DigitAll

Formació en Competències Digitals

3

Creació de continguts digitals

DigitAll

Formació en competències digitals

Creació de contingut digital

Nivell C2

ÍNDEX

DigitAll

Creació de
continguts digitals

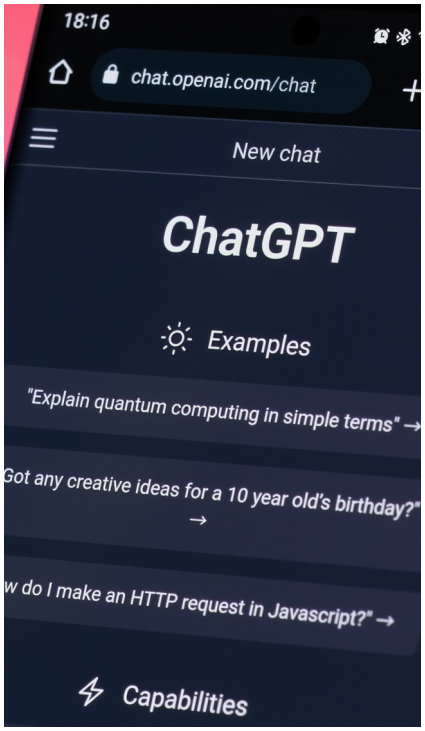
3.1
DESENVOLUPAMENT DE CONTINGUTS

Creació de
continguts digitals

Nivell C2 3.1 Desenvolupament de continguts

Generació automàtica de text mitjançant GPT

Generació automàtica de text mitjançant GPT



Introducció

La generació de text mitjançant GPT és una eina molt valuosa en la creació de continguts digitals amb multitud de beneficis segons el camp en el qual s'apliqui. Així, GPT és capaç de produir text coherent i convincent en una àmplia varietat de temes, la qual cosa ens permetrà estalviar temps i esforç, ja que no hem de redactar cada paraula des de zero. A més, aquesta tecnologia permet als usuaris explorar idees noves i creatives, ja que pot suggerir diferents enfocaments i perspectives. Finalment, pot ser emprada per a moltes de les qüestions que hem treballat en aquest i nivells anteriors, com per exemple la creació d'entrades en el nostre lloc web.

GPT facilita la personalització del contingut, adaptant-se a les necessitats i preferències específiques de cada projecte. En resum, la generació de text mitjançant GPT és una eina poderosa que optimitza la creació de continguts digitals en estalviar temps, millorar la qualitat i fomentar la creativitat.

És habitual que ens trobem descripcions de GPT com "una intel·ligència artificial que està entrenada per a mantenir converses" i, encara que això és cert, aquesta percepció pot ser un impediment a l'hora d'obtenir informació a través d'aquest sistema. Si ho pensem detingudament, resulta una mica obvi: la resposta a una mateixa pregunta serà diferent depenent del context: qui ens la faci, en quin moment i lloc, etc.

Per tant, serà molt important tenir en compte la informació que proporcionarem a ChatGPT, o altres models de generació de text, i que definirà el context en el qual volem que el sistema ens proporcioni respostes. En aquest text plantejam quina informació hem de proporcionar a ChatGPT per obtenir un text que s'ajusti a les nostres necessitats específiques.

Quina informació hem de proporcionar al sistema?

Descriurem una sèrie d'apartats generals que hem de definir i que s'adaptaran a la majoria de les peticions de generació de text que fem.

Propòsit i tema

Començam pel més obvi: hem d'especificar el tema principal sobre el qual volem que tracti el text i també el propòsit amb el qual el necessitem: informatiu, persuasiu, descriptiu, divulgatiu, etc.

Estructura

Hem de subministrar els detalls i dades necessàries que han d'incloure's en el text. Per exemple, podem demanar que ens inclogui dades estadístiques, casos pràctics, successos històrics, etc.

Informació rellevant

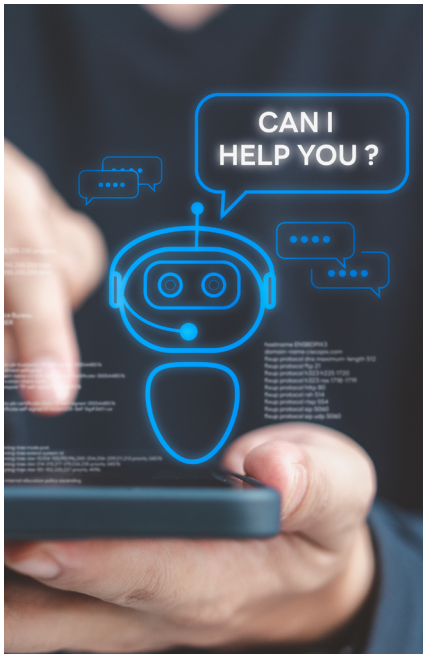
Hem de subministrar els detalls i dades necessàries que han d'incloure's en el text. Per exemple, podem demanar que ens inclogui dades estadístiques, casos pràctics, successos històrics, etc.

Estil i to

A la nostra publicació hem d'indicar l'estil i to que volem que tingui el nostre text. Per exemple, formal, informal, tècnic, persuasiu, neutre, etc. A més, podem indicar si tenim alguna preferència quant a la longitud de les oracions o l'ús de vocabulari. Per exemple, frases curtes o vocabulari tècnic.

Audiència objectiva

A qui va dirigit el nostre text? Això ajudarà a adaptar el llenguatge i l'enfocament perquè sigui apropiat i efectiu per al públic al qual es dirigeix. Per exemple, podem demanar que ens escrigui un conte per a infants o per a adults.



Requisits addicionals

Si hi ha requisits específics, com la inclusió de referències (per exemple, llibres o webs de confiança), l'ús de llenguatge inclusiu o l'extensió del text, podem també esmentar-ho en la nostra petició.

Revisión i ajustaments

És un pas fonamental que moltes vegades s'oblida en fer servir aquest tipus d'eines i és que el procés no acaba quan el sistema ens ha generat un text sobre la base del nostre prompt. Per tant, aquest pas el farem després de rebre el nostre text: ho revisam i, si hi ha parts que no compleixen amb les nostres expectatives, generarem una nova petició amb les modificacions oportunes perquè el sistema s'ajusti millor a les nostres preferències. Per exemple, podem demanar que ens reescrigui un paràgraf determinat incloent exemples per fer-lo més comprensible.

Conclusió

Per tant, com que es tracta d'un model de llenguatge, ChatGPT es basa en patrons i dades existents per a generar respostes coherents. No obstant això, sense informació addicional, el model pot no comprendre per complet el context o les especificacions exactes del text que desitgem. En proporcionar una descripció detallada, podem maximitzar el valor i la utilitat de la generació automàtica de text i obtenir un text adaptat a les nostres necessitats.

Saber-ne més

En l'article "Sparks of Artificial General Intelligence: Early experiments with GPT-4" (arXiv:2303.12712) podem trobar una recerca exhaustiva sobre com es va entrenar GPT-4 i com es va comparar amb el seu predecessor, GPT-3. Encara que es discuteixen possibles aplicacions i limitacions de GPT-4 el més interessant en relació amb aquest text és la multitud de diferents prompts que s'assagen i els resultats que retorna el sistema.

Creació de
continguts digitals

Nivell C2 3.1 Desenvolupament de continguts

Ús avançat de rutes per a selecció, retallada i definició d'àrees

Ús avançat de rutes per a selecció, retallada i definició d'àrees

Introducció

Entre les nombroses eines avançades d'edició i manipulació d'imatges disponibles, les rutes exerceixen un paper fonamental en la selecció, retallada i definició precisa d'àrees en imatges. En aquest text, ens centrarem en l'ús avançat de rutes per a aconseguir resultats precisos i detallats en aquestes tasques completant els coneixements adquirits en el nivell anterior. Continuarem treballant amb el programa GIMP com a exemple.

Controla les teves rutes a través del panell de rutes

Ja hem vist que, a diferència de les eines de selecció bàsiques, les rutes permeten delinear contorns complexos i corbes suaus amb major precisió. Així, podem traçar una ruta al voltant d'un objecte en una imatge i després convertir aquesta ruta en una selecció activa i aplicar diferents efectes o modificacions específiques a aquesta àrea en particular.

Podrem manipular les nostres rutes a través del panell de rutes, una eina que permet gestionar i manipular les rutes creades. Aquest panell proporciona una vista detallada de totes les rutes presents en un projecte i ofereix opcions per a editar, organitzar, mostrar (o no) i, en definitiva, utilitzar aquestes rutes de manera eficient. Com veiem en la Figura 1, apareix al costat del panell de capes i funciona d'una manera molt similar.

Exemple pràctic

Les possibilitats de les rutes en el disseny són enormes, depenent de quin sigui l'objectiu final que ens hàgim marcat per a la nostra composició, podrem emprar les rutes d'un mode o un altre. Ara que sabem els conceptes més bàsics, descriurem pas per pas com fer un disseny emprant rutes. El resultat final d'aplicar aquests passos a GIMP es mostra a la Figura 2.

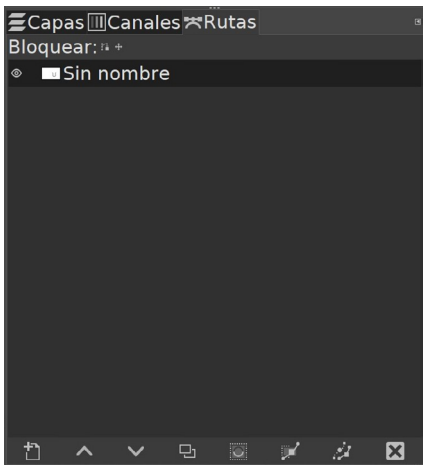


Figura 1 (A). Vista general d'una ruta en el panell de rutes.

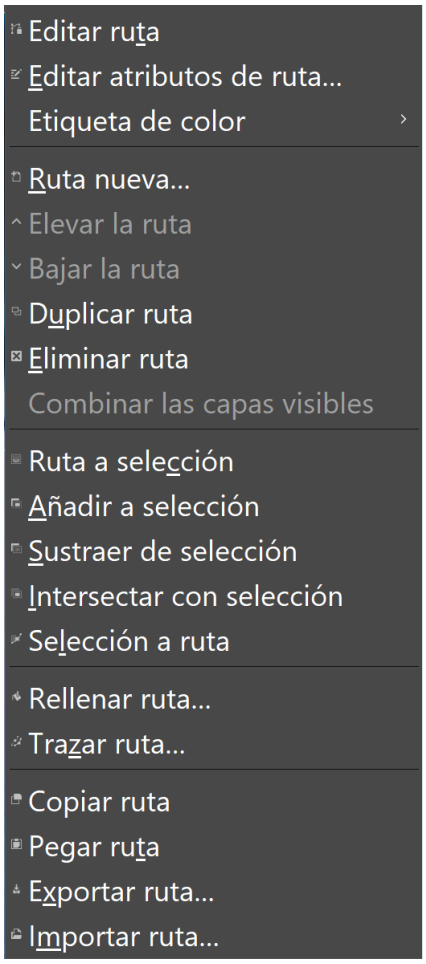


Figura 1 (B). Desplegable que s'obre en pitjar el segon botó del ratolí sobre una ruta.

Descarregam l'imatge original (1920×1293) des de: ***e.digitall.org.es/practica-rutas***

- Obrim la imatge a GIMP.
- Seleccionam l'"eina de rutes" i dibuixam una ruta al voltant de la branca i l'ocell.
- Editam les vegades que siguin necessàries fins que aconseguim un bon resultat.
- Cream una selecció clicant a "Crear selecció a partir d'una ruta" en el menú de l'eina de rutes.
- Copiam la selecció prement CONTROL + C.
- Cream una nova imatge (Arxiu > Nou) on crearem la nostra nova composició.
- Aferram la nostra imatge prement CONTROL + V. Apareixerà en el menú de capes com a "Selecció flotant" així que feim clic en el segon botó i seleccionam del menú "A una nova capa".
- Tornam a la imatge original i en el panell de rutes feim clic a "Copiar ruta".
- Anam al panell de rutes de la nostra nova imatge i feim clic a "Aferrar ruta".
- Ja tenim en la nostra nova composició la part de la nostra imatge que ens interessava i la ruta que cream per retallar-la.
- Una opció és generar una silueta sobre la nostra imatge usant la nostra ruta com a referència, per a això cream una nova capa transparent (entre la nostra capa i el fons).
- Anam al panell de rutes, seleccionam la nostra ruta i clicam "Pintar al llarg de la ruta" (a baix a la dreta).
- En el panell d'opcions que ens apareix seleccionam Traçar línia > Color sòlid (anti-àlies ha d'estar seleccionat) i triam les opcions del traç que aplicarem al nostre gust.
- En prémer a "Traçar" veim que ens apareix un traç de les característiques indicades amb la forma de la nostra ruta en la nostra nova capa.
- Afegirem un segon efecte en el qual crearem una silueta farcida de color amb la forma de la nostra ruta. Per a això duplicam la nostra ruta i cream una nova capa (entre les capes que ja teníem i el fons).
- Ara anam a l'eina de rutes i li direm que volem "Moure" la nostra ruta i la movem lleugerament a la direcció que volem.
- Ara indicam que volem "Emplenar ruta" amb un "Color sòlid" i, de nou, indicam les característiques del color de

IMATGE ORIGINAL EXEMPLE PRÀCTIC
e.digitall.org.es/practica-rutas

Ctrl + C = **COPIAR**

Ctrl + V = **PEGAR**

farciment (anti-àlies ha d'estar seleccionat). Veiem que ens ha emplenat la nostra ruta amb el color que li hem indicat.

- Finalment, podem introduir efectes que ja coneixem per a millorar la composició:
- Podem per exemple, en la capa del fons, afegir color amb l'"Eina de degradat".
- Modificar els colors de la nostra imatge retallada des del panell "Colors".
- O canviar el mode de la capa de farciment que hem creat.

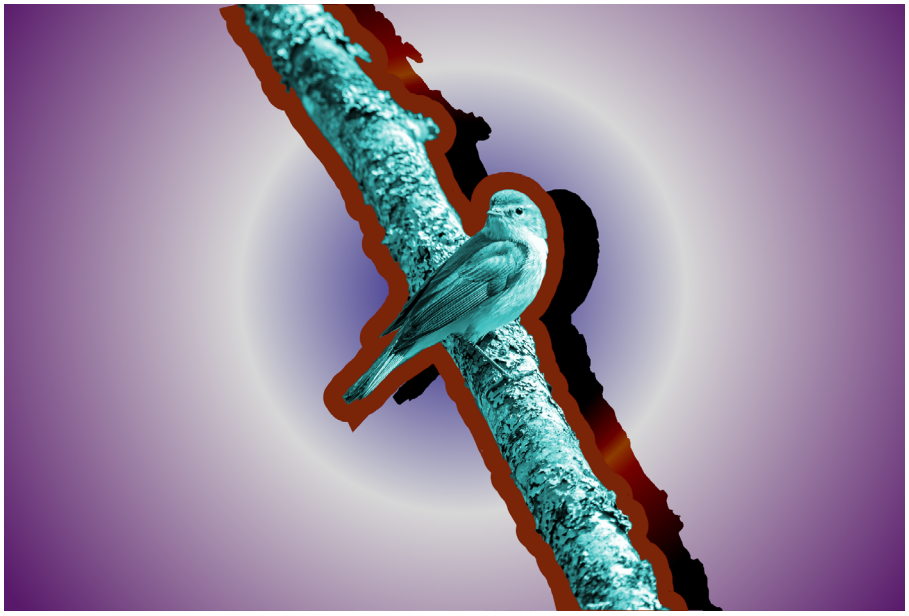


Figura 2. Resultat d'aplicar pas per pas les instruccions descrites en l'apartat "Exemple pràctic".

Saber-ne més

Les eines de dibuix de rutes ofereixen un nivell avançat de precisió i flexibilitat en la selecció, retallada i definició d'àrees en imatges. En tots els programes de disseny ens trobarem amb eines similars, en el programa Photoshop, en comptes de rutes es denominen traçats. Podem accedir a informació addicional sobre com s'organitzen aquestes eines en el popular programa de disseny a través del següent enllaç.

e.digitall.org.es/rutas-photoshop

Creació de
continguts digitals

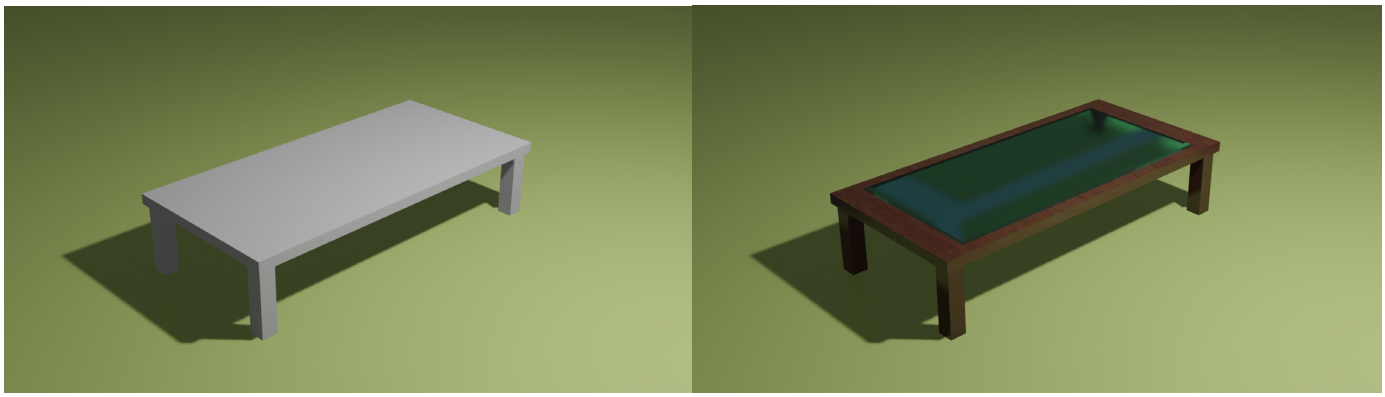
Nivell C2 3.1 Desenvolupament de continguts

Aplicació de materials i texturat

Aplicació de materials i texturat

Introducció

Definir la geometria d'un objecte en el món de la informàtica gràfica és només el primer pas en la creació de models 3D realistes i visualment atractius. Una vegada que el model ha estat dissenyat i modelatge, és essencial assignar materials adequats i, en alguns casos, aplicar textures per a aconseguir una aparença realista i coherent. L'assignació de materials determina com interactua l'objecte amb la llum i com es reflecteixen, refracten o absorbeixen els raigs lluminosos en la seva superfície. A més, el texturat UV Mapping és una tècnica essencial per a aplicar imatges o patrons en la superfície d'un objecte 3D amb precisió i realisme. En aquest document, explorarem la importància d'assignar materials i textures en la creació de gràfics 3D.



A l'esquerra tenim una taula creada sense assignar una textura predeterminada, i a la dreta tenim una taula a la qual se li han aplicat diferents textures. Que es noti la diferència en com actua la llum en cadascuna d'elles, aconseguint una aparença molt més realista.

Assignació de materials i comportament de la llum

Assignació de materials

L'assignació de materials en un objecte 3D és un procés vital per a simular com interactua amb la llum. Cada material té propietats específiques que determinen com la llum incideix i es reflecteix en la seva superfície. Alguns materials poden ser opacs i reflectir la llum en una sola direcció, mentre que uns altres poden ser transparents o refractar la llum quan passa a través d'ells. La configuració dels atributs del material, com la

rugositat, la reflexió i l'índex de refracció, influirà en l'aparença final de l'objecte i el seu realisme en l'escena.

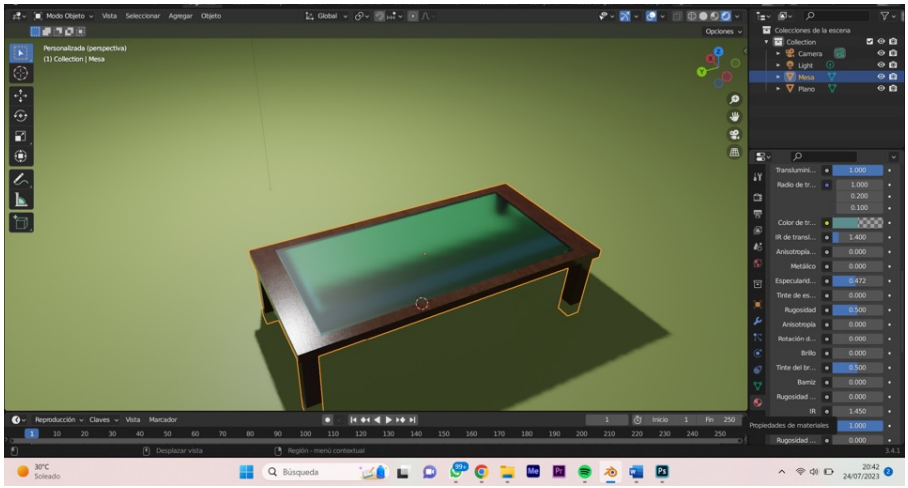
En molts programes de modelatge i renderització 3D, els materials es defineixen utilitzant el model d'il·luminació de Phong o el model d'ombreig de Lambert. Aquests models permeten simular la manera en què la llum interactua amb les superfícies de l'objecte, i com es crea l'efecte d'ombreig i lluentor en funció de la posició de la font de llum i el punt de vista de la càmera.

Comportament de la llum, propietats

Els materials defineixen com interactua l'objecte amb la llum i com es reflecteixen, refracten o absorbeixen els raigs lluminosos en la seva superfície. Cada material té propietats específiques que determinen com es comportarà la llum en relació amb l'objecte. Aquestes propietats es defineixen sempre que s'assignin un material. Algunes de les propietats més importants són:

- **Color:** el color és l'atribut més bàsic d'un material i defineix com es veurà l'objecte sota la llum blanca. Pots triar un color sòlid o una textura de color per obtenir un resultat específic.
- **Lluentor (especularitat):** aquest atribut controla la quantitat i la grandària dels reflexos en la superfície de l'objecte. Un valor alt de lluentor crearà reflexos intensos, mentre que un valor baix donarà un aspecte més mat.
- **Rugositat (*roughness*):** la rugositat defineix com de suau o aspra és la superfície del material. Una superfície molt rugosa dispersarà la llum i donarà un aspecte difús, mentre que una superfície llisa reflectirà la llum en una direcció més precisa.
- **Transparència:** aquest atribut controla que transparent és el material. Pots fer que l'objecte sigui completament transparent o semitransparent per simular vidre o altres materials transparents.
- **Índex de refracció:** l'índex de refracció determina com la llum es doblega en passar a través del material transparent. Això és especialment important per simular materials com el vidre o l'aigua.

- **Textures:** a més dels atributs esmentats anteriorment, pots aplicar textures per a agregar detalls més complexos i realistes a la superfície de l'objecte. Les textures poden incloure patrons, imatges o qualsevol mena de disseny que desitgis incorporar en el model.



A Blender podem definir aquestes propietats seleccionant un objecte i prement en la finestra de materials situada a baix a la dreta de la interfície. En aquesta captura podem observar les propietats emprades per al material creat per simular el vidre.

És important tenir en compte que l'assignació de materials no només afecta l'aparença de l'objecte en un renderitzat estàtic, sinó que també té un impacte en com es veurà l'objecte en diferents condicions d'il·luminació i en moviment en una animació.

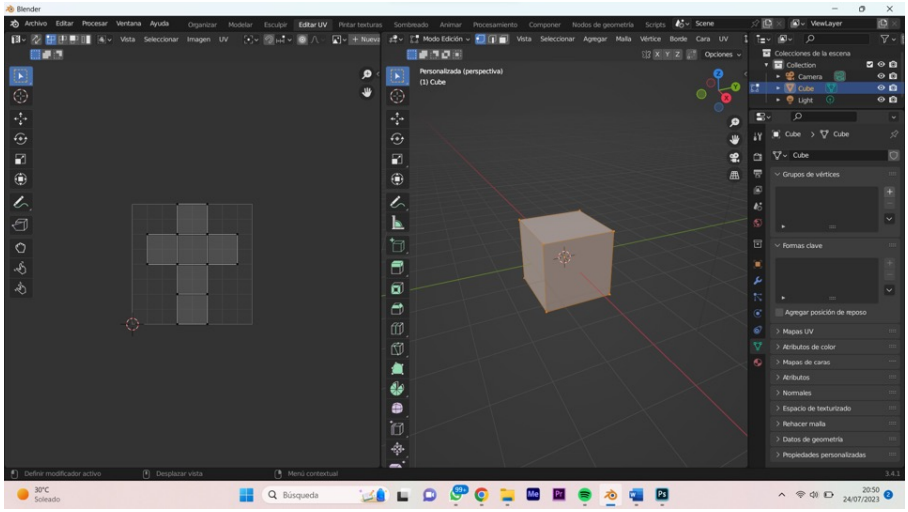
A molts programes de modelatge i renderitzat 3D, com Blender, Maia, 3ds Max i altres, trobaràs una interfície intuïtiva per assignar materials als teus models 3D. Aquests programes ofereixen biblioteques de materials predefinits i també et permeten crear i personalitzar els teus propis materials, ajustant els atributs esmentats anteriorment per obtenir el resultat desitjat.

Tècnica de texturat UV Mapping

Assignar una textura a un objecte és senzill. Podem afegir una textura directament a un objecte (entengui's com a textura una imatge que representa "la pell" de l'objecte) i serà el mateix programa de disseny 3D qui la distribueixi automàticament i

de manera uniforme al llarg de tota la superfície de l'objecte. El problema és que a vegades aquesta distribució automàtica no permet un ajust adequat de la textura. Existeixen altres tècniques més avançades que permeten un ajustament més manual i precís com la tècnica de texturat d'UV Mapping.

El texturat UV Mapping és una tècnica que permet aplicar textures o imatges en la superfície d'un model 3D. Per fer-ho, s'han de "mapejar" coordenades 2D en la superfície de l'objecte 3D, conegudes com a coordenades UV. Aquesta tècnica és especialment útil quan es desitja detallar l'aparença de l'objecte amb major realisme, com agregar patrons, textures o imatges complexes.



Des de Blender tenim una vista pròpia dedicada exclusivament a l'UV mapping que ja ens "Desembolica" l'objecte amb el qual treballam.

El procés de texturat UV Mapping comença desplegant les cares del model 3D en un pla 2D, de manera que es puguin aplicar les textures de manera precisa. Això pot realitzar-se utilitzant eines específiques en programes de modelatge 3D, o fins i tot de manera manual si es necessita un control més precís.

Una vegada que les coordenades UV s'han assignat correctament, es pot aplicar la textura desitjada en un programa d'edició d'imatges. Després, la textura s'assigna al material de l'objecte 3D en el programari de modelatge o renderitzat, on s'ajusten paràmetres com la grandària, la intensitat i la repetició de la textura per obtenir el resultat desitjat.

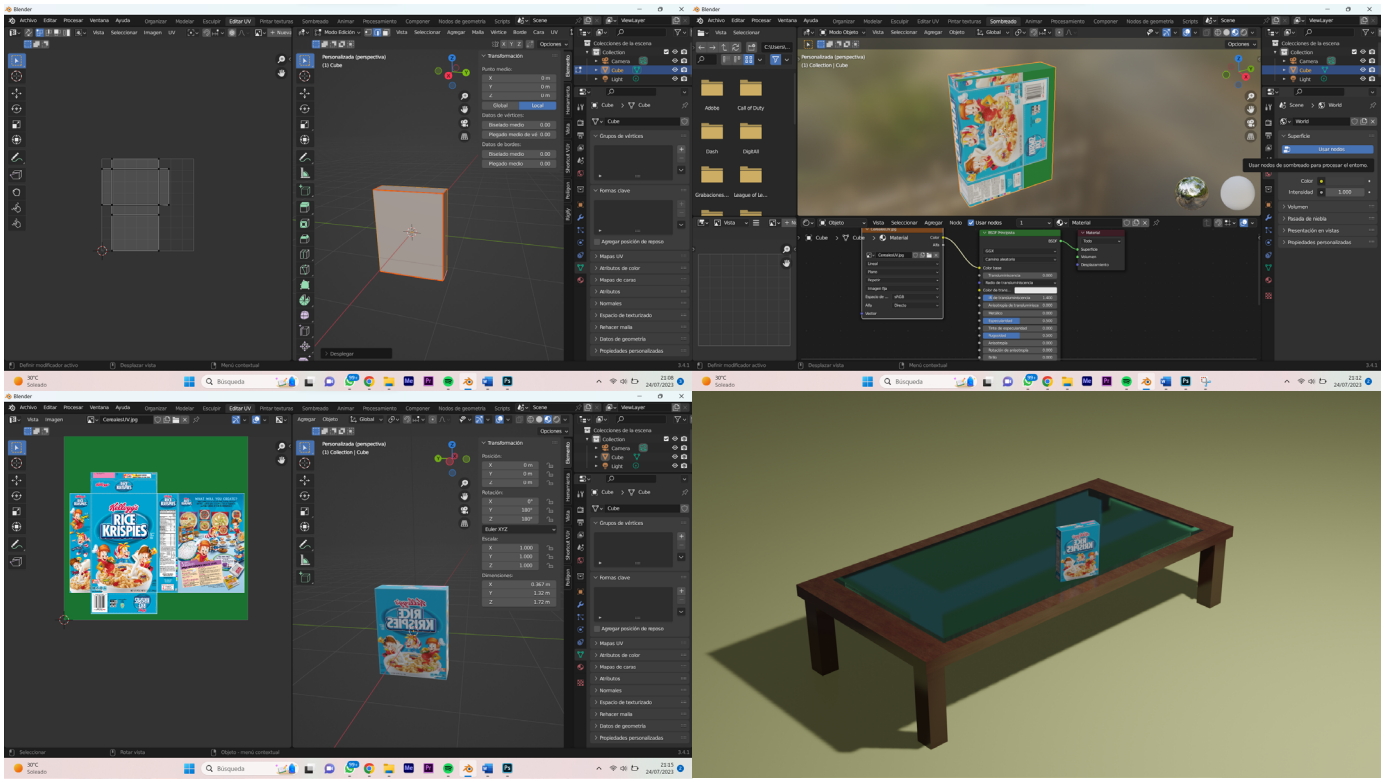
A Blender això es traduiria únicament com desembolicar l'objecte que vulguem des de l'apartat UV editting, aplicar-li una textura d'imatge (seleccionant el mapa de textura que desitgem en png o jpeg com a imatge) i tornar a UV editting per ajustar aquesta imatge al nostre objecte. És important recalcar que a millor resolució tingui la nostra imatge millor es veurà el nostre render.

Com això és més fàcil dir-ho que fer-ho, intentarem fer-nos aquest procés el més fàcil possible afegint algunes pautes a seguir i els menús necessaris per això. Per començar i desembolicar l'objecte, únicament ens hem de situar en la vista "**UV editting**" on ja ens apareixerà l'objecte desembolicat. El següent seria anar a l'apartat "**Ombrejat**". En aquesta vista haurem d'adjuntar la imatge (Que ha de seguir el format típic de mapa UV, només caldria cercar a Google el que vulguem seguit de "**Texture map**"), per això arrossegarem la imatge de la carpeta on la tinguem directament a la part inferior de la vista d'ombreig, al costat del quadre anomenat "BSDF principista". Una vegada tenguem la imatge afegida en aquest "esquema", haurem d'unir el punt "Color" de la nostra imatge amb el punt "Color base" de "BSDF principista" amb el que ja tendríem assignada aquesta imatge a l'objecte que volem, ara només hem de reposicionar-la perquè es vegi correctament.

Per això, tornam a "**UV edditing**". En aquest cas ara veurem la mateixa figura desembolicada al costat de la imatge que hem importat. Perquè la imatge estigui d'acord amb el resultat que vulguem, hem de fer clic on tenim la figura desembolicada i prémer la tecla a. Amb això tindrem tota la figura desembolicada seleccionada, ara hem de moure aquesta (amb la tecla g) o girar-la (prement la tecla R i escrivint els graus que vulguem girar-la si volem major precisió) de manera que la figura desembolicada coincideixi exactament amb la imatge del mapa de textures que hem afegit.

Cal afegir que la imatge que emprem no ha de ser necessàriament un mapa de textures, es pot afegir la imatge que es desitgi i ajustar-la a plaer. No obstant això, els mapes de textures estan pensats específicament per donar-li forma a figures concretes i ens poden ser molt útils.

A	=	SELECCIONAR FIGURA DESEMBOLICADA
G	=	MOURE FIGURA
R	=	GIRAR FIGURA



En l'ordre de les agulles del rellotge, en la primera imatge tenim una galleda que s'ha escalat en forma de caixa desembolicat en l'UV editting. A la segona imatge se'ns mostra com se li ha aplicat a la galleda una textura amb la imatge del mapa de textures d'una caixa de cereals aleatòria. En la tercera imatge es mostra com s'ha ajustat el mapa de textures a la fora uv desembolicada de la galleda. En la quarta imatge es mostra finalment un render de com quedaria la caixa damunt de la taula prèviament realitzada.

Conclusió

En conclusió, l'assignació de materials i l'aplicació de textures són aspectes crítics en la creació de models 3D realistes i visualment atractius. L'assignació de materials permet definir com la llum interactua amb la superfície de l'objecte, influint en la seva aparença i comportament visual en l'escena. D'altra banda, el texturat UV Mapping és una tècnica valuosa per aplicar imatges o patrons en la superfície del model amb precisió i realisme.

Dominar aquestes tècniques i entendre com afecten l'aparença de l'objecte és fonamental per a aconseguir resultats d'alta qualitat en projectes d'animació, videojocs, disseny arquitectònic i altres àrees de la informàtica gràfica. La combinació adequada de materials i textures agrega un nivell de detall i realisme que pot marcar la diferència entre un model 3D ordinari i una creació visualment impactant. És per això que serà necessari dedicar-li moltes hores a l'enteniment i ús d'aquestes tècniques.

Creació de continguts digitals

Nivell C2 3.1 Desenvolupament de continguts

Renderitzat per a la generació d'imatges 2D i vídeo

Renderitzat per a la generació d'imatges 2D i vídeo

Introducció

En el món de l'animació i els gràfics 3D, el renderitzat és un procés crucial per convertir un model 3D en imatges o vídeos finals. Existeixen diverses opcions per a realitzar el renderitzat, cadascuna amb els seus propis motors de *rendering* i configuracions de sortida. A continuació, explorarem algunes de les opcions més populars per al renderitzat, així com les configuracions i opcions disponibles per a obtenir resultats d'alta qualitat.

Motors de *rendering*

En aquest cas no ens centrarem només en Blender, perquè ja vam veure en el seu vídeo d'introducció ("***Introducció general a la interfície d'una eina de disseny 3D***") quins motors de rendering emprava, farem un recorregut global de motors de *rendering*:

- **Cycles (Blender):** Cycles és el motor de renderitzat de Blender i és àmpliament utilitzat a causa de la seva flexibilitat i realisme. Ofereix un enfocament basat en el traçat de raigs que permet simular efectes de llum i ombra de manera precisa, brindant resultats fotorealistes. Cycles és ideal per produir imatges d'alta qualitat i animacions realistes.
- **Arnold (Autodesk Maya):** Arnold és un motor de renderitzat altament valorat i utilitzat en la indústria del cinema i l'animació. És conegut per la seva capacitat per a manejar complexes escenes i efectes visuals. Arnold proporciona un rendiment òptim en el càlcul d'il·luminació i ombreig, la qual cosa el converteix en una opció poderosa per projectes d'alta gamma.
- **RenderMan (Pixar):** RenderMan és un motor de renderitzat desenvolupat per Pixar i és àmpliament utilitzat en la creació de pel·lícules animades. És conegut per la seva capacitat per produir imatges amb detalls i complexitat excepcionals. RenderMan ofereix una àmplia gamma de configuracions i opcions avançades per a un control precís sobre el resultat final.

INTRODUCCIÓ GENERAL A LA INTERFÍCIE D'UNA EINA DE DISSENY 3D

e.digitall.org.es/A3C31C2V04

Configuració i opcions de sortida

- **Resolució:** la resolució determina el nombre de píxels en la imatge final. Una resolució més alta resultarà en imatges més detallades, però també augmentarà el temps de renderitzat.
- **Grandària del fotograma:** defineix la grandària de l'àrea de visualització en l'escena 3D. Pots renderitzar en format de pantalla panoràmica, quadrat o personalitzat.
- **Mostreig i *antialiasing*:** aquestes configuracions controlen la qualitat de les vores i eliminen els efectes d'escala (*aliasing*) en les vores dels objectes, proporcionant imatges més suaus.
- **Profunditat de color:** defineix la quantitat d'informació de color per píxel. Majors profunditats de color (com 16 o 32 bits) permeten un rang més ampli de colors i eviten la pèrdua de detalls en les ombres i llums.
- **Format de sortida:** pots seleccionar el format d'arxiu de sortida, com JPEG, PNG, TIFF o EXR, entre altres. El format triat dependrà de les teves necessitats i de si desitges conservar informació de color i detalls de manera òptima.
- **Compressió:** alguns formats d'arxiu permeten la compressió per a reduir la grandària de l'arxiu resultant. No obstant això, has de tenir en compte que la compressió pot afectar la qualitat de la imatge.

Generació de vídeo

Per generar una animació en vídeo en lloc d'un únic fotograma, es requereix combinar múltiples imatges renderitzades en seqüència. Això s'aconsegueix mitjançant el procés de renderitzat de fotogrames clau o *keyframes*. A continuació es descriuen els passos generals per generar un vídeo:

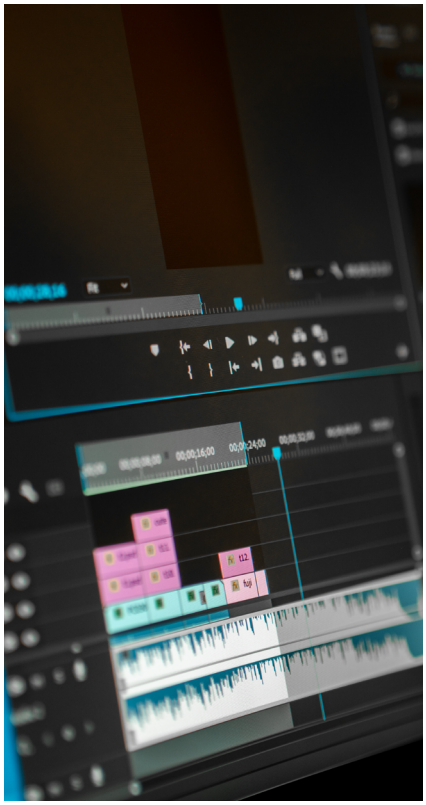
- **Animació i *keyframes*:** defineix l'animació en la teva escena 3D, establint posicions clau (keyframes) per a objectes, càmeres i il·luminació. Els keyframes indiquen com evoluciona l'escena al llarg del temps.
- **Seqüenciador de vídeo o editor de vídeo:** molts programes de modelatge i renderitzat 3D, com Blender, proporcionen un seqüenciador de vídeo intern o s'integren amb un editor de vídeo. Aquí, pots carregar les imatges renderitzades i organitzar-les en una seqüència per crear el vídeo.

- **Taxa de *framerate*:** defineix la durada de cada fotograma i la velocitat de reproducció per al vídeo. La velocitat de reproducció es mesura en frames per segon (FPS), típicament 24 FPS per a una animació fluida.

Ara, combinant els coneixements generals que tenim sobre animació i els motors de rendering, serem capaços de crear qualsevol imatge. D'aquesta manera, serem capaços de crear també qualsevol seqüència d'aquestes, o el que és el mateix, un vídeo, perquè qualsevol motor dels quals hem revisat abans ens permet crear una seqüència d'imatges entre Keyframes ajustant-se a un *framerate* prèviament establert.

Conclusió

En conclusió, el renderitzat és un procés essencial en la creació d'imatges i animacions 3D realistes i visualment atractives. L'elecció del motor de rendering i les configuracions de sortida són determinants per aconseguir resultats d'alta qualitat. A més, la generació de vídeo implica combinar múltiples fotogrames renderitzats en una seqüència animada per a aconseguir animacions impressionants i fluides. Amb el domini d'aquestes opcions i tècniques, podràs crear projectes d'animació i gràfics 3D d'alt nivell.



Saber-ne més

És interessant consultar els manuals que empren actualment els animadors d'empreses famoses com Disney o Pixar perquè inclouen tot el que hem escrit en aquest document encara que molt més desenvolupat.

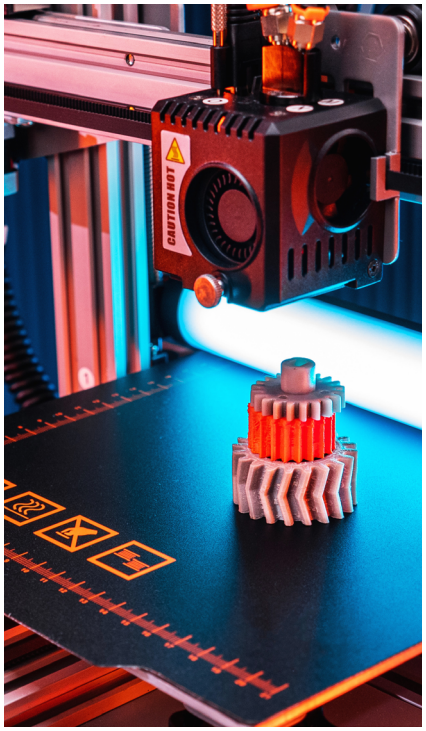
e.digitall.org.es/pixar

Creació de continguts digitals

Nivell C2 3.1 Desenvolupament de contingut

Fabricació d'objectes 3D

Fabricació d'objectes 3D



Introducció

Impressió 3D

Història impressió 3D

La impressió 3D ha revolucionat la manera de fabricar objectes, cosa que permet la creació de prototips, eines i peces personalitzades de manera ràpida i eficient. Una part fonamental del procés d'impressió 3D és l'exportació de models 3D en formats adequats per a la seva posterior impressió. En aquesta primera pàgina, explorarem els formats d'exportació admesos per impressores 3D, així com les aplicacions i eines utilitzades per a la impressió.

Tenim tendència a pensar que la impressió 3D és una cosa purament actual, però en realitat la impressió 3D es remunta a la dècada de 1980, quan el concepte de fabricació additiva va començar a prendre forma. En 1984, Chuck Hull, un enginyer estatunidenc, va inventar el procés d'estereolitografia (SLA), que es considera el primer mètode d'impressió 3D. La SLA permetia crear objectes tridimensionals capa per capa, utilitzant fotopolímers sensibles a la llum ultraviolada. A mesura que van avançar els anys, altres tècniques de fabricació additiva, com la fusió per deposició de material (FDM) i la sinterització selectiva per làser (SLS), van ser desenvolupades i comercialitzades.

Com imprimim en 3D?

Quan vulguem imprimir alguna cosa en 3D hi haurà dues coses que necessitarem principalment. El primer, seria el suport de maquinari, és a dir, una impressora 3D. D'altra banda, necessitarem una peça 3D compatible amb la impressió. Quant al maquinari, és important saber que existeixen principalment dos tipus d'impressora: impressores de resina i impressores de filament.

- **La impressió 3D per filament**, també coneguda com a fDM (*Fused Deposition Modeling*) o FFF (*Fused Filament Fabrication*), és un dels mètodes més comuns i accessibles d'impressió 3D. En aquest procés, un filament termoplàstic, generalment PLA o ABS, és escalfat i extret a través d'un

filtre. El material fos es diposita capa per capa per a construir l'objecte 3D. El model s'imprimeix sobre una plataforma de construcció que es mou en eixos X, I i Z. Una vegada dipositada una capa, la plataforma descendeix i s'inicia la impressió de la següent capa.

- **La impressió 3D per resina**, també coneguda com a SLA (*Stereolithography*) o DLP (*Digital Light Processing*), és una tecnologia d'impressió 3D basada en resina líquida fotosensible. En aquest procés, un làser o un projector DLP exposa selectivament la resina líquida, causant que se solidifiqui capa per capa. A diferència de la impressió per filament, on es construeixen objectes mitjançant la deposició de material fos, la impressió per resina crea objectes submergint una plataforma en la resina i elevant-la de manera controlada a mesura que les capes se solidifiquen.

En resum, utilitzarem una impressora de filament quan no vulguem un gran nivell de detall, però si volem una peça gran, robusta, amb un gran nivell d'acabat superficial o destacar detalls petits d'una peça, llavors hauríem de triar resina. També cal destacar que el filament és l'opció més popular a causa del seu fàcil ús.

En aquest document ens centrarem sobretot en la segona opció, perquè les impressores 3D són un món molt extens i nosaltres ens dedicarem a ampliar i aplicar la formació anterior que tenim sobre editors 3D, en aquest cas per poder donar una forma física al que hem après a modelar. Aprendrem unes certes pautes i formats que hem de seguir si volem que el nostre objecte 3D passi al nostre món com un objecte físic similar al que ens havíem imaginat.

Procés de preparació i impressió

Preparació del model 3D per a impressió

La impressió 3D implica més que simplement carregar un model en la impressora. En aquest punt aprofundirem en el procés de preparació del model, els ajustaments d'impressió necessaris i els passos per dur a terme la impressió en si. Abans d'imprimir, el model 3D ha de ser preparat per assegurar que estigui llest per a la fabricació additiva. Això implica la

revisió del model, la detecció i correcció d'errors geomètrics, i la realització d'ajustos per a garantir l'estabilitat de l'objecte durant la impressió. Així mateix, es poden agregar estructures de suport per a aquells dissenys que ho requereixin. Aquests canvis abans de la impressió els podríem dividir en dues fases, la fase de l'editor i la fase del programari d'impressió.

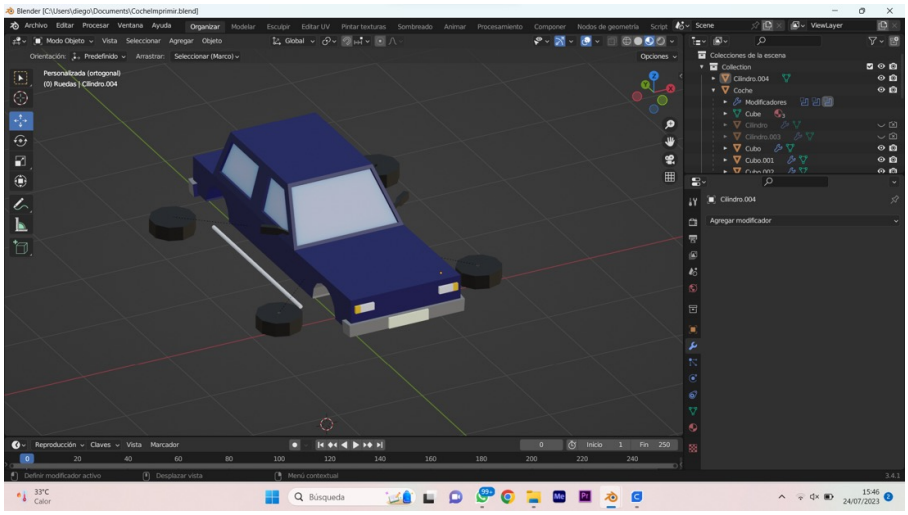
Fase edició

Dins d'un editor, com seria en el nostre cas Blender, hem de tenir bastantes coses en compte. A continuació, deixarem unes pautes:

- **Hem de verificar que ens ajustam a les dimensions i escala reals.** Si bé això és una cosa que podrem corregir a posteriori com veurem en el programari d'impressió 3D, a l'hora de dissenyar una peça que vulguem imprimir hem de tenir molt en compte la grandària de la nostra impressora per no passar-nos si no volem dividir el model en trossos i imprimir-lo en fases.
- **Hem d'optimitzar al màxim el nombre de vèrtexs i polígons.** Neta i optimitza la geometria del teu model. Elimina geometria innecessària i evita triangles i ngons (polígons de més de quatre costats), ja que poden causar problemes durant la impressió.
- **Hem de verificar l'orientació.** Assegura't que l'orientació del model sigui adequada per a la impressió. La base del model ha d'estar plana i en contacte amb la superfície d'impressió. Evita sortints excessivament petits o estructures molt primes, ja que poden ser difícils d'imprimir.
- **No hem de col·locar elements flotants.** Si el teu model té peces mòbils o elements separats, assegura't que estiguin connectats adequadament per a evitar problemes d'impressió i acoblament posterior.
- **No hem d'emprar cares totalment planes**. En el món real, no existeixen els objectes totalment plans, i això és una cosa que hem de tenir en compte perquè les impressores 3D imprimeixen objectes sòlids, i per això assegura't que el teu model tengui un gruix adequat en totes les parts. Si el teu model és només una superfície, converteix-la en un objecte 3D sòlid mitjançant el modificador Solidify a Blender.

- **No hem de superposar peces**. Assegura't que les parts del model no se superposin, ja que això pot causar problemes d'impressió i donar lloc a peces fusionades o distorsionades.

Un altre consell que hauríem de tenir en compte seria exportar sempre els arxius en els formats estàndard STL i OBJ que treballen bé amb qualsevol programari d'impressió 3D.

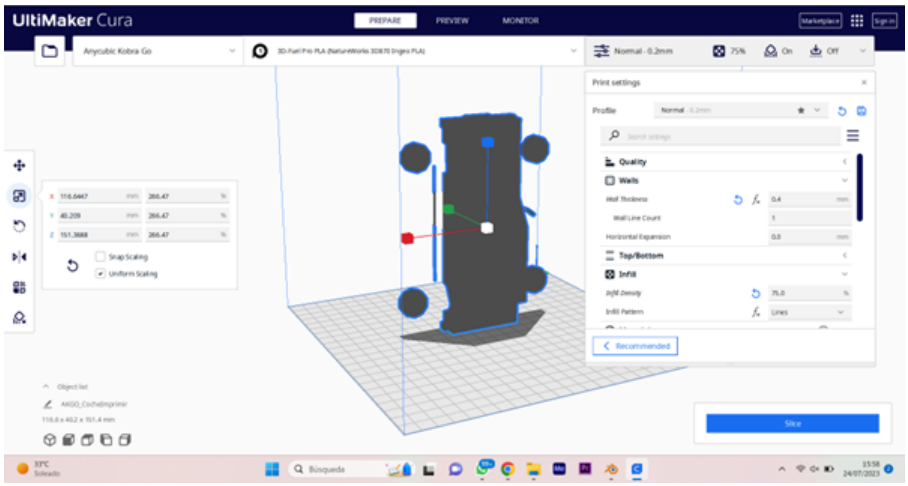


En aquesta captura podem observar un model preparat per a impressió a Blender. Totes les peces estan a nivell del sòl amb el qual no només no tenim peces flotants, sinó que a més hem aconseguit en una mateixa impressió més peces. Cal afegir que és aconsellable tenir activada la vista de renderitzat, ja que, encara que el color no s'imprimirà, pot ser interessant tenir-la perquè ens serveixi de guia.

Fase programari d'impressió

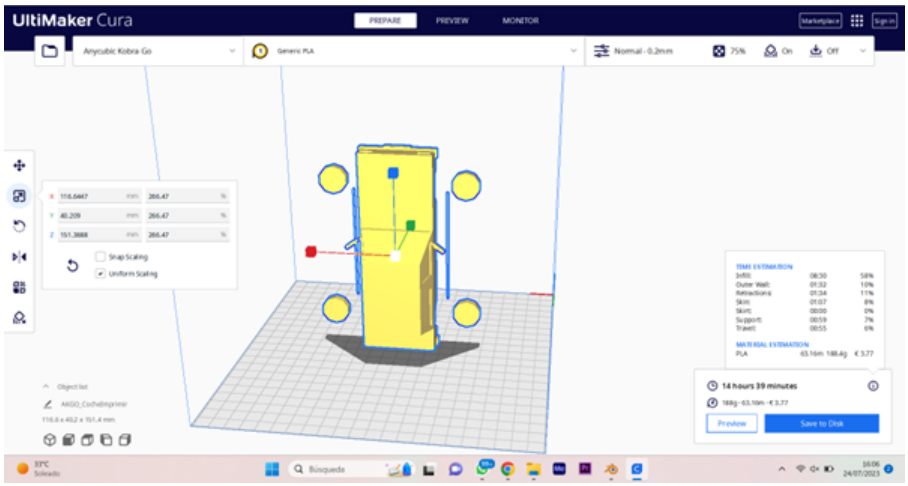
Prendrem com a programari predefinit CURA, una aplicació dissenyada per a impressores 3D gratuïta, en la qual es poden modificar els paràmetres d'impressió i després transformar-los a codi G, el necessari per a qualsevol arxiu destinat a ser imprès.

El primer que s'ha de fer en obrir CURA seria seleccionar el perfil d'impressora corresponent i carregar el teu model 3D en format STL o OBJ. Després, anar a la pestanya "Preparar". Després d'això tendrem molts paràmetres a modificar com la velocitat d'impressió, gruix o temperatura del llit, que de moment deixarem en el seu estat predeterminat perquè per defecte ens serviran per a qualsevol primera peça que vulguem imprimir.



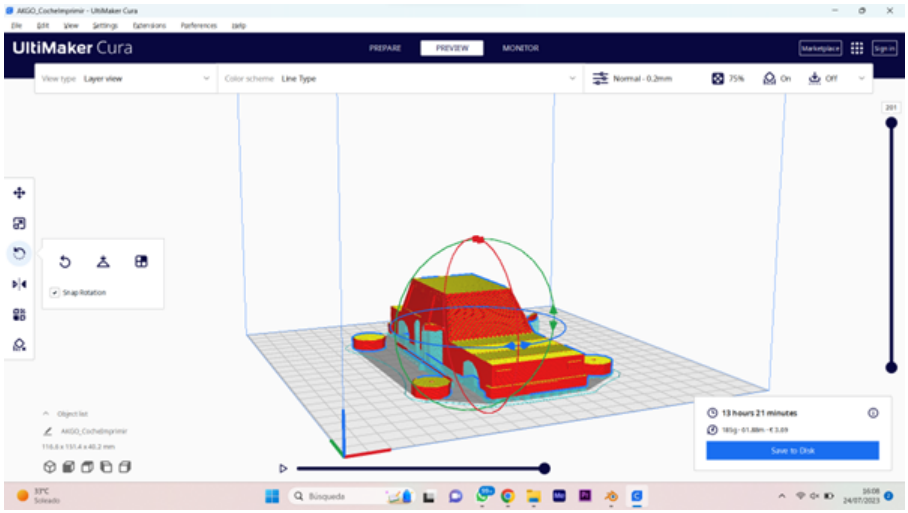
Aquesta seria la primera impressió que tendríem del programa CURA una vegada importat el model. Podem observar que per moure el model tenim les mateixes eines que en qualsevol altre editor 3D. També a la dreta podem observar les opcions esmentades en el paràgraf anterior.

Addicionalment, pots afegir estructures de suport des de la pestanya "Suport" (Support) si el teu model el requereix perquè hàgim deixat alguna peça allunyada del cos principal o no hàgim fet cas a la pauta de no deixar parts "flotants". Revisa la vista de capes per assegurar-te que tot estigui en ordre. Després d'això, ens apareixerà una estimació de temps i si hem ajustat el filament o resina que emprarem i el seu preu. Si alguna d'aquestes dues opcions no fos correcta, podríem modificar els paràmetres de manera que aquestes dues abaixin (temps i preu). Finalment, desa l'arxiu G-code. Aquest arxiu conté les instruccions específiques per a la teva impressora i és el que utilitzaràs per a la impressió.



Amb l'*slice* ara podrem veure que (efectivament) se'ns calcula el temps i el preu (sempre que tenguem aquest precisat en preferències). També podem consultar una estimació precisa de quant temps ocuparà cada procés de la impressió detallat de manera senzilla.

No obstant això, abans d'imprimir la peça que vulguis, podries provar d'imprimir una versió més petita i comprovar si aquesta ha sortit amb algun desperfecte que puguis corregir abans d'imprimir la versió completa.



Comprovant defectes en l'apartat de *preview* amb l'anterior model, hem pogut comprovar que la manera òptima d'imprimir el model era en horitzontal, creant-se així suports per als retrovisors i facilitant el treball post impressió.

Abans d'acabar aquest punt, ens agradaria emfatitzar que no tot és modelar i preparar el disseny per imprimir-lo, també existeix un procés important post impressió que requerirà temps i dedicació perquè el model quedi exactament com el tenim a l'ordinador. Les impressores mai seran 100% precises perquè, com ja hem comentat, serà necessari imprimir suports que farà falta llevar a mà i deixaran rebaves i imperfectes que se sumaran a altres causats per petits defectes que tota impressora té. Per això és important saber que, a més nivell de detall que necessitis, més treball d'escatat post impressió i cura necessitaràs.

Amb tot el vist fins ara, ja tendríem generat un arxiu bàsic per poder imprimir en 3D.

Procés d'impressió i consideracions finals

Impressió del model

A continuació, abordarem el procés d'impressió pròpiament dit i algunes consideracions finals per a obtenir resultats de qualitat i garantir la seguretat en la impressió 3D.

Amb el model 3D preparat i havent seguit els passos definits en el punt anterior, el procés d'impressió pot començar. Durant la impressió, la impressora diposita capes successives del material seleccionat per construir l'objecte en 3D. És essencial supervisar la impressió per detectar problemes potencials i garantir un resultat reeixit.

Sobretot, si utilitzam una impressora de resina, hem de tenir en compte que la impressió 3D involucra l'ús de materials i processos específics que poden presentar riscos potencials si no es manegen adequadament. És important seguir les pautes de seguretat proporcionades pel fabricant de la impressora i prendre mesures per evitar lesions o danys durant el procés d'impressió.

Conclusions finals

La impressió 3D ha revolucionat la fabricació i ha obert un món de possibilitats en diverses indústries. Amb el coneixement dels formats d'exportació, el programari de modelatge i les consideracions d'impressió adequades, es poden crear objectes 3D personalitzats i funcionals que abans eren difícils d'imaginar.

Saber-ne més

Sempre podràs consultar el complet article d'Adam Jorquera Ortega que ens parla més en profunditat sobre la impressió 3D i el modelatge exclusivament dedicat a aquest objectiu, "***Fabricació digital: Introducció al modelatge i impressió 3D***"

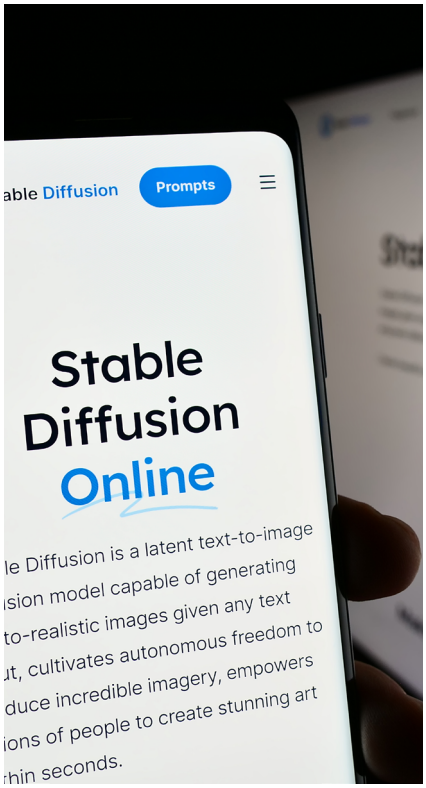
e.digitall.org.es/fabricacion-digital

Creació de continguts digitals

Nivell C2 3.1 Desenvolupament de contingut

Requisits i instal·lació local d'eines d'intel·ligència artificial per a la creació de vídeo i àudio

Requisits i instal·lació local d'eines d'intel·ligència artificial per a la creació de vídeo i àudio



Avaluació de necessitats

En general, la decisió d'instal·lar una IA com Stable Diffusion en local dependrà de les nostres necessitats específiques, requisits de privacitat i seguretat, així com els recursos i capacitats tècniques disponibles. Hem de tenir en compte que aquest procés tendrà uns requisits de maquinari, que comentarem posteriorment, però, a més, treballar en local ens fa als usuaris responsables del manteniment del sistema, així com de les seves actualitzacions i pogués ocórrer que, si no tenim una màquina massa potent tinguem problemes d'escalabilitat, és a dir, que no tinguem tanta capacitat per manejar dades com ens proporciona la mateixa eina al núvol.

No obstant això, aquest procés té també alguns clars avantatges, entre les quals destaquen:

- Tindrem un **control total sobre les nostres dades**. Això serà especialment important en casos on la privacitat i la confidencialitat siguin fonamentals, ja que les dades no surten del nostre entorn local.
- Disposarem d'una major velocitat i latència reduïda. En executar la IA en local, evitem la dependència d'una connexió a Internet per al processament de dades, així com possibles fallades per caigudes en la xarxa o limitacions en l'amplada de banda que tenguem contractat. Per tant, la IA pot respondre de manera més ràpida, la qual cosa és beneficiós en aplicacions que requereixen temps de resposta ràpids.
- Majors graus de personalització i ajust. Ens trobarem amb una major flexibilitat a l'hora de personalitzar i ajustar el model a les nostres necessitats específiques. Podrem adaptar paràmetres del sistema o entrenar a la IA amb les nostres dades, la qual cosa ens proporciona més adaptabilitat i un millor rendiment en funció dels nostres objectius. De fet, si tenim els suficients coneixements, podrem modificar el codi font de la nostra versió local

de *Stable Diffusion* per adaptar-ho als nostres requisits particulars.

Requisits d'instal·lació

La instal·lació de Stable Diffusion en local presenta uns requisits en termes de maquinari, així com de recursos computacionals.

D'una banda, no tenim requisits de processador, però necessitarem un disc dur SSD (recomanat) de més de 256 GB, amb almenys 25 GB lliures, així com un mínim de 8 GB de memòria RAM. D'altra banda, es recomana una targeta gràfica dedicada NVIDIA amb, almenys, 2 GB de VRAM per mostrar resultats de manera més àgil i precisa amb capacitat suficient per a poder usar l'ordinador de manera normal.

Lògicament, com més potent sigui la màquina en la qual treballem, més ràpid obtindrem els nostres resultats. Es calcula que Stable Diffusion instal·lat en local només necessita 5 segons per a generar una imatge de 512x512 píxels usant una NVIDIA 3060 de 12 GB.

Tutorial d'instal·lació

En l'actualitat, s'ha simplificat bastant el procés d'instal·lació local de Stable Diffusion. Hem de començar per anar a la pàgina de GitHub de Stable Diffusion UI, on indicarem en quin sistema operatiu estem treballant per baixar-nos un programa d'instal·lació convencional (Figura 1).

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

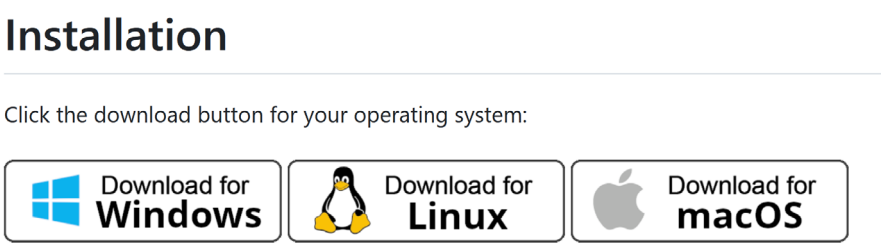


Figura 1. Opcions de selecció de sistema operatiu per a la instal·lació local de *Stable Diffusion*.

Després només haurem d'executar l'instal·lador i acceptar l'acord de llicència per instal·lar Stable Diffusion al nostre PC. La ruta d'enllaç suggerida en la instal·lació és: **C: \ EasyDiffusion** (Figura 2).

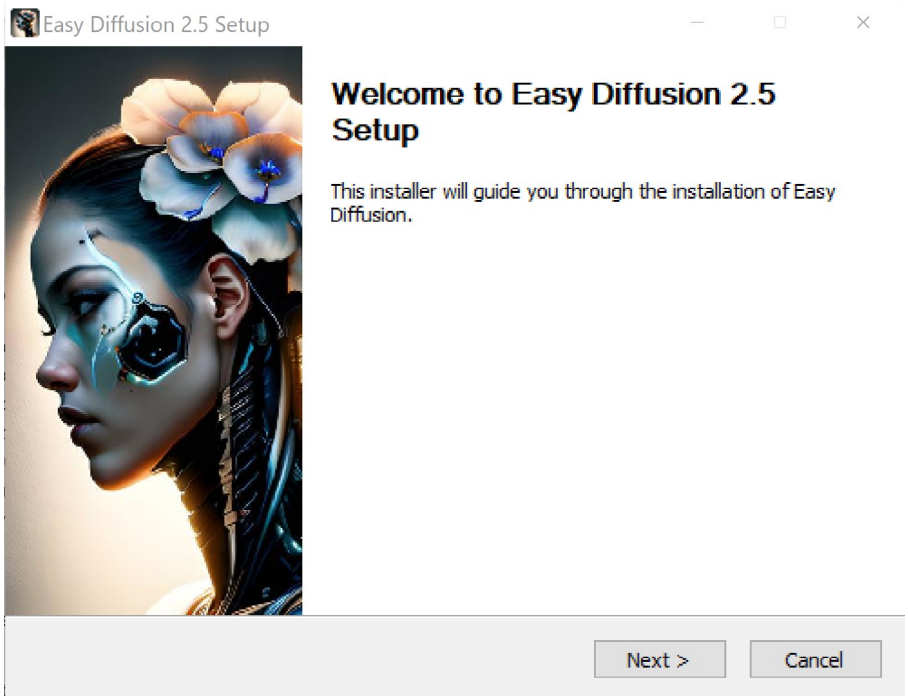


Figura 2. Visió del programa d'auto instal·lació de *Stable Diffusion*.

Una vegada finalitzada la instal·lació podrem executar el programa (*Start Stable Diffusion UI.cmd*) i se'ns obrirà una finestra de CMD, en la qual veurem que es continuen descarregant components necessaris per al correcte funcionament del sistema.

Ja descarregats i instal·lats la resta dels arxius necessaris, se'ns obrirà la interfície de l'aplicació en el nostre navegador predeterminat en l'adreça ***http://localhost:9000/***. Si estàs en Windows, probablement surt-te un avís del tallafocs (Figura 3).

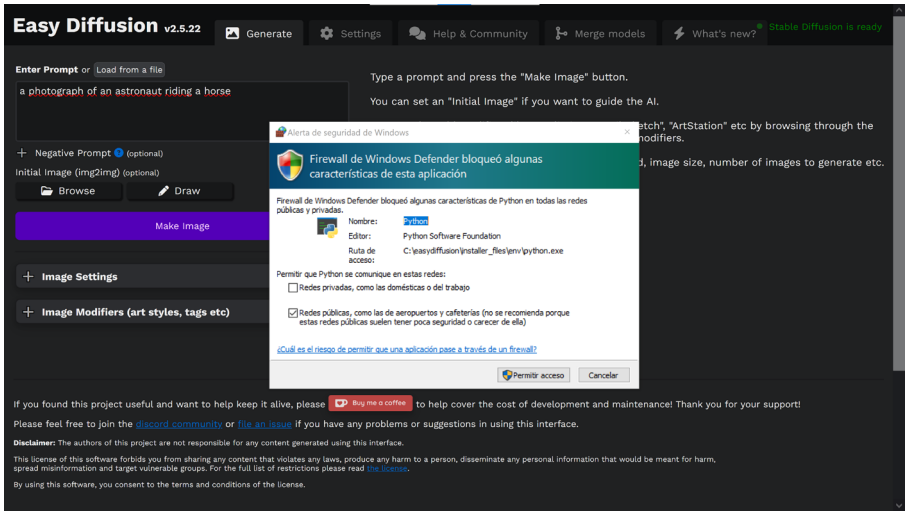


Figura 3. Aspecte del navegador en Windows la primera vegada que s'executa la instal·lació local de *Stable Diffusion*.

Una vegada establerts els permisos del firewall, podrem operar amb la nostra instal·lació local de Stable Diffusion sense problemes. Podem veure exemples d'ús d'aquesta instal·lació local al vídeo:

ÚS LOCAL D'EINES D'INTEL·LIGÈNCIA ARTIFICIAL PER A LA CREACIÓ D'IMATGES A PARTIR D'INFORMACIÓ TEXTUAL

e.digitall.org.es/A3C31C2V08

Saber-ne més

Si vols saber més sobre aquesta instal·lació de Stable Diffusion en el teu propi entorn local, et volem convidar a explorar la seva pàgina oficial en GitHub. A més, en tractar-se d'un programari de codi obert, existeixen en l'actualitat altres versions en GitHub per a instal·lar Stable Diffusion de manera local. Finalment i també per aquest mateix motiu, pot ser que fins i tot quan llegeixis això totes aquestes versions que ara mateix estan disponibles ja hagin estat substituïdes per programes més actuals.

e.digitall.org.es/diffusion

DigitAll

Creació de
continguts digitals

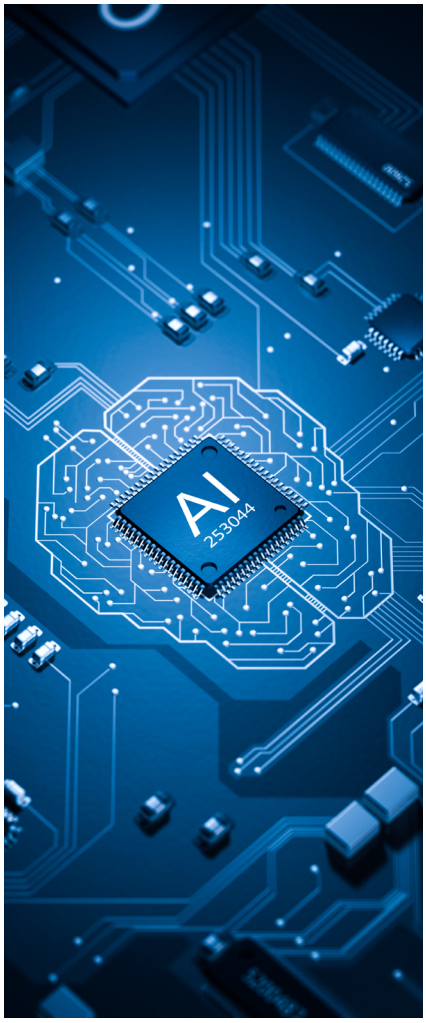
3.2
INTEGRACIÓ I REELABORACIÓ DE CONTINGUT DIGITAL

Creació de continguts digitals

Nivell C2 3.2 Integració i reelaboració de contingut digital

Eines d'intel·ligència artificial per a la presa de decisions

Eines d'intel·ligència artificial per a la presa de decisions



L'**aprenentatge automàtic** és un tipus d'intel·ligència artificial basada en dades i inclou un conjunt de tècniques que permeten que els ordinadors i els microcontroladors aprenguin a fer unes tasques determinades. La integració d'aquestes tècniques en sistemes maquinari-programari enriqueix les regles informàtiques i allibera els programadors d'escriure la lògica d'un programa que faci que a una determinada entrada **x** li correspongui la resposta y. Aquests mètodes actuen com a **caixes negres** que reben una entrada **x** i retornen una sortida **y**. Aquests mètodes poden guiar algunes de les accions del sistema en introduir una certa intel·ligència en ell.

Suposem que el nostre sistema maquinari-programari té un vector de variables d'entrada **x**, per exemple, recollits per un conjunt de sensors, i el sistema ha de decidir una variable resposta **y**. Aquesta variable pot recollir la resposta per a un conjunt d'actuadors o una dada interna necessària per a continuar processant. La classificació dels problemes d'aprenentatge automàtic es basa en dos criteris: I) la possibilitat o no de conèixer la variable resposta (i), i II) la tipologia d'aquesta variable. Es distingeix la següent tipologia de les variables:

- **Variables quantitatives**. Les dades prenen valors numèrics. Al seu torn es distingeix entre variables contínues que prenen valors en un interval de nombres reals o **variables discretes**, que prenen un nombre finit de valors numèrics.
- **Variables ordinals**.Les dades expressen relació d'ordre entre les observacions. Per exemple, podem considerar un *rànquing*.
- **Variables qualitatives**. En aquest cas s'expressa una qualitat d'un objecte. Aquestes variables només poden prendre un conjunt de valors que no mesuren cap magnitud determinada. Un exemple són les variables **binàries** en les quals només es poden prendre dos valors i expressen la presència/absència d'una característica.

Una **taxonomia dels algorismes** d'**aprenentatge automàtic** es basa en la naturalesa de les variables del sistema x i y. Es distingeix entre aprenentatge supervisat en el qual partim d'un conjunt de dades etiquetat prèviament, és a dir, coneixem un conjunt de dades (**xy**,**yi**). L'aprenentatge no supervisat part de dades no etiquetades prèviament, això és, exclusivament es disposa de la informació (**xy**). Dins de l'aprenentatge no supervisat apareixen els problemes de clustering, reducció de la dimensionalitat i detecció d'***outliers***. El primer determina patrons del sistema. El segon redueix la grandària de les dades **x** generats, intentant perdre el mínim d'informació. Per exemple, si els sensors realitzen mesuraments en temps real poden generar grans quantitats dades que ha de ser transmesos. Un exemple notable és la transmissió d'imatges pels satèl·lits. Si els dispositius tenen una capacitat de comunicació reduïda és llavors recomanable aplicar aquestes tècniques. La detecció d'outliers permeten reconèixer que l'estat del sistema programari-maquinari està en un estat (patró) diferent dels a priori planificats per al seu funcionament i, per tant, algun tipus d'alarma o notificació s'ha d'enviar.

En aprenentatge automàtic apareixen tres tipus de problemes bàsics:

1 | **Clustering**. En aquest problema es cerquen patrons dins de les dades. Això és, trobar subconjunts d'observacions que són similars entre si. Matemàticament, es formularia com determinar les etiquetes y de les dades de manera que si dues observacions i i j són similars li associam la mateixa etiqueta, Yi=Yj. Una dificultat d'aquest problema és que no es parteix d'un conjunt previ d'etiquetes del qual aprendre. Un exemple d'aquesta mena de problemes és l'agrupació de dades. Això permet identificar un conjunt finit de patrons en el qual es pot trobar el sistem

2 | **Regressió**. En aquest problema es prediu el valor d'una variable contínua **y** coneguda el valor de la variable **x**. En els contextos de regressió la variable **x** se'l denomina regressor o variable explicativa. Aquests sistemes estimen funcions de regressió **y=f(x)** i permetrien per exemple donar el valor y d'un cert actuador per a l'estat actual del sistema **x**.

3 | Classificació. En aquest problema s'ha de predir el valor de la variable qualitativa **y** a partir de la variable **x**. En classificació la variable **x** es denomina característiques o atributs mentre que **y** es denomina etiqueta o classe. Per exemple, en el cotxe autònom es basa en tres pilars: IoT, tècniques d'aprenentatge automàtic aplicat a Big Data i connexió a Internet en temps real. En la construcció dels ulls del vehicle es requereix resoldre el problema de classificar les imatges que envolten al vehicle i poder així determinar quin tipus d'objecte està davant, darrere o als costats del vehicle.

La Figura 1 resumeix aquesta classificació dels models d'aprenentatge automàtic. Per a cada tipus de problemes s'han proposat multitud d'algorismes. No existeix un mètode que sigui millor que un altre per a totes les circumstàncies. Per tant, és necessari disposar d'una caixa d'eines amb les quals poder assajar diferents solucions. Per exemple, l'algorisme K-means és ràpid i funciona bé per a problemes de *clustering* en els quals els patrons es poden separar linealment (mitjançant hiperplans). En altres situacions aquest algorisme pot no funcionar bé i, per tant, caldria recórrer a tècniques alternatives com a algorismes basats en densitat (DBSCAN). Un altre exemple són les **xarxes neuronals profundes** que mostren un alt rendiment en tots aquests problemes. No obstant això, per a uns certs dispositius, malgrat la seva eficàcia, el cost computacional les pot fer inaplicables.

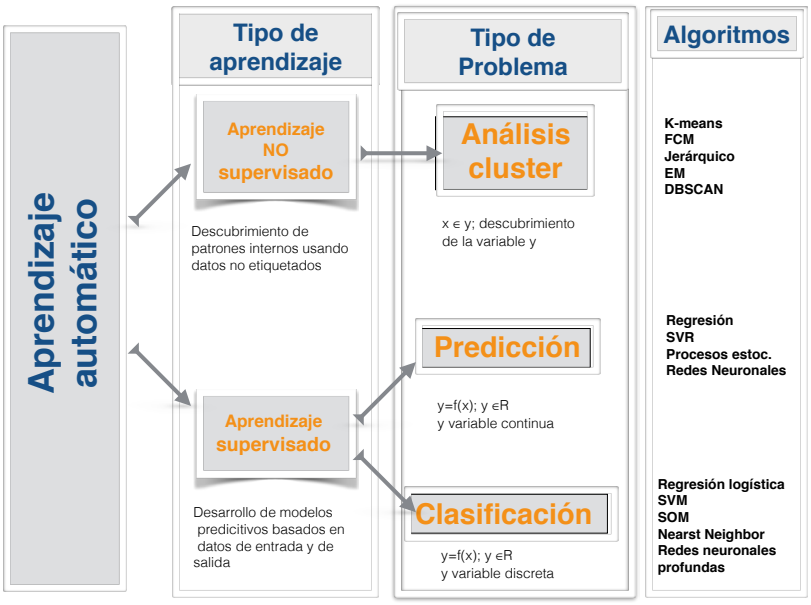


Figura 1. Classificació de problemes d'aprenentatge automàtic.

Aquestes tècniques empren un **aprenentatge inductiu**, ja que a partir de l'observació i l'anàlisi d'exemples concrets es desenvolupen models que expliquen aquestes dades i que permeten dur a terme una generalització. El modelador ha d'entrenar els models sobre un conjunt de dades disponibles i els models entrenats són introduïts en els sistemes.

La Taula 1 mostra algunes de les eines més populars per implementar models d'aprenentatge automàtic. L'elecció de l'eina depèn de les necessitats i preferències del projecte.

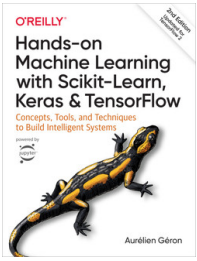
Taula 1. EINES PER A CONSTRUCCIÓ DE MODELS D'APRENENTATGE AUTOMÀTIC

Nom	Característiques	Desenvolupador
TensorFlow	Col·lecció eines per a la creació, entrenament i implementació de models d'aprenentatge automàtic. Empra com a llenguatge de programació Python i C++.	Google
scikit-learn	És una biblioteca d'aprenentatge automàtic de codi obert per a Python.	
PyTorch	Framework popular d'aprenentatge profund. Ofereix una interfície més flexible que TensorFlow.	Facebook
Keras	Keras és una biblioteca de Xarxes Neuronals de codi obert escrita a Python. És capaç d'executar-se sobre TensorFlow.	
Microsoft Azure ML	Plataforma en el núvol per al cicle complet de preparació de dades, implementació de models d'aprenentatge automàtic i monitoratge en producció.	Microsoft

Saber-ne més

Un llibre excel·lent per a integrar en Python aquestes tècniques és ***Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow***.

e.digitall.org.es/hands-on



Creació de continguts digitals

Nivell C2 3.2 Integració i reelaboració de contingut digital

Eines de desenvolupament de robots programables

Eines de desenvolupament de robots programables

Els robots programables han guanyat un gran protagonisme en els darrers anys, gràcies a la seva capacitat per a automatitzar tasques i millorar l'eficiència en diferents àmbits, com la indústria, l'educació o la llar. Dissenyar i programar robots no és una tasca senzilla, ja que requereix coneixements especialitzats en diferents àrees, com a electrònica, mecànica o informàtica.

No obstant això, existeixen diverses eines i plataformes que faciliten aquest procés i permeten a qualsevol persona, independentment del seu nivell d'experiència, desenvolupar robots de manera senzilla i ràpida.

La principal eina sobre la qual es treballa són les denominades **plaques programables**, com **Arduino** o **Raspberry Pi**. Aquestes plaques són petits dispositius que compten amb un microprocessador de baix consum connectat a diferents components, com a mòduls de memòria RAM, connexió per a l'emmagatzematge de dades a través d'una targeta de memòria, components d'entrada-sortida (ports USB, ports Ethernet, mòduls Wi-Fi i Bluetooth, pins de connexió GPIO, etc.) i, fins i tot, petites targetes gràfiques o processadors d'imatge. Aquestes plaques poden connectar-se a diferents sensors i actuadors i controlar el seu funcionament a través de programes escrits en diferents llenguatges de programació.

Una vegada es té la placa programable desitjada, una altra eina fonamental és el llenguatge de programació. Alguns dels llenguatges més populars són **C**, **C++**, **Python** i **Java**. Aquests llenguatges permeten escriure codi que indiqui al robot com ha de funcionar i com ha d'interactuar amb el món exterior. Cada placa suportarà un conjunt diferent de llenguatges de programació:

- Per exemple, les plaques Arduino utilitzen un llenguatge de programació propi basat en C++ i denominat Arduino. No obstant això, existeixen eines de programació que permeten l'ús d'altres llenguatges alternatius com per exemple C, C# o Python, que posteriorment es converteixen en Arduino abans de ser enviats a la placa.

- Per part seva, la placa Raspberry Pi disposa d'un sistema operatiu propi basat en Linux denominat Raspbian. Per això, la placa suporta nombrosos llenguatges de programació com a C, C++, Python, Java, Ruby, Perl, etc.

Saber-ne més

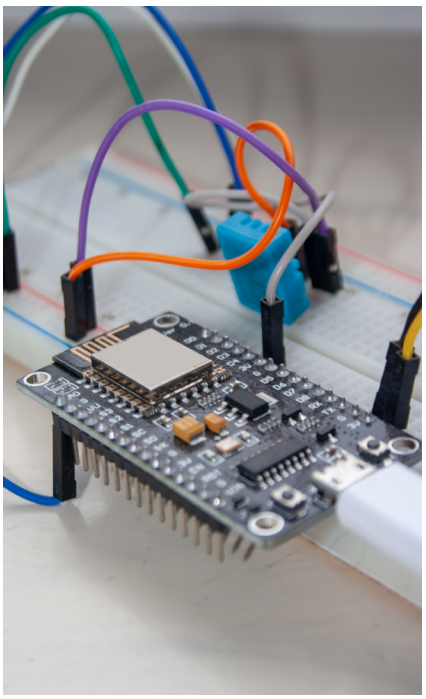
Encara que Raspbian és el sistema operatiu més conegut i utilitzat per a Raspberry Pi, existeixen una altra gran varietat de sistemes operatius compatibles. Per exemple: Windows 10 IoT Core, Ubuntu Core/Desktop/Server, Raspbian, OSMC, Kali Linux, etc.

A més dels **llenguatges de programació,** existeixen diferents aplicacions de desenvolupament i simuladors que permeten dissenyar i simular el comportament dels robots abans de construir-los. Aquestes eines complementen als llenguatges de programació incrementant les seves possibilitats i afegeixen capacitats de simulació. Els **simuladors** són especialment útils per a provar diferents escenaris en un entorn virtual i optimitzar el funcionament del robot sense haver d'esperar a construir-lo físicament i provar-lo en el món real. Gràcies a aquestes aplicacions s'aconsegueix evitar danys o errors durant el desenvolupament del robot, els quals poden ser molt costosos, ja que comportaria tornar a l'etapa de disseny, programació i construcció del robot.

Saber-ne més

Un simulador senzill i gratuït per a Arduino es **SimulIDE**, disponible a: *simulide.com*. Aquest simulador està disponible para Windows, MacOS y Linux.

El disseny i la programació de robots requereixen una gran combinació de coneixements i habilitats específics, però gràcies a l'aparició d'unes certes eines de desenvolupament per a un públic general, **qualsevol persona pot aprendre a crear el seu robot programable**. A més, algunes d'aquestes eines permeten programar els robots sense necessitat d'escriure codi o saber programar. Per exemple, per a Arduino existeix l'eina ArduBlock.



ArduBlock és una eina que permet programar a Arduino utilitzant blocs gràfics, com si es tractés de peces de puzles.

Aquesta eina permet elaborar petits programes sense necessitat de tenir experiència prèvia en programació. Actualment, aquesta eina es troba accessible a: ***ardublock.ru/en***. La Figura 1 mostra un exemple d'un projecte elaborat a ArduBlock.

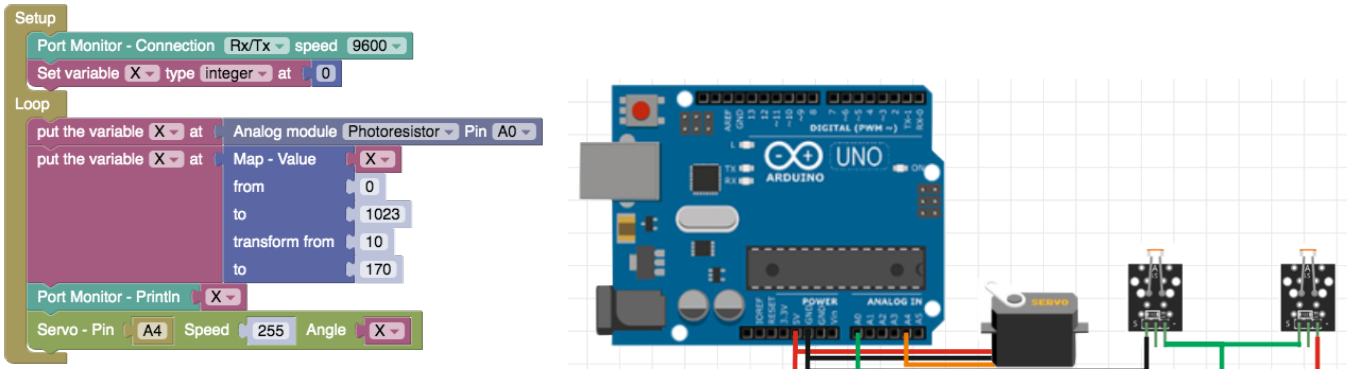


Figura 1: Exemple d'un projecte a ArduBlock per gestionar un servomotor que segueix la trajectòria del sol mitjançant l'ús de dos fotorresistores. Font: ardublock.ru

Saber-ne més

Scratch és una eina de programació mitjançant blocs desenvolupada pel MIT i orientada a la iniciació a la programació. Aquesta eina està enfocada principalment en la programació de videojocs. Existeix una modificació de Scratch denominada **S4A**(*s4a.cat*), la qual permet programar en Arduino utilitzant el llenguatge Scratch.

També existeixen els denominats kits de desenvolupament o de robotització, que són petits kits comercials que inclouen tots els components necessaris per a construir i programar un robot. Alguns kits inclouen les mateixes plaques programables, un ampli conjunt de sensors i actuadors i tot el cablejat necessari, així com eines de programació específiques. Aquests kits solen venir amb guies i tutorials que faciliten el procés de muntatge i programació.

La mateixa web del projecte Arduino ofereix un kit d'iniciació denominat **Arduino Starter Kit** (***e.digitall.org.es/arduino***). No obstant això, existeixen multitud de kits per a Arduino en múltiples botigues d'electrònica. Des de kits per a infants, com

a kits per a entusiastes de la tecnologia o gent que vol iniciar-se. També hi ha una gran varietat de kits per a Raspberry Pi, alguns dels quals permeten construir el teu ordinador personal. Per exemple, el **Raspberry Pi 400 Personal Computer Kit** (*e.digitall.org.es/raspberry*).

Saber-ne més

Una altra eina important són els fòrums i comunitats en línia, on es poden trobar tutorials, exemples i consells d'altres usuaris que també estiguin interessats en el desenvolupament de robots. Aquestes comunitats solen ser una font inesgotable d'inspiració i aprenentatge.

En resum, existeixen diferents eines i paradigmes que faciliten el desenvolupament de robots programables. Des dels llenguatges de programació de propòsit general aplicats a la programació d'aquestes plaques, les aplicacions de desenvolupament, els simuladors i els kits de desenvolupament. Cadascun d'aquests enfocaments té els seus avantatges i desavantatges, i és important triar l'eina adequada en funció de les necessitats del projecte.

DigitAll

Creació de
continguts digitals

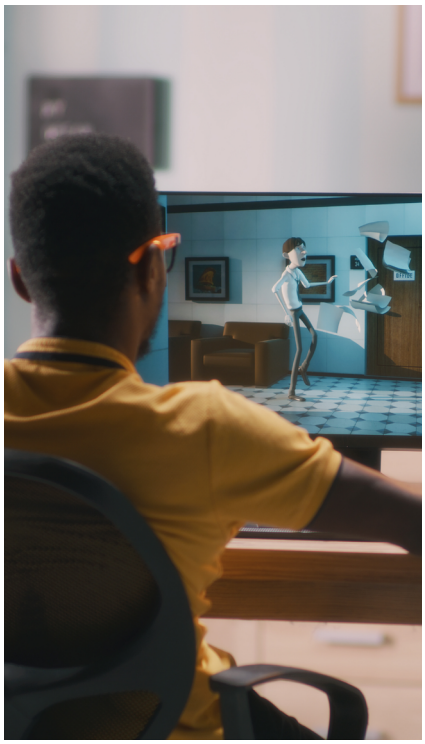
3.3
DRETS D'AUTOR I LLICÈNCIES DE PROPIETAT INTEL·LECTUAL

Creació de
continguts digitals

Nivell C2 3.3 Drets d'autor i llicències de propietat intel·lectual

Registrant el copyright i explotant una obra creada amb ajuda de la IA

Registrant el copyright i explotant una obra creada amb ajuda de la IA



El contingut d'aquest document et farà reflexionar sobre un tema de summa actualitat.

Al dia d'avui, els creadors disposen a Internet d'eines revolucionàries que fan ús de la intel·ligència artificial per a la creació de tota mena de contingut artístic.

Aquestes eines permeten a artistes, músics, escriptors i altres creadors explorar noves fronteres i trobar inspiració en aquesta mena d'eines.

En l'àmbit musical, permeten compondre melodies complexes, harmonies bastant atractives i ritmes fins ara no escoltats. D'altra banda, el camp de la pintura pot generar reproduccions fidels dels grans mestres i noves interpretacions artístiques.

Tots coneixem ja l'èxit de l'eina ChatGPT per generar textos originals. Altres exemples són l'eina DALL·E 2 que genera imatges a partir de descripcions que se li proporcionen, AIVA, que compon pistes de música originals un altre exemple és l'eina AI Time Machine, que permet als usuaris crear imatges d'una persona en diferents períodes de temps al llarg de la història.

Però què passa amb totes les obres que es creuen usant aquestes eines. És possible obtenir el copyright d'una obra creada amb l'ajuda de la **intel·ligència artificial**?

Però què passa amb totes les obres que es creuen usant aquestes eines. És possible obtenir el copyright d'una obra creada amb l'ajuda de la "intel·ligència artificial"?
En la majoria dels països, es poden protegir per mitjà del **Dret d'autor o Copyright les obres originals creades per un ésser humà.**

En les dècades passades, alguns creadors necessitaven la computadora i la utilitzaven com un instrument per crear les seves obres, de la mateixa manera que un escriptor utilitzava el bolígraf i el paper, un altre utilitzava un programa d'ordinador per escriure la seva obra, el pintor cap a ús del pinzell i el llenç i el creador de música electrònica necessitava un sintetitzador. En tots aquests casos prevalia sempre la creativitat del creador.

No obstant això, la revolució tecnològica que està tenint lloc en els darrers anys, impulsada, en part, pel desenvolupament del "**programari d'aprenentatge automàtic**", ens obliga a repensar la interacció entre les computadores i el procés creatiu quan intervé l'esmentat programari.

Saber-ne més

El programari d'aprenentatge automàtic, una de les formes de la IA, és un programa informàtic que pot aprendre a partir de les dades que se li van introduint al mateix programari. Amb la introducció de noves dades evoluciona i presa noves decisions que poden ser dirigides o autònomes.

Quan un creador, per exemple, un pintor, escriptor o compositor, utilitza un programari d'aprenentatge automàtic per a desenvolupar la seva obra, en el procés creatiu el mateix programari va aprenent a partir de la informació que el creador o programador va introduint i a partir d'aquestes dades, comença a prendre decisions independents que donen com a resultat una nova obra d'art. El que ocorre en realitat en aquests casos és que, **si bé el creador defineix alguns paràmetres en el procés creatiu, l'obra és generada pel programa informàtic.**

En casos com l'anterior, podríem concloure que el programa informàtic ja no és una eina que utilitza el creador, com en els casos que es van esmentar anteriorment, sinó que pren decisions associades al procés creatiu sense que intervingui el creador, per la qual cosa es podria entendre que el creador és el mateix programa i no l'ésser humà. Tenint en compte la legislació espanyola, una obra creada utilitzant un programari d'aprenentatge automàtic, que és un subgrup de la intel·ligència artificial no podria registrar-se per Dret d'autor.

En el cas que les obres creades amb la intervenció de la intel·ligència artificial no es poguessin protegir mitjançant el Dret d'autor per considerar-se que no han estat creades per l'ésser humà qualsevol persona podria utilitzar-les lliurement sense cometre plagi, la qual cosa seria un problema molt seriós per a empreses del sector que embenin aquestes obres. Per exemple, hem de pensar en la productora de pel·lícules que inverteix molts diners a desenvolupar un sistema que generi música per a les seves pel·lícules i posteriorment la llei no li permetés protegir-les, qualsevol persona en el món podria

utilitzar-la lliurement, ocasionant un gran perjudici econòmic a l'empresa.

En l'actualitat és un problema que no està resolt, no obstant això, hi ha indicis que la legislació de nombrosos països no és favorable al dret d'autor que no s'aplica a l'ésser humà, com ja s'han pronunciat als Estats Units i Austràlia i molts països europeus. Així i tot, en altres països com Hong Kong, l'Índia, Irlanda, Nova Zelanda i el Regne Unit estarien disposats a concedir-li l'autoria a la persona que realitza els arranjaments necessaris per a la creació de l'obra".

En un futur pròxim, amb l'avanç de la informàtica, quan l'ús de la intel·ligència artificial estigui més generalitzada i la majoria dels creadors facin ús del programari d'aprenentatge automàtic, els ordinadors produiran obres creatives cada vegada millors i podria ser difícil distingir entre una obra d'art feta per un ésser humà i la realitzada per la màquina i llavors caldrà decidir quin tipus de protecció se'ls hauria de concedir a les obres creades amb poca o cap intervenció humana.

Des del punt de vista de l'explotació comercial de l'obra, l'assenyat seria concedir el dret d'autor a la persona que fa possible el funcionament del programari d'aprenentatge automàtic, d'aquesta manera, d'una banda, es garantiria la continuïtat de la indústria creativa com la sostenibilitat de l'artista que viu de l'explotació de les seves obres, i per un altre, que les empreses continuïn invertint en la tecnologia, amb la seguretat de saber que obtindran rendiments de la seva inversió.

Saber-ne més

En aquest article trobaràs informació sobre intel·ligència artificial (IA) explicada de manera molt senzilla i clara:

e.digitall.org.es/inteligencia-artificial

DigitAll

Creació de
continguts digitals

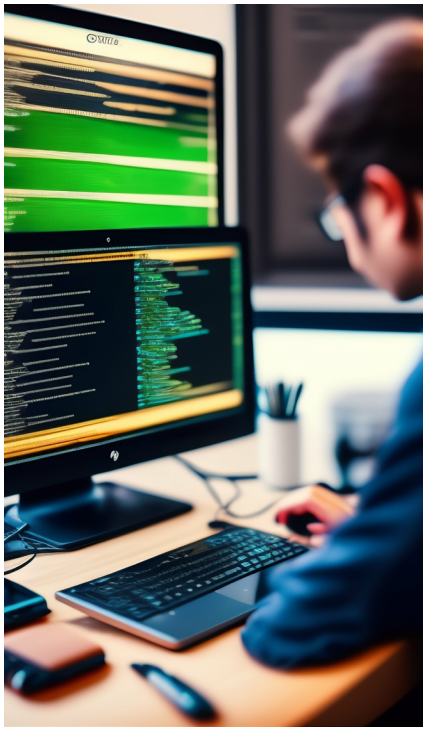
3.4
PROGRAMACIÓ

Creació de
continguts digitals

Nivell C2 3.4 Programació

Paradigmes de programació. Principis generals

Paradigmes de programació. Principis generals



En l'actualitat existeixen diferents estils que defineixen com abordar el disseny i l'escriptura d'un programa, la qual cosa es coneix com a *paradigmes de programació*. Cadascun posseeix característiques (regles, tècniques i principis) específiques que ho diferencia dels altres, fent-ho més o menys apropiat per resoldre diferents tipus de problemes en funció de les seves necessitats. Els paradigmes de programació més utilitzats són l'imperatiu, l'orientat a objectes i el funcional. És important conèixer els avantatges i els inconvenients de cadascun amb la finalitat de triar el més adequat per abordar un problema, ja que pot fer que el codi sigui més llegible, mantenible i escalable. En aquest sentit, els llenguatges de programació juguen també un paper significatiu, ja que estan dissenyats per a suportar un o més paradigmes de programació. Això significa que uns certs paradigmes poden ser més fàcils d'implementar o més efectius en uns certs llenguatges de programació que en uns altres. Per exemple, la programació orientada a objectes és àmpliament utilitzada en llenguatges com Java i C++, mentre que per a la programació funcional s'usen llenguatges com Haskell i Lisp. A més, alguns llenguatges de programació són específics d'un únic paradigma, com els llenguatges Pascal i C, que s'utilitzen principalment per a la programació imperativa. Altres llenguatges són multiparadigma, cosa que significa que permeten l'ús de diversos paradigmes diferents en el mateix programa. Per exemple, Python és un llenguatge de programació multiparadigma que admet programació imperativa, orientada a objectes i funcional. La capacitat d'un llenguatge de programació per a suportar múltiples paradigmes pot brindar als desenvolupadors la flexibilitat i el poder per a triar l'enfocament de programació més adequat per al seu projecte.

A continuació, es descriuen els tres paradigmes de programació més comuns, juntament amb el tipus d'aplicacions en el qual són més indicats:

• La **programació imperativa** es caracteritza per posar el focus en "com" es resol el problema, per la qual cosa els programes consisteixen en la descripció precisa i detallada de la seqüència de passos que cal realitzar per resoldre un problema. La idea és fer ús de les instruccions per anar modificant l'estat del programa. Per això, es compta amb les variables, que són els elements que permeten emmagatzemar les dades (tant d'entrada com de sortida), i amb els bucles i els condicionals, que controlen l'ordre d'execució de les instruccions. A més, els subprogrames (procediments o funcions) s'usen per a modularitzar i reutilitzar el codi. El gran avantatge d'aquest paradigma és que el model baix el que es basa és molt intuïtiu, ja que és molt semblant a un "manual d'instruccions pas a pas". Per això, és el que se sol utilitzar per ensenyar i aprendre a programar. No obstant això, a l'hora de resoldre problemes més complexos, el codi pot fer-se massa llarg, la qual cosa pot dificultar el seu manteniment.

• La **programació orientada a objectes (POO)** considera un programa com una col·lecció d'objectes que interactuen entre si, de manera més semblant a com passa a la vida real. Un objecte constitueix un exemplar d'una classe, que és la "plantilla" o model que defineix la representació, les propietats i el comportament comuns a un conjunt d'objectes. La representació de l'objecte es fa per mitjà de variables que es denominen atributs. Per exemple, per a definir les propietats comunes dels i les estudiants de l'assignatura de programació, com el seu nom, cognoms i qualificació alfanumèrica ("suspès", "aprovat", "notable" i "excel·lent") es podria definir una classe, anomenada Estudiant, amb tres variables (atributs) denominats Nom, Cognoms i Qualificació, respectivament

El comportament dels objectes el defineixen els mètodes, que són els procediments o funcions que implementen les operacions per manipular-los i per interactuar amb altres objectes. Per exemple, la classe Estudiant hauria de

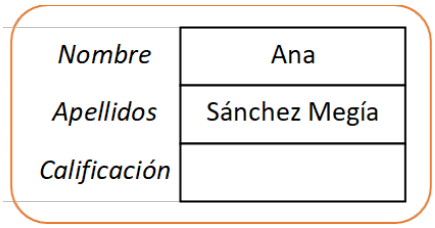


Figura 1. Exemple d'objecte de la classe *Estudiant.*

contenir, almenys, mètodes per conèixer el nom, els cognoms i la qualificació d'un o una estudiant. Existeix un mètode particular, anomenat constructor, que és el que s'invoca per a crear un objecte. Així, per crear l'objecte corresponent a l'alumna Ona Serra Vallespir podria invocar-se al constructor amb el nom i els cognoms com a paràmetres. El resultat implica la creació d'una "variable" similar a un registre amb tres camps com il·lustra la Figura 1. AnyRealització

Una vegada definida una classe, es poden crear tants objectes o instàncies com es necessitin.

La POO es fonamenta en els tres pilars següents:

- **Encapsulació** s'usa per ocultar la representació concreta dels objectes, la qual cosa els protegeix d'usos indeguts. Per exemple, que no es pugui assignar una qualificació inexistent o que dues qualificacions no es puguin sumar.
- **Herència** s'utilitza per crear subclasses, és a dir, classes derivades d'una altra, que hereten les seves propietats i mètodes, però que poden tenir altres addicionals. Per exemple, es podria crear la classe EstudiantRepetidor, com a subclasse d'Estudiant, per a representar els estudiants que ja van cursar un altre any l'assignatura, amb l'atribut específic AnyRealització, a més dels quals hereta (Nom, Cognoms, Qualificació). AñoRealización
- **Polimorfisme**, que permet que els objectes de diferents classes es comportin de manera similar en funció del context. Per exemple, els objectes de la classe EstudiantRepetidor també són objectes de la classe Estudiant, per la qual cosa poden comportar-se de totes dues formes. Així, poden formar part de la llista de tots els estudiants de l'assignatura, formada per objectes de la classe Estudiant.

El polimorfisme dota de gran flexibilitat als programes, la qual cosa constitueix un avantatge d'aquest paradigma; l'herència facilita la reutilització de codi, i l'encapsulament ajuda a prevenir errors. A més, el disseny de classes permet abordar el programa de manera modular, la qual cosa facilita, a més, inserció de nous objectes i la

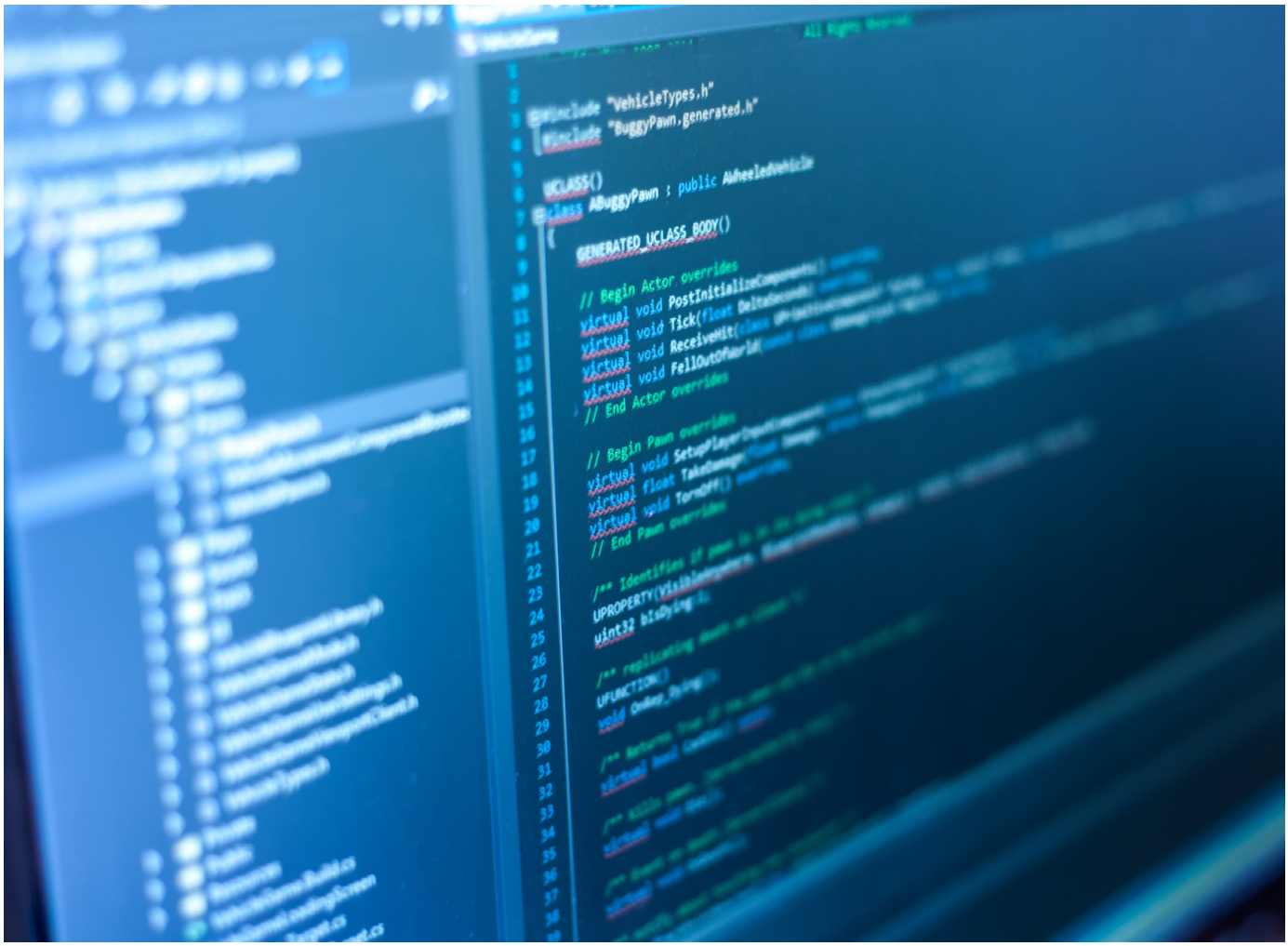
modificació dels existents. Per tant, és un paradigma idoni per al desenvolupament de grans aplicacions en equip. No obstant això, cal tenir en compte que l'execució dels programes és més lenta i que el disseny de classes pot ser massa complicat.

- **La programació funcional** s'enquadra en el paradigma de programació declarativa, mitjançant el qual els programes especifiquen què volen aconseguir sense descriure com fer-ho. En el paradigma funcional, els elements amb els quals s'escriuen els programes són les funcions pures, és a dir, aquelles que per a una mateixa entrada sempre produeixen el mateix resultat. Al contrari que en el paradigma imperatiu o en l'orientat a objectes, no interessa modificar l'estat del programa, per la qual cosa les dades romanen **immutables**: durant l'execució del programa les dades no es modifiquen, sinó que es van creant altres nous. Per això, és imprescindible que les funcions estiguin correctament parametritzades i que la invocació a les mateixes es faci amb els valors adequats. Cal tenir en compte que les funcions poden ser emprades també com a paràmetres d'una altra funció. A més, les funcions no produeixen efectes col·laterals indesitjats. D'altra banda, en lloc de bucles s'usa la **recursió**. La recursió ocorre quan una funció es diu a si mateixa, per la qual cosa es crea una repetició en el qual els valors antics romanen inalterables. Un exemple de funció recursiva pot ser la que calcula la suma des d'1 fins a n, sent n el seu argument, i que es podria definir així:

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

D'aquesta manera, `Suma(3)=Suma(2)+3;` però `Suma(2)=Suma(1)+2`. Como `Suma(1)=1`, ara es van "desfent" les crides, és a dir, ara ja es pot calcular `Suma(2)=1+2=3`; i després, `Suma(3)=3+3=6`. El valor obtingut és el resultat d'`1+2+3.`

Els avantatges de la programació funcional són vàries: el codi sol ser més concís i expressiu, ja que les funcions solen ser petites i independents entre si, la qual cosa facilita el manteniment dels programes. A més, la immutabilitat de les dades evita l'existència d'efectes col·laterals, per la qual cosa és més fàcil detectar errors i facilita la concurrència. Tenint en compte que la complexitat de les aplicacions és cada vegada major, especialment per la quantitat de dades que processen, cal desenvolupar programes que puguin ser executats en diverses màquines alhora, suportant concurrència i paral·lelisme. Per això, aquest paradigma està ressorgint amb força en el món empresarial i industrial. No obstant això, la programació funcional no és senzilla, ja que la recursivitat és una tècnica complexa, que pot donar a errors greus si no es domina. D'altra banda, el manteniment dels programes és complicat i el codi és difícilment reutilitzable.



Creació de
continguts digitals

Nivell C2 3.4 Programació

Depuració a Python. Aspectes generals

Depuració a Python. Aspectes generals

Introducció

En programació, la necessitat de la creació de codi que funcioni correctament i sense errors sempre és present com a objectiu. En aquest context, la depuració es converteix en un aspecte essencial del procés de programació. La depuració és un procediment sistemàtic que ajuda a detectar, aïllar i rectificar errors o "bugs" en un programa, assegurant així tant el seu correcte funcionament com la generació d'un codi lliure d'errors.

L'habilitat per depurar programes a Python, igual que en qualsevol altre llenguatge de programació, és una competència crucial per a qualsevol desenvolupador. Un codi correctament depurat minimitza l'existència d'errors sense detectar, prevé fallades o comportaments imprevistos del programa en la seva execució, i permet als desenvolupadors entendre millor la lògica i el flux del seu codi, cosa que facilita el manteniment del programari.

Entre les diverses eines disponibles per facilitar la depuració en Python, una de les més destacades és pdb. Pdb, acrònim de *Python DeBugger*, és el depurador integrat en el llenguatge Python. Aquest depurador brinda una sèrie de funcionalitats valuoses que permeten als programadors recórrer el seu codi pas a pas, inspeccionar variables i avaluar expressions.

Tipus d'errors

En depurar programes a Python, és probable que ens trobem amb diversos tipus derrors. Els errors a Python es classifiquen generalment en tres categories: errors de sintaxi, excepcions i errors lògics.

1 | **Errors de sintaxi**: aquests ocorren quan el codi viola les regles de sintaxi del llenguatge Python. Exemples d'aquesta mena d'error podria ser des d'oblidar posar dos punts (:) al final d'una declaració de funció o classe, fins a errors de sagnia o fins i tot ometre parèntesi o claus. Quan Python troba un error de sintaxi, interromp el procés d'interpretació

de codi i mostra un missatge d'error que indica la línia i la naturalesa de l'error.

2 | **Excepcions**: aquests són errors que ocorren durant l'execució del programa. Encara que la sintaxi del codi pot ser correcta, el programa pot generar un error quan intenta executar una instrucció. Les excepcions inclouen situacions com intentar dividir un número per zero, obrir un arxiu que no existeix, o accedir a una variable no definida, entre altres. Python és molt explícit quan es produeixen excepcions i proporciona un rastreig de pila detallat, incloent-hi la línia en la qual va ocórrer l'excepció i el tipus d'excepció.

⚠ ATENCIÓ

Un rastreig de pila a Python, també conegut com a '*stack trace*', és una representació de la seqüència de les crides de funcions que ha realitzat el teu programa fins a arribar a un punt determinat. És com una empremta digital de com el teu programa ha arribat on està, generalment presentat quan ocorre un error. Si el teu programa falla (un error), Python et mostrarà la 'ruta' que va seguir, indicant les funcions i mètodes que es van dir i en quina ordre, per ajudar-te a diagnosticar on podia haver sorgit el problema.

3 | **Errors lògics**: aquests són els errors més difícils de detectar i corregir. Els errors lògics succeeixen quan la lògica o el disseny d'un programa és incorrecte, però el programa s'executa sense generar errors de sintaxi o excepcions. Un exemple comú d'un error lògic és un bucle infinit. Encara que el bucle pot ser sintàcticament correcte, si la condició de terminació del bucle mai es compleix, el programa continuarà executant-se indefinidament. Aquests errors no són capturats pel sistema d'excepcions de Python, per la qual cosa requereixen una acurada revisió del codi i una comprensió sòlida de la lògica del programa per a la seva resolució.

Depurador pdb

El depurador pdb és una eina vital per al desenvolupament de codi a Python. Ofereix una gamma de funcionalitats que faciliten el procés de depuració, cosa que permet als desenvolupadors rastrejar el seu codi, establir punts de ruptura, avançar pas a pas, i molt més. Aquí, revisarem alguns dels conceptes clau per començar a depurar amb pdb.

1 | **Punts de ruptura:** els punts de ruptura són marcadors que pots establir en una línia de codi específica on vols que l'execució del programa es detingui. Això és útil quan saps que un error té lloc en una secció específica del codi, però no estàs segur exactament on o per què. Amb pdb, pots usar la funció "`set_traci()`" per establir un punt de ruptura.

2 | **Execució línia per línia:** una vegada que l'execució del programa s'ha detingut en un punt de ruptura, pots usar la funció "`step`" (o "`s`" en la seva forma abreujada) per avançar línia per línia en el codi

3 | **Inspecció de variables:** en aturar-se en un punt de ruptura o en avançar línia per línia, pots voler veure el valor d'una variable específica. Amb pdb, pots fer-ho simplement ingressant el nom de la variable en la consola de pdb. També pots usar la funció "`args`" per a veure els arguments de la funció actual.

4 | **Continuar la execució:** si has detingut l'execució en un punt de ruptura i has determinat que tot està funcionant correctament fins a aquest punt, pots usar la funció "*continue*" (o "c" en la seva forma abreujada) per a reprendre l'execució fins al pròxim punt de ruptura o fins al final del programa.

5 | **Sortir de pdb:** si necessites aturar la depuració per qualsevol motiu, pots usar la funció "*quit*" (o "q" en la seva forma abreujada) per sortir de pdb i acabar l'execució del programa.

Exemple de depuració d'un programa

Partim del següent codi d'exemple que divideix un número per un denominador i després va baixant el denominador fins que arriba a 0, la qual cosa ocasionarà una excepció del tipus ZeroDivisioError.

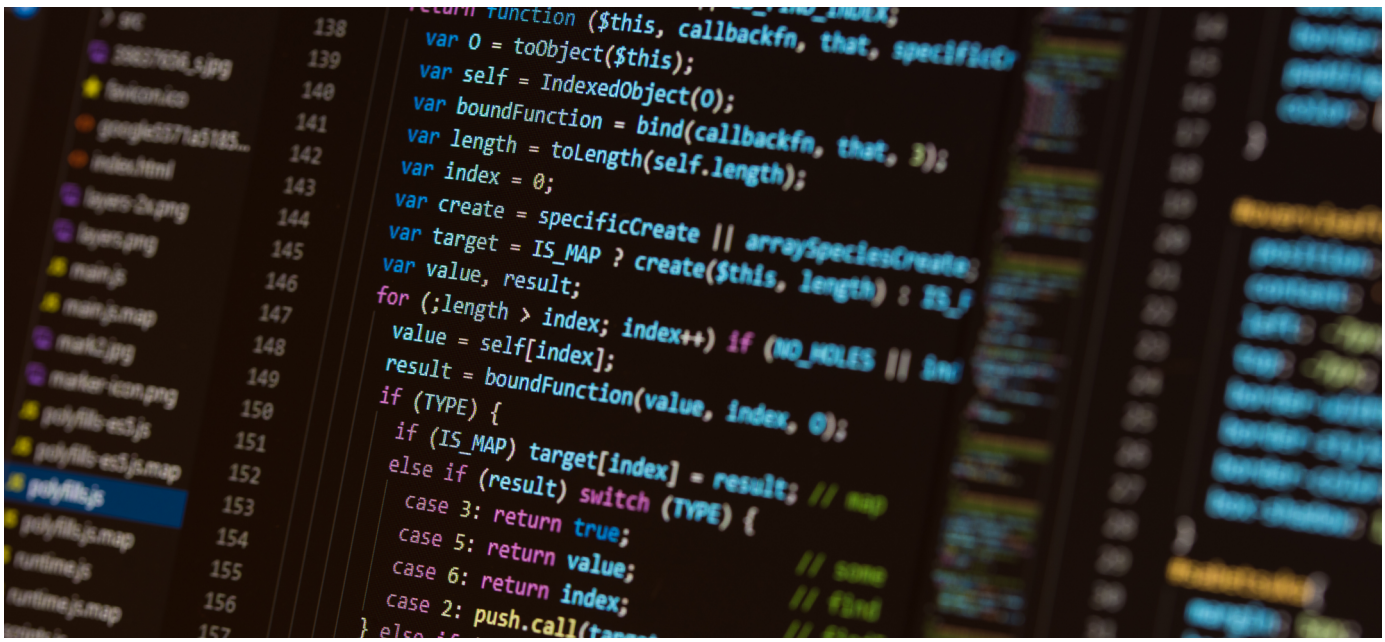
```
def divisio_decreixent(numerador, denominador):
    while denominador >= 0:
    resultat = numerador / denominador
        print(f"El resultat de dividir {numerador} entre {denominador} és
        {resultat}")
        denominador -= 1

divisio_decreixent(10, 3)
```

Per depurar el codi, inserirem un punt de ruptura en la funció abans de la línia que creiem que està causant el problema:

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

divisio_decreixent(10, 3)
```



Una possible simulació de la depuració en la consola de Python podria ser la següent:

```
> python3 el_meu_programa.py
> /ruta/a/el_meu_programa.py(6)divisio_decreixent()
-> resultat = numerador / denominador
(Pdb) p denominador
3
(Pdb) c
El resultat de dividir 10 entre 3 és 3.3333333333333335
> /ruta/a/el_meu_programa.py(6)divisio_decreixent()
-> resultat = numerador / denominador
(Pdb) p denominador
2
(Pdb) c
El resultat de dividir 10 entre 2 és 5.0
> /ruta/a/el_meu_programa.py(6)divisio_decreixent()
-> resultat = numerador / denominador
(Pdb) p denominador
1
(Pdb) c
El resultat de dividir 10 entre 1 és 10.0
> /ruta/a/el_meu_programa.py(6)divisio_decreixent()
-> resultat = numerador / denominador
(Pdb) p denominador
0
(Pdb) s
ZeroDivisioError: division by zero
```

En aquesta simulació, utilitzam el comandament "p" per a inspeccionar el valor del denominador en cada iteració del bucle. També utilitzem el comandament "c" per a continuar l'execució fins al pròxim punt de ruptura, que està dins del bucle, per la qual cosa s'atura en cada iteració. Finalment, quan veim que el denominador és 0, utilitzam el comandament "s" per a avançar a la línia següent, la qual cosa provoca el `ZeroDivisioError`. Mitjançant la depuració, el programador podria adonar-se de la línia en la qual s'està produint l'error i just la situació que el provoca (valors concrets de numerador i denominador).

Saber-ne més

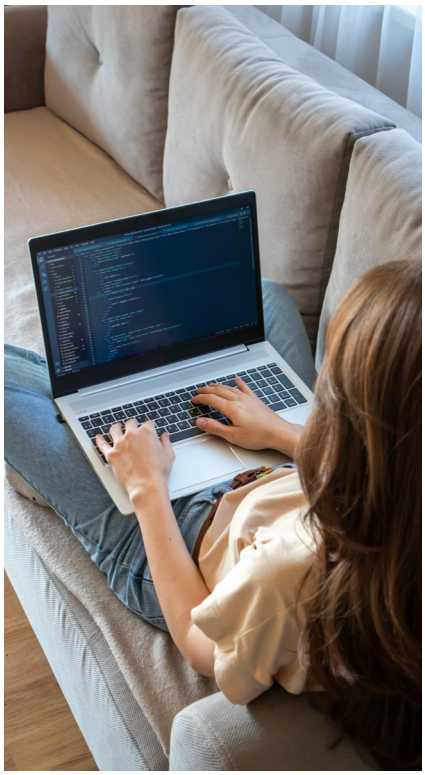
Pots aplicar els teus coneixements en depuració a Python amb pdb a través de la documentació oficial de Python, en concret, a *e.digitall.org.es/depuracion*

Creació de
continguts digitals

Nivell C2 3.4 Programació

Disseny de codi a Python. Bones pràctiques

Disseny de codi a Python. Bones pràctiques



Introducció

El disseny de codi efectiu i eficient és un dels principals reptes als quals s'enfronta qualsevol programador. En aquest sentit, Python, amb el seu enfocament en la simplicitat i llegibilitat, ofereix un mitjà ideal per a desenvolupar codi que no només sigui funcional, sinó que també sigui fàcil d'entendre i mantenir. L'adopció de bones pràctiques de disseny de codi és crucial, no només per millorar la qualitat del programari, sinó també per facilitar la col·laboració i el manteniment a llarg termini del codi. De fet, seguir bones pràctiques pot ser considerat una inversió a llarg termini. Pot requerir més temps i esforç inicialment, però definitivament val la pena a llarg termini, ja que minimitza els problemes i augmenta l'eficiència

L'aplicació de bones pràctiques redueix la complexitat del codi, augmenta el seu mantenibilitat i facilita la detecció i correcció d'errors i millora la qualitat del programari.

Bones pràctiques per al disseny de codi a Python

A continuació, es discuteixen algunes de les bones pràctiques més habituals que hauries de considerar en el disseny i codificació de programes.

Nom descriptiu

Els noms de les variables, funcions i classes han de ser prou descriptius per indicar-ne el propòsit o ús al programa. Els noms han de ser entenedors i proporcionar una idea del que fa la funció o del que representa la variable. Aquesta pràctica facilita la lectura i comprensió del codi, no només pel programador mateix, sinó també per altres desenvolupadors que puguin treballar o revisar el codi. Per exemple, suposem que escriviu una funció per calcular l'àrea d'un cercle. En lloc de nomenar la funció de manera genèrica com a `func1` i utilitzar `x` per a la variable del radi, pots utilitzar noms més descriptius:

```
#Nom poc descriptiu

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

```
#El mateix exemple amb nom de variables descriptiu

def calcular_area_cercle(radi):
   return 3.1416 * (radi**2)
```

Ús apropiat de comentaris

Els comentaris són una eina útil per explicar el propòsit o la funcionalitat d'una secció de codi, especialment si és complexa o no està clarament indicada pel codi en si. Haurien d'utilitzar-se per a agregar detalls que no són immediatament obvis només amb llegir el codi. No obstant això, també és important no sobrecarregar el codi amb comentaris innecessaris, ja que això pot fer que el codi sigui més difícil de llegir.

Exemple: Suposem que estàs implementant un algorisme que ordena una llista de números utilitzant el mètode de la bombolla. Aquí és com podries comentar correctament aquest codi:

```
def ordre_bombolla(llista):
   # Iterar sobre cada element a la llista
   for i in range(len(llista)):
      # Comparam l'element actual amb el següent
      for j in range(0, len(llista) - i - 1):
         # Si l'element actual és major que el següent,
         els intercanviam
         if llista[j] > llista[j + 1] :
            llista[j], llista[j + 1] = llista[j + 1], lista[j]
```

Aplicació del principi de responsabilitat única

Aquest principi suggereix que cada funció, mòdul o classe ha de tenir una única responsabilitat. En termes més senzills, haurien de fer només una cosa, però fer-la bé. El compliment d'aquest principi fa que el codi sigui més manejable, fàcil de mantenir i menys inclinat a errors, ja que, si sorgeix un problema, saps exactament on cercar.

Exemple: posem que tens una aplicació que processa comandes d'un comerç electrònic.

```
def manejar_comanda(client, producte, quantitat):
    # Verificar disponibilitat del producte
    ...
    # Actualitzar inventari
    ...
    # Processar el pagament
    ...
    # Generar factura
    ...
    # Enviar correu de confirmació al client
    ...
```

En lloc de tenir una funció que manegi tot el procés de comanda, seria millor tenir funcions separades per a cada pas del procés.

```
def verificar_disponibilitat(producte, quantitat):
    ...
def actualitzar_inventari(producte, quantitat):
    ...
def processar_pagament(client, monto):
    ...
def generar_factura(client, detalls_comanda):
    ...
def enviar_confirmacion_per_correu(client, detalls_comanda):
    ...
```

Ús de funcions i abstracció

L'abstracció és un mètode essencial en la programació que se centra en distanciar els detalls tècnics d'una part específica del codi de la seva aplicació pràctica. En el llenguatge Python, es poden emprar les funcions per simplificar operacions i englobar seccions de codi que compleixen amb un propòsit definit. L'ús de l'abstracció mitjançant funcions optimitza la llegibilitat del codi, la seva possibilitat de reutilització i facilita el seu manteniment.

Per exemple, si vols crear un programa que fa càlculs matemàtics complexos diverses vegades, en lloc d'escriure la mateixa lògica de càlcul una vegada i una altra, podries abstreure aquesta lògica en una funció. Llavors, en lloc de repetir el mateix codi, simplement cridaries a la funció cada vegada que necessitis realitzar aquest càlcul. Així, si necessites canviar la manera en què es realitza el càlcul, només hauries de fer-ho en un lloc (dins de la funció), en lloc de cercar i canviar múltiples instàncies del mateix codi.

NOTA

Quan s'utilitzen funcions a Python, un bon consell és seguir el principi que "menys és més". Les funcions haurien de ser petites i fer només una cosa. Si trobes que una funció creix massa o comença a fer massa coses, probablement és el moment de dividir-la en diverses funcions més petites

Maneig d'errors i excepcions

Un bon programa és aquell que és robust i pot manejar errors i situacions inesperades. El maneig d'errors i excepcions a Python permet als programes manejar errors i continuar executant-se, fins i tot si una cosa inesperada succeeix. Els programadors poden definir el que hauria d'esdevenir si hi ha una excepció a una mena d'error particular.

Exemple: Imagina que tens una funció que divideix dos números. En lloc de deixar que el programa falli si intentes dividir per zero, pots capturar i manejar aquesta situació de la manera següent:

```
def dividir(numerador, denominador):
    try:
        return numerador / denominador
    except ZeroDivisioError:
        print("Error: No es pot dividir per zero.")
        return None
```

En aquest exemple, si intentes dividir per zero, el programa captura l'excepció `ZeroDivisioError`, imprimeix un missatge d'error i retorna `None`. D'aquesta manera, el programa no s'atura i pot continuar executant-se, a pesar que es va intentar fer una operació invàlida. Això fa que el teu programa sigui més robust i amigable per a l'usuari.

Evitar codi redundant

Aquest és un principi de desenvolupament de programari fonamental que suggereix que qualsevol funcionalitat hauria de ser implementada en un sol lloc. Si trobes que estàs escrivint el mateix codi més d'una vegada, pot ser un signe que hauries d'encapsular aquesta funcionalitat en una funció o classe i reutilitzar-la.

Ús de les convencions d'estil de Python (PEP 8)

La guia d'estil *Python Enhancement Proposal 8*, comunament coneguda com a PEP 8, és un conjunt de recomanacions sobre com formatar el codi Python. Seguir aquestes convencions ajuda a mantenir la consistència i millora la llegibilitat del codi. Encara que algunes d'aquestes regles poden semblar arbitràries, adherir-se a elles pot fer que sigui més fàcil per a uns altres (i per a tu mateix) llegir i entendre el teu codi.

Pots accedir a la guia a: ***e.digitall.org.es/pep8***

Prova sempre que el teu codi funciona correctament

Les proves unitàries són un component essencial de la programació saludable. Permeten al programador verificar que les peces individuals de codi (o "unitats") estiguin funcionant correctament sota diferents circumstàncies i entrades. A Python, el mòdul **unittest** és una eina poderosa per dur a terme aquestes proves.

Escriure proves unitàries per al teu codi pot ajudar-te a identificar i corregir errors més de pressa, millorar la qualitat del codi, i també pot facilitar la modificació i extensió del codi en el futur.

Exemple: Suposa que tens la següent funció que calcula el factorial d'un número:

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

Podries escriure la següent prova unitària per assegurar-te que la funció funciona correctament:

```
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 classe `TestFactorial` conté un mètode anomenat `test_factorial` que fa diversos tests sobre la funció factorial. Es comprova que el resultat de la funció sigui correcte per a diversos valors d'entrada. Si algun d'aquests tests falla, la funció `unittest.main()` ens informarà sobre la fallada i podríem llavors corregir el problema en la funció factorial.

Ús de patrons de disseny

Els patrons de disseny són solucions provades a problemes comuns en el disseny de programari. Ofereixen un marc per enfrontar situacions comunes, cosa que permet una millor organització del codi i facilitant el seu manteniment i expansió en el futur. Utilitzar patrons de disseny pot ajudar-te a escriure codi més net, modular i eficient.

Saber-ne més

Per saber més sobre Patrons de disseny, es recomana la lectura de Gamma, E. (2002). Patrons de disseny. Espanya: Pearson Educación.

Documenta el codi

A més dels comentaris i els docstrings en el codi, també és important proporcionar una documentació més àmplia i completa. Aquesta pot incloure la documentació de l'API per a biblioteques o mòduls, els manuals de l'usuari per a programes, i les guies d'instal·lació o configuració per al programari. La documentació és crucial per al manteniment i l'escalabilitat d'un projecte, així com per a la col·laboració amb altres desenvolupadors.

Una bona documentació del codi pot ser en forma d'arxius README, wikis en el repositori de codi, o fins i tot pàgines web dedicades amb la documentació completa del projecte.

Saber-ne més

A més de les bones pràctiques existeixen moltes més. Per això, et recomanem la lectura dels següents llibres:

"**Clean Code: A Handbook of Agile Programari Craftsmanship**" per Robert C. Martin: Encara que aquest llibre no està escrit específicament per a Python, les lliçons i principis que ensenya són aplicables a qualsevol llenguatge de programació. El llibre tracta sobre com escriure codi d'alta qualitat que sigui fàcil de llegir, mantenir i estendre. T'ajudarà a entendre com els professionals pensen sobre el disseny del programari i per què fan les coses de la manera que les fan.

"**Fluent Python: Clear, Concise, and Effective Programming**" per Luciano Ramalho: Aquest llibre és un excel·lent recurs per a programadors intermedis i avançats de Python. No només cobreix les característiques del llenguatge, sinó que també ofereix perspectives sobre les "formes pythonicas" de fer les coses. A través d'aquest llibre, pots aprendre a escriure codi Python que és més idiomàtic, eficient i clar.

Creació de continguts digitals

Nivell C2 3.4 Programació

Maneig d'excepcions a Python

Maneig d'excepcions a Python

Excepcions a Python

El llenguatge de programació Python distingeix dos tipus d'errors principals: i) els errors de sintaxi i ii) les excepcions. Els primers tenen lloc a causa de construccions incorrectes a l'hora d'utilitzar la sintaxi del llenguatge. El següent llistat mostra un senzill exemple, on es pot apreciar com l'intèrpret de Python assenyala una posició pròxima al potencial error (en aquest cas, la falta del caràcter: després de 'while True'). Les segones, com ja s'ha introduït prèviament, responen a esdeveniments que tenen lloc durant l'execució d'un programa, alterant de manera no desitjada el seu flux d'execució estàndard.

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

Quan es produeix una excepció a Python, i no es controla, llavors l'intèrpret mostra el nom de l'excepció que s'ha llançat. Per exemple, el següent llistat mostra l'excepció ZeroDivisioError, definida pel mateix llenguatge quan s'intenta dividir per 0.

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

Afortunadament, és possible escriure programes a Python que manegin excepcions de manera adequada. Per això, és possible utilitzar la **sentència** try. Aquesta funciona de la manera següent:

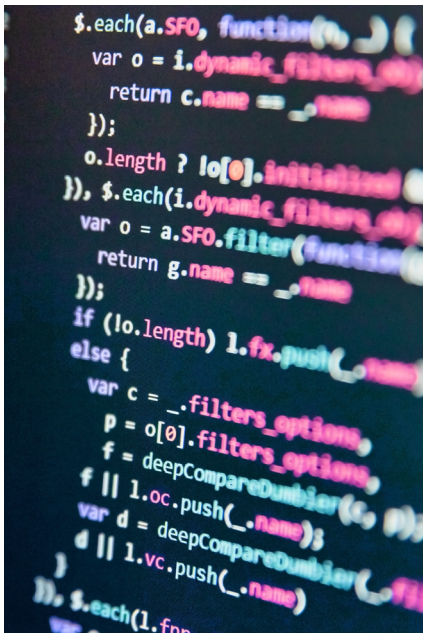
1 | En primer lloc, s'executa la clàusula try (les sentències entre les paraules clau try i except).
2 | Si no es produeix cap excepció, la clàusula except se salta i finalitza l'execució de la sentència try.
3 | Si es produeix una excepció durant l'execució de la clàusula try, evita la resta de la clàusula. Llavors, si el seu tipus coincideix amb l'excepció nomenada després de la paraula clau except, s'executa la clàusula *except* i l'execució continua després del bloc try/except.

4 | Si hi té lloc una excepció que no coincideix amb l'excepció nomenada en la clàusula `except`, es passa a les sentències try externes. Si no es troba un manejador per a l'excepció, llavors es tracta d'una excepció no manejada i l'execució s'atura amb un missatge com el del llistat anterior.

Una sentència `try` pot tenir més d'una clàusula *except*, amb l'objectiu d'especificar manejadors per a diferents excepcions. D'altra banda, la sentència `try` té una clàusula `else` opcional que, quan és present, ha de seguir a totes les clàusules `except`. Aquesta clàusula else és útil per incloure el codi que ha d'executar-se si la clàusula `try` no llança una excepció. El següent fragment de codi mostra un exemple en el qual es tracta d'obrir un arxiu, el nom del qual s'emmagatzema en el `meu_arxiu`. Si l'arxiu no existeix, llavors es llançaria una excepció del tipus OSError, capturada en la clàusula `except`. Si l'arxiu existeix, llavors, en aquest exemple, s'executaria el codi de la clàusula `else`, on s'imprimeix el nombre de línies que té l'arxiu.

```
try:
   f = open(el_meu_arxiu, "r")
except OSError:
   print("No es pot obrir", el_meu_arxiu)
else:
   print(el_meu_arxiu, "té", len(f.readlines()), "línies.")
   f.close()
```

És possible definir excepcions d'usuari, és a dir, excepcions la semàntica de les quals està associada al codi que representa la solució a un problema concret. Un exemple d'excepció definida per l'usuari, ja introduïda anteriorment, podria contemplar l'intent d'emmagatzematge d'un número de telèfon que no tingui exactament 9 dígits. El següent llistat mostra el codi representatiu d'aquest exemple, en el qual es pot apreciar la creació de l'excepció `FormatDeNumeroNoValidException`, definida com una nova classe a Python. Aquesta classe heretaria de la classe genèrica `Exception`.



```
class FormatDeNumeroNoValidException(Exception):
   pass

numero = input("Introduïu un número de telèfon de 9 dígits...")
if len(numero) != 9:
   raise FormatDeNumeroNoValidException
```

L'exemple anterior també mostra com utilitzar la sentència raise per llançar una excepció. En aquest cas, es podria suposar que la capa superior de codi inclouria la lògica necessària per capturar, i tractar, l'excepció llançada.

Per concloure aquesta secció, esmentarem la clàusula finally. En essència, si existeix una clàusula finally, aquesta s'executarà com a darrera tasca abans que finalitzi la sentència try. La clàusula finally s'executa tant si la sentència try produeix una excepció com si no. El següent exemple mostra un fragment de codi més complet.

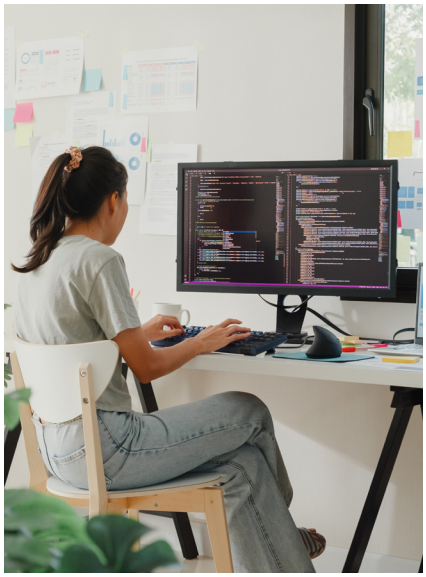
```
def dividir(a, b):
   try:
      resultat = a / b
   except ZeroDivisioError:
      print("Divisió per zero.")
   else:
      print("El resultat és", resultat)
   finally:
      print("Executant la clàusula finally")
```

Creació de
continguts digitals

Nivell C2 3.4 Programació

Proves de codi a Python

Proves de codi a Python



Introducció

Les proves són una part essencial en el procés de desenvolupament d'aplicacions. Gràcies a aquestes és possible verificar la correcta funcionalitat del codi i garantir la seva qualitat. Un clar exemple de la seva importància és el cas del Desenvolupament Guiat per les Proves (TDD).

En aquest document, s'explorarà com escriure proves unitàries, en concret per a aplicacions desenvolupades a Python. Abans de continuar, és aconsellable que revisis la documentació ***A3C34C1D06***.

PROVES DE CODI. ASPECTES FONAMENTALS

Document introductori a les proves de codi i l'enfocament TDD.

Document referenciat: ***A3C34C1D06***

Tipus de proves. Proves unitàries

Les proves utilitzades durant el desenvolupament d'aplicacions són de diferents tipus, l'ús d'unes o altres depèn del que s'hi vulgui verificar. Algunes de les més utilitzades són: les proves unitàries, proves d'integració, proves funcionals i proves de regressió. Aquest document se centra en les primeres, les proves unitàries.

Les proves unitàries són un tipus de proves focalitzades a verificar el correcte funcionament d'unitats individuals de codi. És a dir, assegurar que funcions, mètodes o classes funcionin correctament de manera aïllada.

A Python, existeixen diferents eines i entorns de treballs que faciliten la creació i execució de proves unitàries. Algunes de les llibreries més utilitzades a Python són unittest, pytest i nose.

Proves a Python: unittest

Unittest és el framework de proves integrat a la biblioteca estàndard de Python. Proporciona una sèrie de mòduls i eines que ajuden a comprovar les funcions del codi i facilitar el desenvolupament de les aplicacions.

Exemple pràctic

En aquesta secció es veurà un exemple pràctic de proves unitàries amb Python. En concret, s'ha desenvolupat una aplicació que funciona a tall de calculadora. Es tracta d'una classe en la qual s'han implementat els mètodes de suma, resta, multiplicació i divisió:

```
class Calculadora:
    def suma(self, a, b):
        return a + b
    def resta(self, a, b):
        return a - b
    def multiplicacio(self, a, b):
        return a * b
    def divisio (self, a, b):
        if b == 0:
            raise ValueError("Error: no es pot dividir entre zero")
        return a / b
```

Abans d'implementar les proves, cal analitzar els diferents escenaris a provar. Per exemple, en el cas de la calculadora serà necessari comprovar la suma de dos nombres positius, la resta de dos nombres negatius, la divisió entre zero, etc.

Una vegada estudiats els casos a implementar, es procedeix a la seva codificació. En aquest cas, amb unnitest, es poden seguir els següents passos:

- Crear una classe de proves anomenades `TestCalculadora` que hereta de `unittest.TestCase`.
- Importar el mòdul a provar (Calculadora).
- Crear el mètode `setUp` per començar una instància de la calculadora abans de cada prova.
- Definir mètodes de prova que cobreixin diferents possibilitats d'execució. En cada mètode s'empraran assercions com "`assertEquals`", "`assertNoEquals`", "`assertTrue`" o "`assertFalse`" per comprovar la funcionalitat.
- El nom dels mètodes ha de començar amb "test_".

Aquest codi mostra un exemple de proves unitàries per cobrir la classe 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 aquest cas s'ha implementat una prova unitària per cada mètode de la classe Calculadora. No obstant això, el correcte seria cobrir els diferents escenaris d'execució que pot tenir-ne de cadascun.

En resum, en aquest document s'ha abordat la importància de les proves en el desenvolupament d'aplicacions. En particular, s'ha vist un exemple pràctic de creació de proves unitàries amb l'entorn de treball de la llibreria estàndard de Python, unittest. És important tenir en compte que el món de les proves és ampli i complex, i les proves unitàries són només una part del panorama complet de les proves de programari.

NOTA

Per a agilitzar la creació i comprovació de proves unitàries, és recomanable utilitzar alguns dels IDE que donen suport a Python. Com que en temes anteriors es va veure l'entorn de Visual Studio Code, és interessant ampliar els coneixements utilitzant les llibreries de proves de Python que poden ser utilitzades des de l'IDE.

Saber-ne més

A continuació, es llisten alguns dos dels principals entorns de treball de proves per a aplicacions desenvolupades a Python.

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

DigitAll

Formació en
Competències
Digitals

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