seguir los siguientes pasos:

- Crear una clase de pruebas llamada `TestCalculadora` que hereda de `unittest.TestCase`.
- Importar el módulo a probar (Calculadora).
- Crear el método `setUp` para inicializar una instancia de la calculadora antes de cada prueba.
- Definir métodos de prueba que cubran distintas posibilidades de ejecución. En cada método se emplearán aserciones como "`assertEquals`", "`assertNoEquals`", "`assertTrue`" o "`assertFalse`" para comprobar la funcionalidad.
- El nombre de los métodos debe comenzar con "`test_`".

Este código muestra un ejemplo de pruebas unitarias para cubrir la clase Calculadora.

```
import unittest
from Calculadora import Calculadora
class TestCalculadora(unittest.TestCase):
    def setUp(self):
        self.calculadora = Calculadora()

    def test_suma(self):
        result = self.calculadora.suma(-1, 3)
        self.assertEqual(result, 2)

    def test_subtraction(self):
        result = self.calculadora.resta(10, 4)
        self.assertFalse(result!= 6)

    def test_multiplication(self):
        result = self.calculadora.multiplicacion(0, 5)
        self.assertTrue(result == 0)

    def test_division(self):
        result = self.calculadora.division(80, 10)
        self.assertNotEqual(result, 80)

if __name__ == '__main__':
    unittest.main()
```

En este caso se ha implementado una prueba unitaria por cada método de la clase Calculadora. Sin embargo, lo correcto sería cubrir los diferentes escenarios de ejecución que puede tener cada uno de ellos.

En resumen, en este documento se ha abordado la importancia de las pruebas en el desarrollo de aplicaciones. En particular, se ha visto un ejemplo práctico de creación de pruebas unitarias con el framework de la librería estándar de Python, unittest. Es importante tener en cuenta que el mundo de las pruebas es amplio y complejo, y las pruebas unitarias son solo una parte del panorama completo de las pruebas de software.

NOTA

Para agilizar la creación y comprobación de pruebas unitarias, es recomendable utilizar algunos de los IDEs que dan soporte a Python. Puesto que en temas anteriores se vio el entorno de Visual Studio Code, es interesante ampliar los conocimientos utilizando las librerías de pruebas de Python que pueden ser utilizadas desde el IDE.

Saber más

A continuación, se listan algunos dos de los principales frameworks de pruebas para aplicaciones desarrolladas en Python.

Framework unittest: *e.digitall.org.es/unittest*
PyTest: *e.digitall.org.es/pytest*
Nose2: *e.digitall.org.es/nose2*

DigitAll

Formación en
Competencias
Digitales

https://www.digitall.org.es/

DigitAll

Coordinación General

Universidad de Castilla-La Mancha
Carlos González Morcillo
Francisco Parreño Torres

Coordinadores de área

Área 1. Búsqueda y gestión de información y datos

Universidad de Zaragoza
Francisco Javier Fabra Caro

Área 2. Comunicación y colaboración

Universidad de Sevilla
Francisco Javier Fabra Caro
Francisco de Asís Gómez Rodríguez
José Mariano González Romano
Juan Ramón Lacalle Remigio
Julio Cabero Almenara
María Ángeles Borrueco Rosa

Área 3. Creación de contenidos digitales

Universidad de Castilla-La Mancha
David Vallejo Fernández
Javier Alonso Albusac Jiménez
José Jesús Castro Sánchez

Área 4. Seguridad

Universidade da Coruña
Ana M. Peña Cabanas
José Antonio García Naya
Manuel García Torre

Área 5. Resolución de problemas

UNED
Jesús González Boticario

Coordinadores de nivel

Nivel A1

Universidad de Zaragoza
Ana Lucía Esteban Sánchez
Francisco Javier Fabra Caro

Nivel A2

Universidad de Córdoba
Juan Antonio Romero del Castillo
Sebastián Rubio García

Nivel B1

Universidad de Sevilla
Francisco de Asís Gómez Rodríguez
José Mariano González Romano
Juan Ramón Lacalle Remigio
Montserrat Argandoña Bertran

Nivel B2

Universidad de Castilla-La Mancha
María del Carmen Carrión Espinosa
Rafael Casado González
Víctor Manuel Ruiz Penichet

Nivel C1

UNED
Antonio Galisteo del Valle

Nivel C2

UNED
Antonio Galisteo del Valle

Maquetación

Universidad de Salamanca
Fernando De la Prieta Pintado
Pilar Vega Pérez
Sara Alejandra Labrador Martín

Creadores de contenido

Área 1. Búsqueda y gestión de información y datos

1.1 Navegar, buscar y filtrar datos, información y contenidos digitales

Universidad de Huelva
Ana Duarte Hueros (coord.)
Arantxa Vizcaíno Verdú
Carmen González Castillo
Dieter R. Fuentes Cancell
Elisabetta Brandi
José Antonio Alfonso Sánchez
José Ignacio Aguaded
Mónica Bonilla del Río
Odiel Estrada Molina
Tomás de J. Mateo Sanguino (coord.)

1.2 Evaluar datos, información y contenidos digitales

Universidad de Zaragoza
Ana Belén Martínez Martínez
Ana María López Torres
Francisco Javier Fabra Caro
José Antonio Simón Lázaro
Laura Bordonaba Plou
María Sol Arqued Ribes
Raquel Trillo Lado

1.3 Gestión de datos, información y contenidos digitales

Universidad de Zaragoza
Ana Belén Martínez Martínez
Francisco Javier Fabra Caro
Gregorio de Miguel Casado
Sergio Ilarri Artigas

Área 2. Comunicación y colaboración

2.1 Interactuar a través de tecnología digitales

Iseazy

2.2 Compartir a través de tecnologías digitales

Universidad de Sevilla
Alién García Hernández
Daniel Agüera García
Jonatan Castaño Muñoz
José Candón Mena
José Luis Guisado Lizar

2.3 Participación ciudadana a través de las tecnologías digitales

Universidad de Sevilla
Ana Mancera Rueda
Félix Biscarri Triviño
Francisco de Asís Gómez Rodríguez
Jorge Ruiz Morales
José Manuel Sánchez García
Juan Pablo Mora Gutiérrez
Manuel Ortigueira Sánchez
Raúl Gómez Bizcocho

2.4 Colaboración a través de las tecnologías digitales

Universidad de Sevilla
Belén Vega Márquez
David Vila Viñas
Francisco de Asís Gómez Rodríguez
Julio Barroso Osuna
María Puig Gutiérrez
Miguel Ángel Olivero González
Óscar Manuel Gallego Pérez
Paula Marcelo Martínez

2.5 Comportamiento en la red

Universidad de Sevilla
Ana Mancera Rueda
Eva Mateos Núñez
Juan Pablo Mora Gutiérrez
Óscar Manuel Gallego Pérez

2.6 Gestión de la identidad digital

Iseazy

Área 3. Creación de contenidos digitales

3.1 Desarrollo de contenidos

Universidad de Castilla-La Mancha
Carlos Alberto Castillo Sarmiento
Diego Cordero Contreras
Inmaculada Ballesteros Yáñez
José Ramón Rodríguez Rodríguez
Rubén Grande Muñoz

3.2 Integración y reelaboración de contenido digital

Universidad de Castilla-La Mancha
José Ángel Martín Baos
Julio Alberto López Gómez
Ricardo García Ródenas

3.3 Derechos de autor (copyright) y licencias de propiedad intelectual

Universidad de Castilla-La Mancha
Gabriela Raquel Gallicchio Platino
Gerardo Alain Marquet García

3.4 Programación

Universidad de Castilla-La Mancha
Carmen Lacave Rodero
David Vallejo Fernández
Javier Alonso Albusac Jiménez
Jesús Serrano Guerrero
Santiago Sánchez Sobrino
Vanesa Herrera Tirado

Área 4. Seguridad

4.1 Protección de dispositivos

Universidade da Coruña
Antonio Daniel López Rivas
José Manuel Vázquez Naya
Martiño Rivera Dourado
Rubén Pérez Jove

4.2 Protección de datos personales y privacidad

Universidad de Córdoba
Aida Gema de Haro García
Ezequiel Herruzo Gómez
Francisco José Madrid Cuevas
José Manuel Palomares Muñoz
Juan Antonio Romero del Castillo
Manuel Izquierdo Carrasco

4.3 Protección de la salud y del bienestar

Universidade da Coruña
Javier Pereira Loureiro
Laura Nieto Riveiro
Laura Rodríguez Gesto
Manuel Lagos Rodríguez
María Betania Groba González
María del Carmen Miranda Duro
Nereida María Canosa Domínguez
Patricia Concheiro Moscoso
Thais Pousada García

4.4 Protección medioambiental

Universidad de Córdoba
Alberto Membrillo del Pozo
Alicia Jurado López
Luis Sánchez Vázquez
María Victoria Gil Cerezo

Área 5. Resolución de problemas

5.1 Resolución de problemas técnicos

Iseazy

5.2 Identificación de necesidades y respuestas tecnológicas

Iseazy

5.3 Uso creativo de la tecnología digital

Iseazy

5.4 Identificar lagunas en las competencias digitales

Iseazy

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia