DigitAll

Gaitasun
digitaletan
prestakuntza

3

Eduki digitalak sortzea

DigitAll

Gaitasun
digitaletan
prestakuntza

C1 maila

AURKIBIDEA

DigitAll

Eduki digitalak sortzea

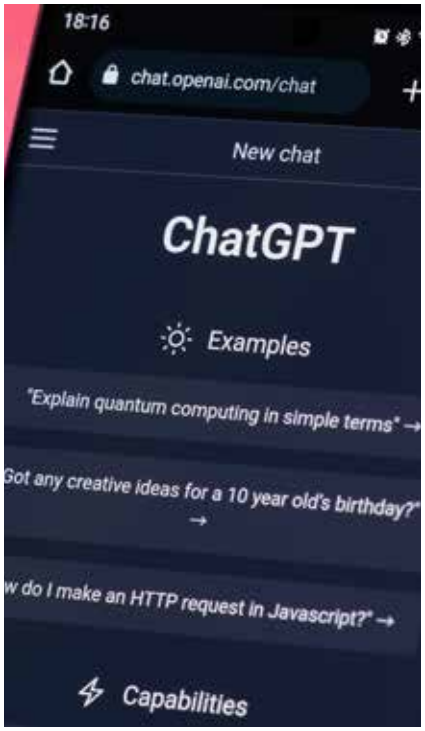
3.1 EDUKIAK GARATZEA

Eduki digitalak sortzea

C2 maila 3.1 Edukiak garatzea

GPT bidez testua automatikoki sortzea

GPT bidez testua automatikoki sortzea



Sarrera

GPT bidez testua sortzea oso tresna baliagarria da eduki digitalak sortzeko, eta onura ugari ditu zer eremutan aplikatzen den. Horrela, GPT gai da askotariko gaietan testu koherente eta sinesgarria sortzeko, eta horrek denbora eta ahalegina aurreztea ahalbidetuko digu, ez dugulako hitz bakoitza hutsetik idatzi beharko. Gainera, teknologia horrek ideia berriak eta sortzaileak esploratzeko aukera ematen die erabiltzaileei, ikuspegi eta perspektiba desberdinak iradokitzen baititu. Azkenik, maila honetan eta aurrekoetan landu ditugun gai askotarako erabil daiteke, hala nola gure webgunean sarrerak sortzeko.

GPTk edukia pertsonalizatzea errazten du, eta proiektu bakoitzaren behar eta lehentasun espezifikoetara egokitzen da. Laburbilduz, testua GPT bidez sortzea tresna ahaltsua da, eta eduki digitalen sorrera optimizatzen du denbora aurrezten duelako, kalitatea hobetzen duelako eta sormena sustatzen duelako.

Ohikoa da GPTren honelako deskribapenak aurkitzea: «elkarrizketak izateko trebatuta dagoen adimen artifiziala», eta hori egia den arren, pertzepzio hori baliteke oztopoa izatea sistema horren bidez informazioa lortzeko orduan. Arretaz pentsatzen badugu, begien bistakoa da: galdera beraren erantzuna aldatu egingo da testuinguruaren arabera: nork egiten digun, zein une eta tokitan, eta abar.

Beraz, oso garrantzitsua izango da kontuan hartzea ChatGPTri edo testua sortzeko beste eredu batzuei zer informazio ematen diogun, horrek definituko baitu sistemak erantzunak ematen dizkigun testuingurua.
Testu honetan, ChatGPTri zer informazio eman behar diogun azalduko dugu, gure behar espezifikoetara egokitzen den testua lortzeko.

Zer informazio eman behar diogu sistemari?

Definitu behar ditugun atal orokor batzuk deskribatuko ditugu, zeinak egokituko baitira testua sortzeko egiten ditugun eskaera gehienetara.

Helburua eta gaia

Lehenik eta behin, begien bistakoena: testuak jorratzea nahi dugun gai nagusia zehaztu behar dugu, bai eta zer helbururekin behar dugun ere: informatzailea, limurtzailea, deskribatzailea, dibulgaziozkoa, etab.

Egitura

Adierazi behar dugu ea gure testua sortzean sistemak jarraitzea nahi dugun egitura espezifikorik dagoen. Horrek lagunduko dio sistemari modu eraginkorragoan antolatzen emango digun informazioa. Adibidez, ipuin baten eskema klasikoa eska dezakegu: sarrera, korapiloa eta amaiera. Baina gure gustuko atalak sortzeko argibideak ere eman ditzakegu; adibidez, gai bati buruzko adibide praktikoak dituen atal espezifiko bat.

Informazio garrantzitsua

Testuan jaso beharreko xehetasunak eta datuak eman behar ditugu. Eska dezakegu, adibidez, datu estatistikoak, kasu praktikoak, gertakari historikoak eta abar barne hartzeko.

Estiloa eta tonua

Gure *prompt*ean, adierazi behar dugu gure testuak zer estilo eta tonu izatea nahi dugun. Adibidez, formala, informala, teknikoa, limurtzailea, neutroa, etab. Gainera, adierazi behar dugu ea lehentasunik dugun esaldien luzerari eta hiztegiaren erabilerari dagokienez. Adibidez, esaldi laburrak eta hiztegi teknikoa.

Xede Ikus-entzuleak

Norentzat da gure testua? Horrek hizkuntza eta ikuspegia egokitzen lagunduko du, hartzaileentzat egokia eta eraginkorra izan daitezen. Adibidez, haurrentzako edo helduentzako ipuina idazteko eska dezakegu.

Eskakizun gehigarriak

Berariazko betekizunak badaude, hala nola erreferentziak gehitzea (adibidez, liburu eta webgune fidagarriak), gure eskaeran ere aipa dezakegu hizkuntza inklusiboa erabiltzea eta zer testu-luzera nahi dugun.

Berrikuspena eta doikuntzak

Funtsezko urratsa da, eta askotan ahazten da horrelako tresnak erabiltzean; izan ere, prozesua ez da amaitzen sistemak gure *prompt*ean oinarritutako testu bat sortzen digunean. Beraz, urrats hori gure testua jaso ondoren egingo dugu: berrikusi egingo dugu, eta, gure itxaropenak betetzen ez dituen zatirik badago, beste eskaera bat egingo dugu, behar diren aldaketak eginda, sistema gure lehentasunetara hobeto egokitu dadin. Adibidez, paragrafo jakin bat berriz idazteko eska dezakegu, adibideak barne hartuta, ulergarriagoa izan dadin.

Ondorioa

Beraz, hizkuntza-eredua denez, ChatGPT dauden patroi eta datuetan oinarritzen da erantzun koherenteak sortzeko. Hala ere, informazio gehigarririk gabe, baliteke ereduak guztiz ez ulertzea nahi dugun testuaren testuingurua eta zehaztapen zehatzak. Deskribapen xehatua emanez gero, testu-sorkuntza automatikoaren balioa eta erabilgarritasuna maximizatuko ditugu, eta gure beharretara egokitutako testua lortuko dugu.

Informazio gehiago

«Sparks of Artificial General Intelligence: Early experits with GPT-4» (arXiv:2303.12712) ikerketa sakona dago GPT-4 nola entrenatu zen eta haren aurrekoarekin (GPT-3) nola alderatu zen jakin dezagun. GPT-4ren balizko aplikazioak eta mugak aztertu arren, testuaren interesgarriena da askotariko *prompt*ak saiatu izana eta sistemak itzultzen dituen emaitzak.

Eduki digitalak sortzea

C2 maila 3.1 Edukiak garatzea

Areak hautatzeko, ebakitzeko eta definitzeko bideen erabilera aurreratua

Areak hautatzeko, ebakitzeko eta definitzeko bideen erabilera aurreratua

Sarrera

Irudiak editatzeko eta manipulatzeko dauden tresna aurreratu ugarien artean, bideek funtsezko zeregina betetzen dute irudietako areak edo eremuak hautatzean, ebakitzean eta zehazki definitzean. Testu honetan, bideen erabilera aurreratuan jarriko dugu arreta, zereginotan emaitza zehatzak lortzeko, aurreko mailan eskuratutako ezagutzak osatze aldera. GIMP programarekin jarraituko dugu lanean, adibide gisa.

Kontrolatu zure bideak bide-panelaren bidez

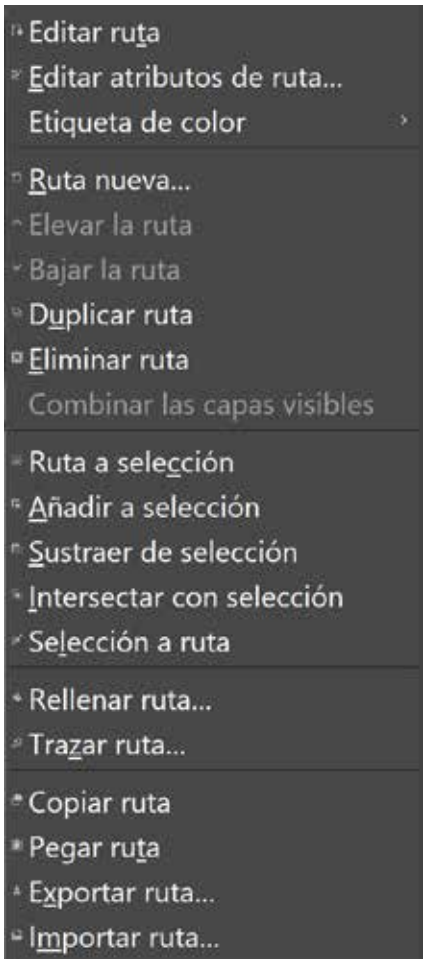
Ikusi dugunez, oinarrizko hautespen-tresnek ez bezala, bideek aukera ematen dute inguru konplexu eta kurba leunak zehaztasun handiagoz delineatzeko. Hala, irudi batean, objektu baten inguruko bide bat marratu dezakegu, eta, ondoren, bide hori hautespen aktibo bihurtu, eta area edo eremu zehatz horretan zenbait efektu edo aldaketa espezifiko aplikatu.

Gure bideak kudea ditzakegu bide-panelaren bidez, tresna horrek aukera ematen baitigu sortutako bideak kudeatzeko eta erabiltzeko. Panel horrek proiektu batean dauden bide guztien ikuspegi zehatza ematen du, eta aukera ematen du bideak editatzeko, antolatzeko, erakusteko (edo ez) eta, azken batean, eraginkortasunez erabiltzeko. 1. irudian ikus dezakegun bezala, geruza-panelaren ondoan agertzen da, eta antzeko funtzionamendua du.

Adibide praktikoa

Bideek ezin konta ahala aukera ematen dituzte diseinuaren arloan, gure konposiziorako zer azken helburu ezarri dugun, bideak modu batera edo beste batera erabiliko ditugu. Kontzeptu oinarrizkoenak ezagutzen ditugu dagoeneko, eta orain urratsez urrats deskribatuko dugu diseinu bat nola egin bideak erabiliz. Urrats horiek GIMP plataforman aplikatzearen azken emaitza 2. irudian dago ikusgai.

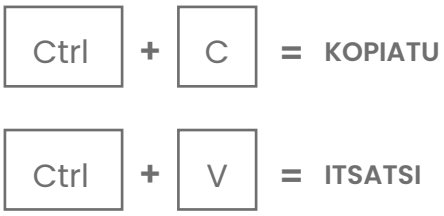
1. irudia (A). Bide baten ikuspegi orokorra bide-panelean.



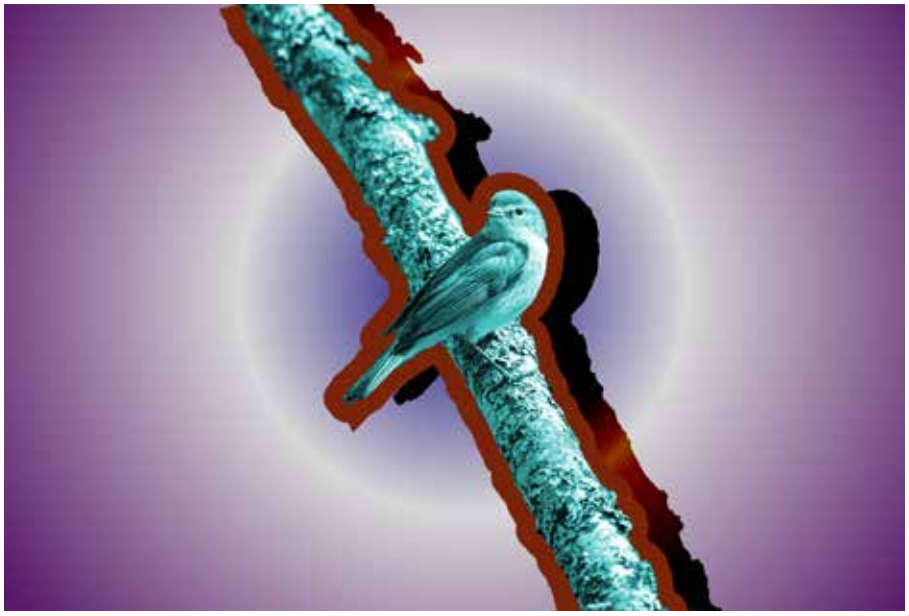
1. irudia (B). Menu zabalgarria, bide baten gainean saguaren bigarren botoia sakatzean irekitzen da.

- Jatorrizko irudia (1920×1293) hemen deskargatuko dugu: ***e.digitall.org.es/practica-rutas***
- Irudia GIMP-n irekiko dugu.
- «Bide-tresna» aukeratuko dugu, eta bide bat marraztuko dugu adarraren eta txoriaren inguruan.
- Behar adina aldiz editatuko dugu emaitza ona erdietsi arte.
- Hautespen bat sortuko dugu bide-tresnaren menuko «Sortu hautespena bide batean oinarrituta» aukeran klikatuz.
- Hautespena kopiatzeko, sakatu ***CONTROL + C***.
- Irudi berri bat sortuko dugu (Fitxategia > Berria), gure konposizio berria sortzeko.
- Gure irudia itsatsiko dugu ***CONTROL + V*** sakatuz. Geruzen menuan «Hautespen flotatzailea» gisa agertuko da. Beraz, bigarren botoia sakatu eta «Beste geruza batera» hautatuko dugu menuan.
- Jatorrizko irudira itzuliko gara, eta bide-panelean «Kopiatu bidea» sakatuko dugu.
- Gure irudi berriko bide-panelera joango gara, eta «Itsatsi bidea» sakatuko dugu.
- Baditugu jada gure konposizio berrian interesatzen zitzaigun gure irudiaren zatia eta irudia ebakitzeko sortutako bidea.
- Aukera bat da gure irudiaren gainean silueta bat sortzea, gure bidea erreferentzia gisa erabiliz; horretarako, geruza garden berri bat sortuko dugu (gure geruzaren eta hondoaren artean).
- Bide-panelera joango gara, gure bidea hautatuko dugu eta «Margotu bidean zehar» sakatuko dugu (behean, eskuinean).
- Agertzen zaigun aukera-panelean, Marratu lerroa > Kolore solidoa (*anti-alias* hautatuta egon behar da) aukeratuko dugu, eta aplikatu nahi dizkiogun marra-aukerak hautatuko ditugu.
- «Trazatu» sakatzean, egin dugun bidearen formako ezaugarriak dituen marra bat agertuko zaigu gure geruza berrian.
- Bigarren efektu bat gehituko dugu, eta bertan kolorez betetako silueta bat sortuko dugu gure ibilbidearen formarekin. Horretarako, gure bidea bikoiztu eta geruza berri bat sortuko dugu (lehendik genituen geruzen eta hondoaren artean).
- Orain, bide-tresnara joango gara, eta gure bidea «Mugitu» nahi dugula esango diogu, eta nahi dugun norabidean mugituko dugu pixka bat.

JATORRIZKO IRUDIA ADIBIDE PRAKTIKOA
e.digitall.org.es/practica-rutas



- Orain adieraziko dugu «Bidea bete» nahi dugula «Kolore solidoa» erabiliz, eta, berriz ere, betegarri-kolorearen ezaugarriak adieraziko ditugu (*anti-alias* hautatuta egon behar da). Gure ibilbidea adierazi diogun koloreaz bete duela ikusiko dugu.
- Azkenik, ezagutzen ditugun efektuak gehi ditzakegu konposizioa hobetzeko:
  - Adibidez, hondoko geruzan kolorea gehi dezakegu «Degradatzeko tresna» erabiliz.
  - Ebakitako gure irudiaren koloreak alda ditzakegu «Koloreak» paneletik.
  - Edo sortu dugun betegarri-geruzaren modua alda dezakegu.



2. irudia. Atal honetan deskribatutako jarraibideak urratsez urrats aplikatzearen emaitza «Adibide praktikoa».

Informazio gehiago

Bideak marrazteko tresnek zehaztasun- eta malgutasun-maila aurreratua ematen dute irudietan eremuak hautatu, ebaki eta definitzeko orduan. Diseinu-programa guztietan antzeko tresnak aurkituko ditugu; Photoshop programan ez dira bideak, trazatuak baizik. Esteka honen bidez ikusi ahal izango dugu tresna horiek nola antolatzen diren diseinu-programa ezagunean.

e.digitall.org.es/rutas-photoshop

Eduki digitalak sortzea

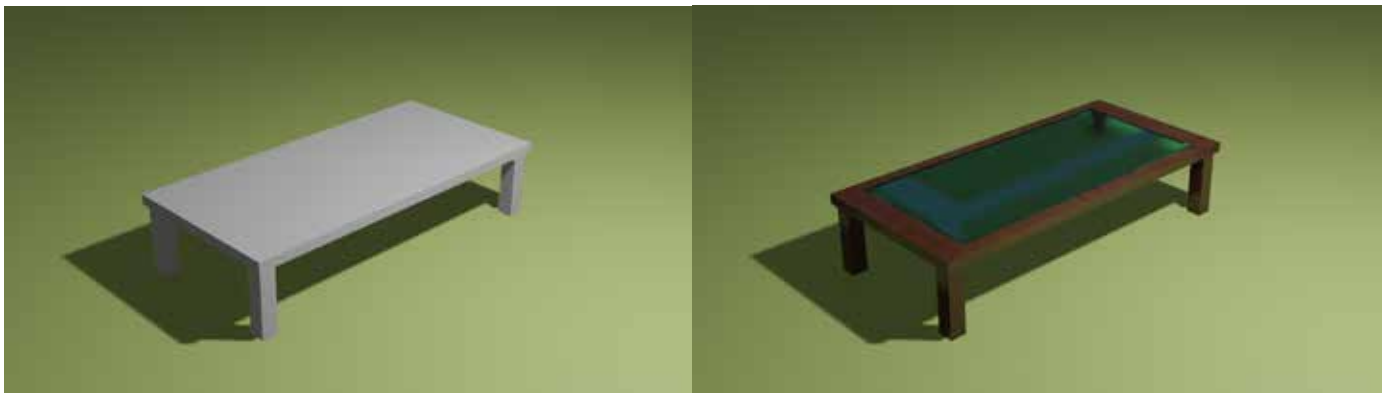
C2 maila 3.1 Edukiak garatzea

Materialen aplikazioa eta testurizatzea

Materialen aplikazioa eta testurizatzea

Sarrera

Informatika grafikoaren munduan objektu baten geometria definitzea lehen urratsa baino ez da 3D eredu errealistak eta bisualki erakargarriak sortzeko. Eredua diseinatu eta modelatu ondoren, ezinbestekoa da material egokiak esleitzea eta, kasu batzuetan, testurak aplikatzea, itxura errealista eta koherentea lortzeko. Esleitutako materialak definituko du objektua eta argia elkarreraginean nola aritzen diren, eta haren gainazalean argi-izpiak nola islatzen, errefraktatzen edo xurgatzen diren. Gainera, UV Mapping testurizatzea funtsezko teknika da 3D objektu baten gainazalean irudiak edo txantiloiak zehaztasunez eta errealismoz aplikatzeko. Dokumentu honetan, aztertuko dugu zer garrantzi duen, 3D grafikoak sortzean, materialak eta testurak esleitzeak.



Ezkerrean mahai bat dugu, testura lehenetsirik esleitu gabe sortua; eskuineko mahaiari, berriz, testura batzuk aplikatu zaizkio. Ohartu argiak haietako bakoitzean nola jokatzen duen, eta itxura errealistagoa nola lortzen duen.

Materialak esleitzea eta argiaren portaera

Materialak esleitzea

3D objektu batean materialak esleitzea funtsezko prozesua da argiarekin elkarreraginean nola aritzen den simulatzeko. Material bakoitzak propietate espezifikoak ditu, eta zehaztu egiten dute argiak haren gainazalean nola eragiten duen eta nola islatzen den. Material batzuk opakuak dira, eta argia norabide bakarrean islatzen dute; beste batzuk, berriz, gardenak dira, edo argia errefraktatzen dute haietatik igarotzen denean. Materialaren atributuen konfigurazioak —hala nola zimurtasunak, islapenak eta errefrakzio-indizeak— eragina izango du objektuaren azken itxuran eta eszenako errealismoan.

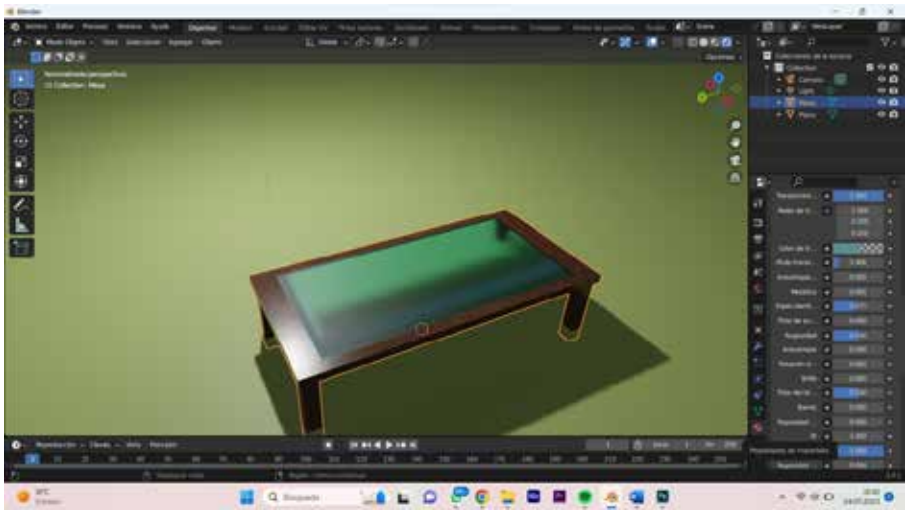
3Dn modelatu eta renderizatzeko programa askotan, materialak Phong argiztapen-eredua edo Lambert-en itzaldura-eredua erabiliz definitzen dira. Eredu horiek aukera ematen dute argia eta objektuaren gainazalak elkarreraginean nola aritzen diren simulatzeko, bai eta itzaldura- eta distira-efektua nola sortzen den simulatzeko ere, kontuan hartuta argi-iturriaren posizioa eta kameraren ikuspuntua zein diren.

Argiaren portaera, propietateak

Materialek definituko dute objektua eta argia elkarreraginean nola aritzen diren, eta haren gainazalean argi-izpiak nola islatzen, errefraktatzen edo xurgatzen diren. Material bakoitzak propietate espezifikoak ditu, eta zehaztu egiten dute argiak objektuarekiko nola jokatuko duen. Material bati esleitzen zaizkion aldiro definitzen dira propietateok. Hona hemen propietate garrantzitsuenetako batzuk:

- **Kolorea:** kolorea material baten atributurik oinarrizkoena da, eta objektua argi zuriaren eraginpean nola ikusiko den definitzen du. Kolore solido bat edo kolore-testura bat aukera dezakezu emaitza espezifiko bat lortzeko.
- **Distira (espekularitatea):** atributu honek erreflexuen kantitatea eta tamaina kontrolatzen ditu objektuaren gainazalean. Distira-balio altu batek erreflexu biziak sortuko ditu, eta balio baxu batek, berriz, itxura mateagoa emango du.
- **Zimurtasuna (*roughness*):** materialaren gainazalaren leuntasun- edo laztasun-maila nolakoa den definitzen du. Azalera oso zimurtsuak argia sakabanatuko du, eta itxura lausoa emango du; gainazal leunak, berriz, argia norabide zehatzagoan islatuko du.
- **Gardentasuna:** atributu honek materialaren gardentasun-maila kontrolatzen du. Objektua erabat gardena edo erdi-gardena izatea eragin dezakezu, beira edo beste material garden batzuk simulatzeko.
- **Errefrakzio-indizea**: errefrakzio-indizeak zehazten du argia nola tolesten den material gardena zeharkatzean. Hori bereziki garrantzitsua da zenbait materiala simulatzeko; adibidez, beira eta ura simulatzeko.

- **Testurak:** aurrez aipatutako atributuez gain, testurak aplika ditzakezu objektuaren gainazalari xehetasun konplexuagoak eta errealistagoak gehitzeko. Testurek barne har ditzakete txantiloi edo patroiak, irudiak eta ereduan gehitu nahi duzun beste edozein diseinu mota.



Blender softwarean propietate horiek defini ditzakegu objektu bat hautatuz eta interfazearen beheko eskuinaldeko material-leihoan sakatuz. Screenshot honetan, kristala simulatzeko sortutako materialerako erabilitako propietateak ikusten dira.

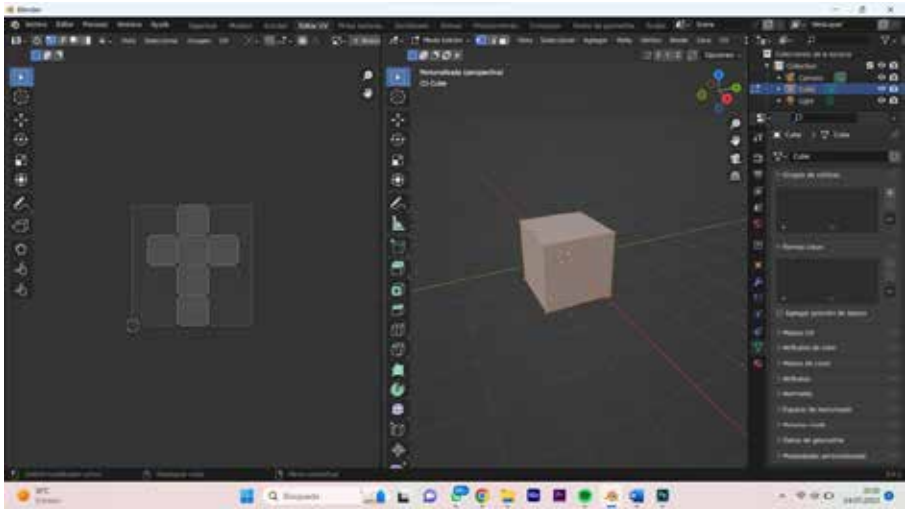
Garrantzitsua da kontuan hartzea materialak esleitzeak ez diola soilik objektuaren itxurari eragiten renderizazio estatiko batean, eta eragina ere baduela objektua askotariko argiztapen-kondizioetan eta animazio baten mugimenduan ikusteko moduan.
3D modelatu eta renderizatzeko programa askotan, hala nola Blender, Maya, 3ds Max eta beste batzuetan, 3D modeloei materialak esleitzeko interfaze intuitiboa aurkituko duzu. Programa horiek aurrez definitutako materialen liburutegiak eskaintzen dituzte, eta zure materialak sortzeko eta pertsonalizatzeko aukera ere ematen dute, arestian aipatutako ezaugarriak doituz, nahi dugun emaitza lortzeko.

UV Mapping testurizatze-teknika

Erraza da objektu bati testura bat esleitzea. Testura bat gehitu diezaiokegu zuzenean objektu bati (har dezagun testuratzat objektuaren «azala» irudikatzen duen irudia), eta 3D diseinu-programak berak banatuko du automatikoki eta uniformeki objektuaren gainazal osoan zehar.

Arazoa da banaketa automatiko horrek batzuetan ez duela ahalbidetzen testura behar bezala doitzea. Beste teknika aurreratuago batzuek aukera ematen dute eskuz eta zehaztasun handiagoz doitzeko; adibidez, UV Mapping testurizatze-teknikak.

UV Mapping testurizatzea deritzon teknikak aukera ematen du 3D eredu baten gainazalean testurak eta irudiak aplikatzeko. Horretarako, 2D koordenatuak mapeatu behar dira 3D objektuaren gainazalean (UV koordenatuak). Teknika hau bereziki erabilgarria da objektuaren itxura errealismo handiagoz zehaztu nahi dugunean; hau da, patroiak, testurak eta irudi konplexuak gehitu nahi baditugu.



Blender programan UV mapping egiteko ikuspegi propioa dugu, eta bertan lan egiten ari garen objektua «zabaldu» egiten digu.

UV Mapping testurizatze-prozesua hasiko da 3D ereduaren aurpegiak 2D plano batean zabaltzen, testurak zehaztasunez aplika daitezen. Horretarako, tresna espezifikoak erabil daitezke 3D modelatze-programetan, bai eta eskuz ere, kontrol zehatzagoa behar izanez gero.

UV koordenatuak behar bezala esleitu ondoren, nahi den testura aplika daiteke irudiak editatzeko programa batean. Gero, modelatzeko edo renderizatzeko softwarean 3D objektuaren materialari esleitzen zaio testura, eta bertan zenbait parametro doitzen dira, hala nola tamaina, intentsitatea eta testuraren errepikapena, nahi den emaitza lortzeko.

Blender programan hau baino ez genuke egingo: UV editting atalean objektua zabaldu, irudi-testura bat aplikatu (nahi dugun testura-mapa png edo jpeg formatuetan irudi gisa hautatu) eta UV editting atalera itzuli irudia gure objektura doitzeko. Garrantzitsua da zera azpimarratzea: gure irudiak zenbat eta bereizmen handiagoa izan, orduan eta hobeto ikusiko da gure renderra.

Hori esatea errazagoa da egitea baino, horregatik, saiatuko gara prozesua ahalik eta errazen egiten, jarraitu beharreko zenbait jarraibide eta horretarako beharrezkoak diren menuak gehituz. Hasi eta objektua zabaltzeko, «**UV editting**» ikuspegira joko dugu, eta objektua zabalduta agertuko zaigu bertan. Hurrenik, «**Itzaldura**» atalera joango gara. Ikuspegi horretan irudia erantsi beharko dugu (UV maparen formatu tipikoari jarraitu behar dio, aski da Googlen zer nahi dugun bilatzea, eta jarraian besterik ez duela egin beharko, eta, ondoren, «**Texture map**»). Horretarako, irudia arrastatu egingo dugu dagoen karpetatik zuzenean itzaldura-ikuspegiaren beheko aldera, «BSDF principista» izeneko koadroaren ondora. «Eskema» horretan irudia gehitu ondoren, gure irudiaren «Kolorea» puntua eta «BSDF principista» ataleko «Oinarrizko kolorea» puntua lotu beharko ditugu, eta, ondorioz, irudi hori nahi dugun objektuari esleituko genioke; orain birkokatu baino ez dugu egin behar, behar bezala ikus dadin.

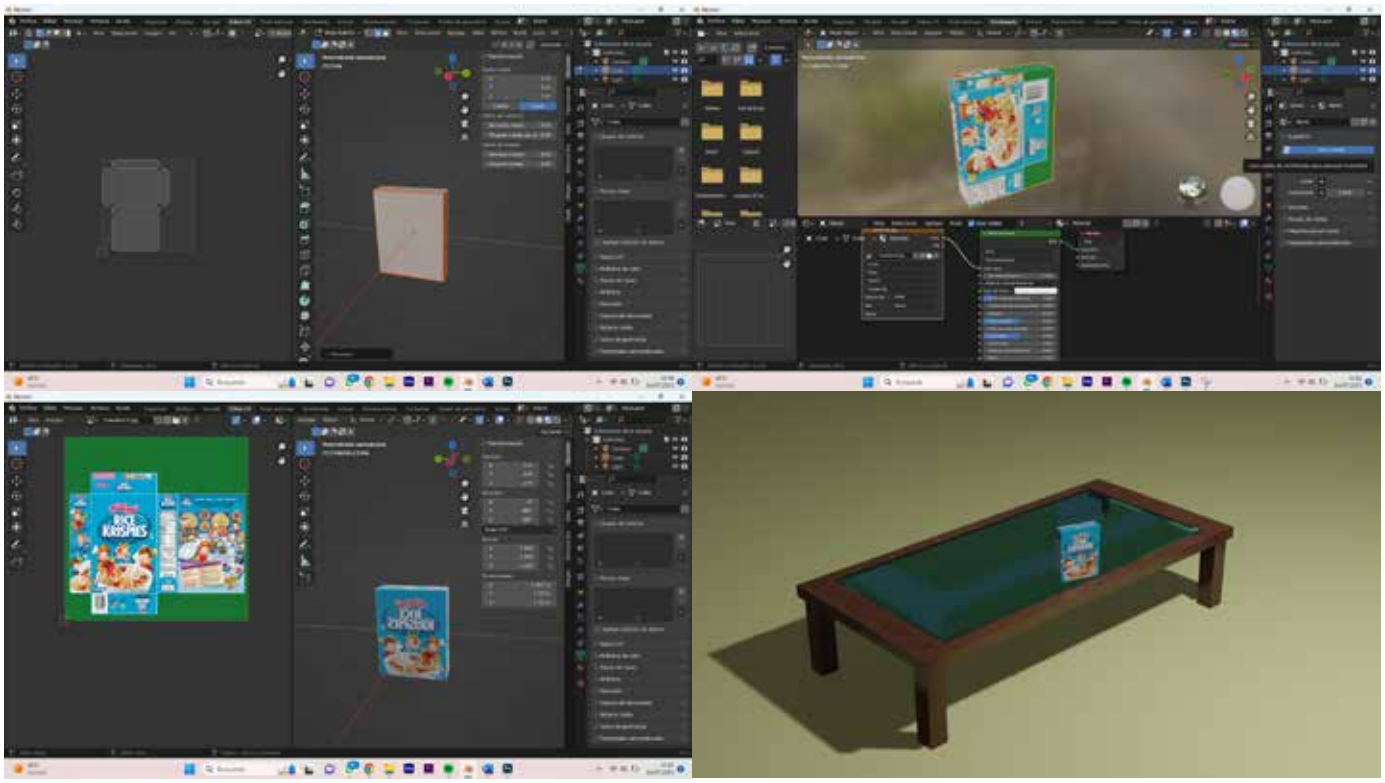
Horretarako, «**UV edditing**» ikuspegira itzuliko gara. Kasu honetan, irudi zabaldu bera ikusiko dugu orain, inportatu dugun irudiaren ondoan. Irudia nahi dugun emaitzarekin bat etor dadin, irudi zabaldua dagoen lekuan klik egin eta **a tekla** sakatu behar dugu. Horrela, irudi zabaldu osoa aukeratuko dugu; orain, hura mugitu behar dugu (**g tekla**rekin) edo biratu (**r tekla** sakatuz eta, zehaztasun handiagoa nahi badugu, zenbat gradu biratu nahi dugun idatziz), era horretan irudia zehazki bat etor dadin gehitu dugun testura-mapako irudiarekin.

Gainera, erabiltzen dugun irudiak ez du zertan testura-mapa bat izan; nahi dugun irudia gehi dezakegu eta nahierara egokitu. Hala ere, testura-mapak berariaz pentsatuta daude irudi jakin batzuei forma emateko, eta oso baliagarriak izan zaizkigu.

A = AUKERATU IRUDI ZABALDUA

G = MUGITU IRUDIA

R = BIRATU IRUDIA



Erlojuaren orratzen hurrenkeran, lehenengo irudian kubo bat dugu, kutxa modura eskalatua, zeina UV editting atalean zabaldu baita. Bigarren irudian ikusten dugu kuboari testura bat aplikatu zaiola, ausazko zereal-kaxa baten testura-maparen irudiarekin. Hirugarren irudian ikusten da testura-mapa kuboaren uv forma zabalduari nola doitu zaion. Laugarren irudian, azkenik, errepresentazio batean ikusten da kutxa nola geldituko litzatekeen aurrez egindako mahaiaren gainean.

Ondorioa

Laburbilduz, materialak esleitzea eta egiturak aplikatzea alderdi kritikoak dira 3D eredu errealistak eta bisualki erakargarriak sortzeko orduan. Materialak esleitzean definitzen dugu argia eta objektuaren gainazalera nola arituko diren elkarreraginean, eta horrek eragina du haren itxuran eta eszenako portaera bisualean. Bestalde, UV Mapping testurizatzea teknika baliagarria da ereduaren gainazalean irudiak eta patroiak zehaztasunez eta errealismoz aplikatzeko.

Funtsezkoa da teknika horiek ondo jakitea eta objektuaren itxurari nola eragiten dioten ulertzea kalitate handiko emaitzak lortuko baditugu askotariko proiektuetan (animazioa, bideojokoak, diseinu arkitektonikoa eta informatika grafikoaren bestelako arloak). Materialen eta egituren konbinazio egokiak ematen duen xehetasun- eta errealismo-mailak aldea ezartzen du 3D eredu arrunt baten eta sorkuntza bisualki harrigarri baten artean. Horregatik, hain zuzen ere, izango da beharrezkoa ordu asko ematea teknika horiek ulertzen eta erabiltzen.

Eduki digitalak sortzea

C2 maila 3.1 Edukiak garatzea

2D irudiak eta bideoa sortzeko renderizatzea

2D irudiak eta bideoa sortzeko renderizatzea

Sarrera

Animazioaren eta 3D grafikoen munduan, renderizazioa prozesu erabakigarria da 3D eredu bat irudi eta bideo bihur dadin. Askotariko aukerak daude renderizazioa egiteko, bakoitzak bere rendering-motorrekin eta irteerako konfigurazioekin. Jarraian, renderizaziorako aukera ezagunenetako batzuk aztertuko ditugu, baita kalitate handiko emaitzak lortzeko erabilgarri dauden konfigurazioak eta aukerak ere.

Rendering-motorrak

Kasu honetan ez dugu Blender bakarrik aztertuko; izan ere, haren sarrerako bideoan («***3D diseinu-tresna baten interfazeari buruzko sarrera orokorra***») ikusi genuen zer rendering-motor erabiltzen zituen. Hortaz, rendering-motorren ibilbide globala egingo dugu:

- **Cycles (Blender):** Cycles da Blender programaren renderizatze-motorra, eta asko erabiltzen da, malgua eta errealista delako. Izpien trazaduran oinarritutako ikuspegia ematen du, eta horrek argi- eta itzal-efektuak zehaztasunez simulatzea ahalbidetzen du; emaitza fotorrealistak ematen ditu. Cycles ezin hobea da kalitate handiko irudiak eta animazio errealistak sortzeko.
- **Arnold (Autodesk Maya):** Arnold zinemaren eta animazioaren industrian asko baloratzen eta erabiltzen duten renderizatze-motorra da. Ezaguna da eszena eta efektu bisual konplexuak erabiltzeko gaitasunagatik. Arnodl-ek errendimendu ezin hobea ematen du argiztapena eta itzaldura kalkulatzean, eta, horregatik, aukera ahaltsua da gama altuko proiektuetarako.
- **RenderMan (Pixar):** Pixarrek garatutako renderizazio-motor bat da, eta asko erabiltzen da film animatuak sortzeko. Ezaguna da, ezohiko xehetasun eta konplexutasuneko irudiak sortzeko gaitasuna duelako. RenderMan-ek konfigurazio eta aukera aurreratu ugari ematen ditu, azken emaitzaren gaineko kontrol zehatza izateko.

3D DISEINU-TRESNA BATEN INTERFAZEARI BURUZKO SARRERA OROKORRA

e.digitall.org.es/A3C31C2V04

Konfigurazioa eta irteera-aukerak

- **Bereizmena:** amaierako irudiko pixel kopurua zehazten du. Bereizmen handiagoak irudi zehatzagoak ekarriko ditu, baina renderizatze-denbora ere handituko da.
- **Fotogramaren tamaina:** 3D eszenako bistaratze-eremuaren tamaina definitzen du. Pantaila panoramiko, karratu edo pertsonalizatuko formatuan renderizatu dezakezu.
- **Laginketa eta anti-aliasing:** konfiguraziook ertzen kalitatea kontrolatzen dute; eskailera-efektuak (*aliasing*) ezabatzen dituzte objektu-ertzetan, eta irudi leunagoak ematen dituzte.
- **Kolore-sakonera:** pixel bakoitzeko kolore-informazio kantitatea definitzen du. Kolore-sakonera handiagoek (16 edo 32 bit, adibidez) kolore-tarte zabalagoa ahalbidetzen dute, eta itzal nahiz argietan xehetasunak galtzea eragozten dute.
- **Irteera-formatua:** irteerako fitxategi-formatua hauta dezakezu, hala nola JPEG, PNG, TIFF eta EXR, besteak beste. Aukeratutako formatua zure beharren araberakoa izango da, bai eta koloretako eta xehetasunetako informazioa ahalik eta hoberen gorde nahi duzun ere.
- **Konpresioa:** fitxategi-formatu batzuek konprimitzea ahalbidetzen dute, sortzen den fitxategiaren tamaina murrizteko. Hala ere, kontuan izan balitekeela konpresioak irudiaren kalitatean eragina izatea.

Bideoa sortzea

Fotograma bakar baten ordez bideoko animazioa sortzeko, sekuentzian renderizatutako irudi ugari konbinatu behar dira. Hori lortzeko, gako-fotogramak edo *keyframe*ak renderizatzeko prozesua erabiltzen da. Hona hemen bideo bat sortzeko urrats orokorrak:

- **Animazioa eta keyframeak:** zure 3D eszenako animazioa definitzen du, eta objektu, kamera eta argiztapenerako gako-posizioak (*keyframe*ak) ezartzen ditu. *Keyframe*ek adierazten dute eszenak denboran zehar zer bilakaera izan duen.

- **Bideo-sekuentziadorea edo bideo-editorea:** 3D modelatu eta renderizatzeko programa askok, hala nola Blenderrek, barneko bideo-sekuentziadorea dute edo bideo-editore batekin integratzen dira. Hemen, renderizatutako irudiak kargatu ditzakezu eta sekuentzia batean antolatu, bideoa sortzeko.
- **Framerate-tasa:** fotograma bakoitzaren iraupena eta bideorako erreprodukzio-abiadura definitzen ditu. Erreprodukzio-abiadura irudi kopurua segundoko (fps) neurtzen da, normalean 24 fps animazio arin baterako.

Orain, animazioari eta rendering-motorrei buruz ditugun ezagutza orokorrak konbinatuta, edozein irudi sortzeko gai izango gara. Horrela, haien edozein sekuentzia sortzeko gai izango gara, edo beste modu batera esanda, bideo bat, lehen aztertu ditugun motorrek aukera ematen baitigute *Keyframe*en arteko irudi-sekuentzia bat sortzeko, aldez aurretik ezarritako *framerate* bati egokituta.

Ondorioa

Laburbilduz, renderizazioa funtsezko prozesua da 3D irudi eta animazio errealistak eta bisualki erakargarriak sortzeko orduan. Rendering-motorra hautatzea eta irteera-konfigurazioak erabakigarriak dira kalitate handiko emaitzak lortzeko. Gainera, bideoa sortzeko, askotariko fotograma renderizatu konbinatu behar dira sekuentzia animatu batean, animazio harrigarriak eta arinak lortzeko. Aukera eta teknika horiek ondo erabiltzen jakinez gero, goi-mailako animazio-proiektuak eta 3D grafikoak sortu ahal izango dituzu.

Informazio gehiago

Interesgarria da Disney eta Pixar enpresa ospetsuetako animatzaileek, besteak beste, gaur egun erabiltzen dituzten eskuliburuak kontsultatzea, dokumentu honetan idatzi dugun guztia jasotzen baitute, askoz ere garatuagoa bada ere.

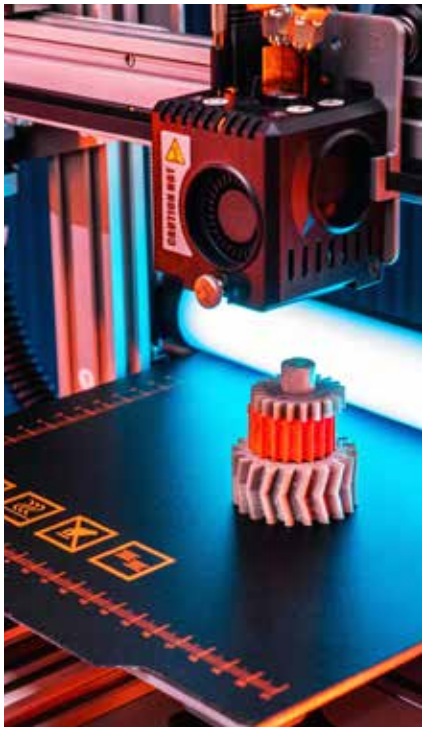
e.digitall.org.es/pixar

Eduki digitalak sortzea

C2 maila 3.1 Edukiak garatzea

3D objektuak fabrikatzea

3D objektuak fabrikatzea



Sarrera

3D inprimatzea

3D inprimatzearen historia

3D inprimaketak iraultza ekarri du objektuak fabrikatzeko modura, eta aukera eman du prototipoak, tresnak eta pieza pertsonalizatuak azkar eta eraginkortasunez sortzeko. 3D inprimatze-prozesuaren funtsezko zati bat zera da: 3D ereduak formatu egokietan esportatzea, ondoren inprimatzeko. Lehen orri honetan, 3D inprimagailuek onartutako esportazio-formatuak aztertuko ditugu, bai eta inprimatzeko erabiltzen diren aplikazioak eta tresnak ere.

3D inprimaketa gaur egungo zerbait dela pentsatu ohi dugu, baina, egiaz, 1980ko hamarkadan hasi zen 3D inprimaketa, orduan hasi baitzen forma hartzen fabrikazio gehigarriaren kontzeptua. 1984an, Chuck Hull ingeniari estatubatuarrak estereolitografia-prozesua (SLA) asmatu zuen, eta horixe hartzen da 3D inprimaketako lehen metodotzat. SLAk hiru dimentsioko objektuak geruzaz geruza sortzea ahalbidetzen zuen, argi ultramorearekiko sentikorrak ziren fotopolimeroak erabiliz. Urteek aurrera egin ahala, fabrikazio gehigarriko beste teknika batzuk garatu eta merkaturatu dira, hala nola material-jalkitze bidezko fusioa (FDM) eta laser bidezko sinterizazio selektiboa (SLS).

Nola inprimatzen dugu 3Dn?

3Dn zerbait inprimatu nahi dugunean, bi gauza beharko ditugu nagusiki. Lehenik eta behin, hardware-euskarria izango da, hau da, 3D inprimagailu bat. Bestalde, inprimaketarekiko 3D pieza bateragarri bat beharko dugu.
Hardwareari dagokionez, garrantzitsua da jakitea batez ere bi inprimagailu mota daudela: erretxina bidezko inprimagailuak eta filamentu bidezkoak.

- **Filamentu bidezko 3D inprimaketa** —FDM (Fused Deposition Modeling) edo FFF (Fused Filament Fabrication)— da 3D inprimaketa-metodo ohikoenetako eta eskuragarrienetako bat. Prozesu horretan, filamentu termoplastiko bat, normalean PLA edo ABS, pita baten bidez berotu eta estruitzen da. Urtutako materiala geruzaz geruza jartzen da 3D objektua eraikitzeko. Eredua eraikitze-plataforma baten gainean inprimatzen da; plataforma X, Y eta Z ardatzetan mugitzen da. Geruza bat jarri ondoren, plataforma jaitsi egiten da, eta hurrengo geruzaren inprimaketa hasten da.
- **Erretxina bidezko 3D inprimaketa** —SLA (Stereolithography) edo DLP (Digital Light Processing)—: erretxina likido fotosentikorrean oinarritutako 3D inprimaketa-teknologia bat da. Prozesu horretan, laser edo DLP proiektore batek modu selektiboan azaltzen du erretxina likidoa, eta geruzaz geruza solidotzea dakar. Filamentu bidezko inprimaketan ez bezala (azken honetan material urtua jalkiz eraikitzen dira objektuak), erretxina bidezko inprimaketan objektuak sortzeko, plataforma bat erretxinan murgiltzen da, eta modu kontrolatuan igotzen da geruzak solidotu ahala.

Laburbilduz, filamentuzko inprimagailu bat erabiliko dugu xehetasun-maila handia nahi ez badugu; aitzitik, pieza handia eta sendoa nahi badugu, eta gainazaleko akabera-maila handia, edo pieza baten xehetasun txikiak nabarmendu nahi baditugu, erretxina aukeratu beharko dugu. Nabarmentzekoa da, halaber, filamentua dela aukerarik zabalduena, erabiltzen erraza baita.

Dokumentu honetan bigarren aukera aztertuko dugu batez ere, 3D inprimagailuen mundua oso zabala baita; hala bada, 3D editoreei buruz aurretik hartutako prestakuntza zabaldu eta aplikatuko dugu guk, kasu honetan modelatzen ikasi dugunari forma fisikoa emateko. Zenbait jarraibide eta formatu ikasiko ditugu gure 3D objektua gure mundura ekar dezagun imajinatu dugunaren antzeko objektu fisiko baten modura.

Prestatzeko eta inprimatzeko prozesua

3D eredua inprimaketarako prestatzea

3D inprimaketak eredu bat inprimagailuan kargatzea baino gehiago dakar. Puntu honetan, honako alderdi hauetan sakonduko ditugu: eredua prestatzeko prozesua, beharrezko diren inprimaketa-doikuntzak eta inprimaketa bera gauzatzeko urratsak.
Inprimatu aurretik, 3D eredua prestatu behar da fabrikazio gehigarrirako prest dagoela ziurtatzeko. Horrek esan nahi du eredua berrikusi behar dela, akats geometrikoak antzeman eta zuzendu behar direla, eta doiketak egin behar direla inprimatzean objektuaren egonkortasuna bermatzeko. Euskarri-egiturak gehituko dira, orobat, hala behar duten diseinuetarako. Inprimaketaren aurreko aldaketa horiek bi fasetan banatzen dira: editorearen fasea eta inprimaketa-softwarearen fasea.

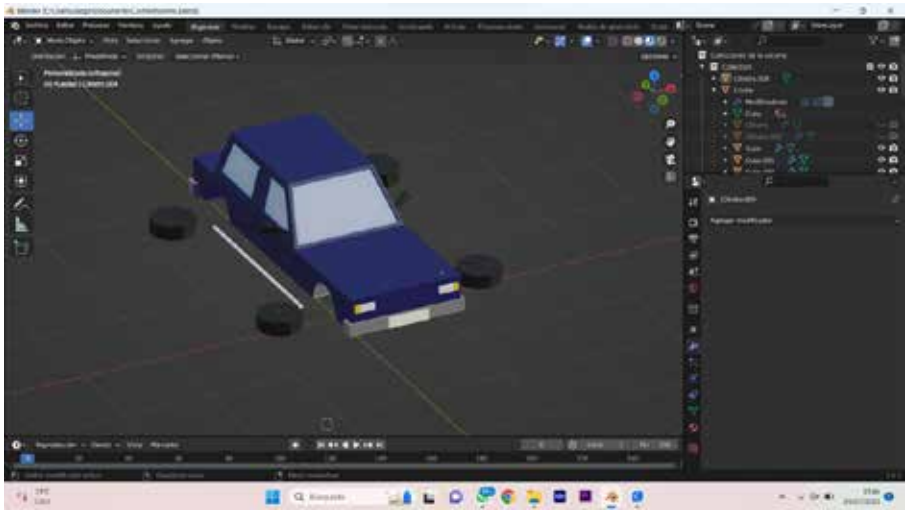
Edizio-fasea

Editore baten barruan (Blender gure kasuan), gauza asko hartu behar ditugu kontuan. Hona hemen jarraibide batzuk:

- **Benetako dimentsio eta eskaletara egokitzen garela egiaztatu behar dugu.** Inprimatu nahi dugun pieza bat diseinatzeko orduan, oso kontuan izan behar dugu gure inprimagailuaren tamaina, pasa ez gaitezen, betiere eredua zatika zatitu eta faseetan inprimatu nahi ez badugu, baina 3D inprimatze-softwarean ikusiko dugun bezala, hori gerora zuzentzeko aukera ere izango dugu.
- **Ahalik eta gehien optimizatu behar dugu erpin eta poligono kopurua.** Garbitu eta optimizatu zure ereduaren geometria. Ezabatu alferrikako geometria eta saihestu triangeluak eta ngonak (lau alde baino gehiagoko poligonoak), balitekeelako inprimatzean arazoak sortzea.
- **Orientazioa egiaztatu behar dugu.** Ziurtatu ereduaren orientazioa inprimatzeko egokia dela. Ereduaren oinarriak laua izan behar du, eta inprimatze-gainazalarekin kontaktuan. Saihestu irtengune txikiegiak edo egitura oso meheak, balitekeelako inprimatzen zailak izatea.
- **Ez dugu elementu flotatzailerik jarri behar.** Zure modeloak pieza mugikorrak edo elementu bereiziak baditu, ziurtatu behar bezala konektatuta daudela, ondoren inprimatze- eta mihiztatze-arazorik egon ez dadin.

- **Ez dugu aurpegi laurik erabili behar.** Mundu errealean ez dago objektu erabat laurik, eta hori kontuan hartu behar dugu, 3D inprimagailuek objektu solidoak inprimatzen baitituzte; hortaz, ziurtatu zure ereduak lodiera egokia duela alde guztietan. Zure eredua azalera bat baino ez bada, bihurtu ezazu 3D objektu solido Blenderreko Solidify aldatzailearen bidez.
- **Ez ditugu piezak gainjarri behar.** Ziurtatu ereduaren zatiak ez direla gainjartzen, gerta bailiteke horrek inprimaketa-arazoak sortzea eta piezak fusionatu edo distortsionatzea.

Kontuan hartu beharreko beste aholku bat zera da: fitxategiak STL eta OBJ formatu estandarretan esportatu behar ditugu beti, 3D inprimaketako software guztiekin ondo lan egiten baitute.

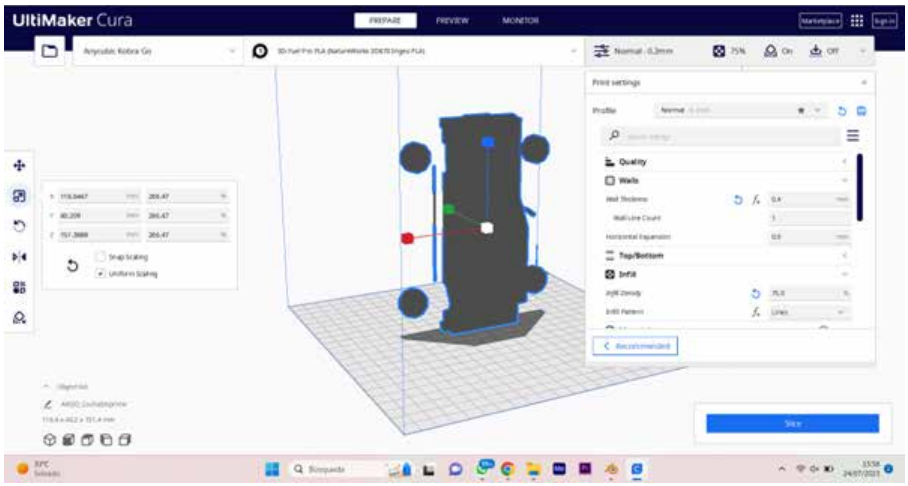


Screenshot honetan, Blender programan inprimatzeko prest dagoen eredu bat ikusten da. Pieza guztiak lurzoruaren mailan daude; hau da, ez dago pieza flotatzailerik, eta gainera, pieza gehiago lortu ditugu inprimaketa berean. Gainera, renderizazio-ikuspegia aktibatuta edukitzea komeni da; izan ere, kolorea inprimatuko ez bada ere, interesgarria da hura edukitzea gida gisa erabil dezagun.

Inprimaketa-softwarearen fasea

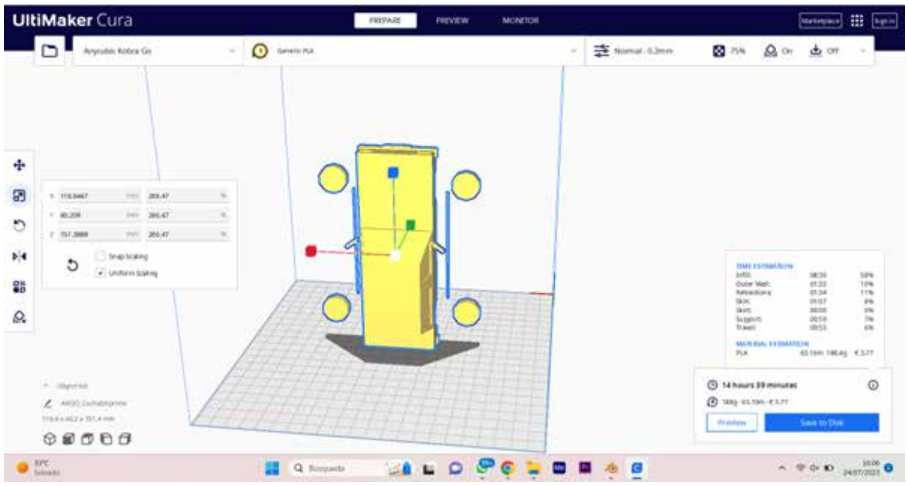
CURA hartuko software aurredefinitutzat; 3D inprimagailuetarako diseinatutako doako aplikazio bat da. Bertan, inprimatze-parametroak alda daitezke eta, ondoren, G kode bihurtu; azken hori behar da inprimatu beharreko fitxategietan.

CURA irekitzean, lehenik eta behin, dagokion inprimagailuaren profila hautatu behar da, eta 3D eredua STL edo OBJ formatuan kargatu. Gero, joan «Prestatu» fitxara (*Prepare*, ingelesez). Horren ondoren, parametro asko aldatzeko aukera izango dugu, hala nola inprimatzeko abiadura, lodiera eta ohearen tenperatura; oraingoz, lehenetsi bezala utziko ditugu, inprimatu nahi dugun lehen pieza guztietarako balioko baitigute horrela.



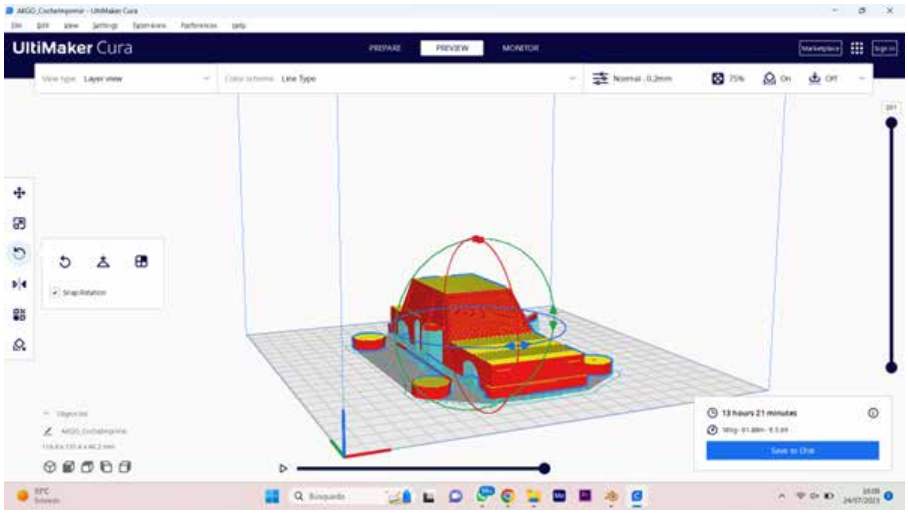
Hori izango da eredua inportatu ondoren izango dugun CURA programaren lehen inpresioa. Ikusiko dugu beste edozein 3D editoretan dauden tresna berak ditugula eredua mugitzeko. Eskuinean ere ikusiko ditugu aurreko paragrafoan esandako aukerak.

Halaber, zure ereduak hala behar badu, euskarri-egiturak gehitzeko aukera ere baduzu «Euskarria» (*Support*, ingelesez) fitxan; hau da, piezaren bat gorputz nagusitik urrun utzi dugulako edo zati flotatzailerik ez uzteko jarraibideari kasurik egin ez diogulako. Aztertu geruzen ikuspegia, dena behar bezala dagoela ziurtatzeko. Horren ondoren, denboraren estimazio bat agertuko zaigu, baita erabiliko dugun filamentua edo erretxina doitu dugun eta haren prezioa ere. Bi aukera horietako bat zuzena ez bada, parametroak alda ditzakegu, aukera horiek jaisteko (denbora eta prezioa). Azkenik, gorde G-code fitxategia. Fitxategi horretan daude zure inprimagailurako instrukzio espezifikoak, eta horixe erabiliko duzu inprimatzeko.



Orain, *slice*-rekin orain ikusiko dugu (egiazki) denbora eta prezioa kalkulatzen dituela (baldin eta lehentasunetan zehaztuta badugu). Erraz kalkulatzeko aukera izango dugu, halaber, inprimaketa-prozesu bakoitzak zenbat denbora iraungo duen.

Hala ere, nahi duzun pieza inprimatu aurretik, bertsio txikiago bat inprima dezakezu, eta akatsik duen egiaztatuko duzu horrela, bertsio osoa inprimatu aurretik zuzentzeko.



Aurreko ereduarekin Preview atalean akatsik zegoen egiaztatzean ikusi dugu eredua inprimatzeko modurik onena modu horizontala zela; izan ere, horrela, atzerako ispiluetarako euskarriak baino ez dira sortu behar, eta horrek inprimaketa ondoko lana errazten du.

Puntu hau amaitu baino lehen, nabarmendu nahi dugu guztia ez dela diseinua modelatzea eta inprimatzeko prestatzea; inprimaketa ondoko prozesu garrantzitsu bat ere badago, eta denbora nahiz dedikazioa beharko dira horretarako, eredua ordenagailuan dugun bezalaxe gera dadin. Inprimagailuak ez dira inoiz % 100 zehatzak izango; izan ere, esan dugun bezala, euskarriak inprimatu eta eskuz kendu beharko ditugu. Atal horiek bizarrak eta akatsak utziko dituzte, eta inprimagailu guztiek dituzten akats txikien ondorioz sortutakoei gehituko zaizkie. Horregatik, garrantzitsua da jakitea ezen, zenbat eta xehetasun-maila handiagoa behar duzun, orduan eta inprimaketa ondoko lixatze-lan eta arreta gehiago beharko direla.

Orain arte ikusitako guztiarekin, oinarrizko fitxategi bat sortuko genuke 3Dn inprimatzeko.

Inprimaketa-prozesua eta azken gogoetak

Eredua inprimatzea

Ondoren, inprimaketa-prozesua bera jorratuko dugu, bai eta azken gogoeta batzuk ere, 3D inprimaketan kalitatezko emaitzak lortzeko eta segurtasuna bermatzeko.
3D ereduarekin prestatuta eta aurreko puntuan zehaztutako urratsei jarraituta, inprimaketa-prozesua has daiteke. Inprimatzean, inprimagailuak hautatutako materialaren ondoz ondoko geruzak jarriko ditu objektua 3Dn eraikitzeko. Funtsezkoa da inprimaketa gainbegiratzea, izan litezkeen arazoak detektatzeko eta emaitza arrakastatsua bermatzeko.

Batez ere, erretxinazko inprimagailu bat erabiltzen badugu, kontuan izan behar dugu 3D inprimaketak material eta prozesu espezifikoak erabiltzea dakarrela, eta baliteke arrisku potentzialak egotea behar bezala erabili ezean. Garrantzitsua da inprimagailuaren fabrikatzaileak emandako segurtasun-jarraibideak betetzea eta neurriak hartzea inprimatzeko prozesuan lesiorik edo kalterik egon ez dadin.

Azken ondorioak

3D inprimaketak iraultza ekarri du fabrikazioaren alorrera, eta aukerak zabaldu ditu askotariko industrietan. Esportazio-formatuak, modelatze-softwarea eta inprimatze-kontsiderazio egokiak ezagututa, lehen imajinatzen zailak ziren 3D objektu pertsonalizatu eta funtzionalak sor daitezke.

Informazio gehiago

Aukeran duzu beti Adam Jorquera Ortegaren artikulu hau kontsultatzea, sakon jorratzen baitu bai 3D inprimaketa, bai helburu horretarako modelatzea: «***Fabricación digital: Introducción al modelado e impresión 3D***».

e.digitall.org.es/fabricacion-digital

Eduki digitalak sortzea

C2 maila 3.1 Edukiak garatzea

Adimen artifizialeko tresnen betekizunak eta instalazio lokala bideoa eta audioa sortzeko

Adimen artifizialeko tresnen betekizunak eta instalazio lokala bideoa eta audioa sortzeko

Beharrak ebaluatzea

Oro har, lokalean AA bat, *Stable Diffusion* adibidez, instalatzeko erabakia hartuko dugu honako alderdi hauen arabera: gure behar espezifikoak, pribatutasun- eta segurtasun-betekizunak, bai eta eskura ditugun baliabide eta gaitasun teknikoak. Kontuan izan behar dugu prozesu horrek hardware-eskakizun batzuk izango dituela —eta ondoren aipatuko ditugu—, baina, gainera, lokalean lan eginez gero, erabiltzaileok bihurtzen gara sistema mantentzearen eta haren eguneratzeen arduradun, eta gerta liteke, makina ahaltsurik ez badugu, eskalagarritasun-arazoak izatea; hau da, hodeian tresna berak datuak erabiltzeko ematen digun adinako ahalmenik ez izatea.

Alabaina, prozesu horrek baditu abantaila argi batzuk; honako hauek, besteak beste:

- **Gure datuak erabat kontrolatuko** ditugu. Hori bereziki garrantzitsua izango da pribatutasuna eta konfidentzialtasuna funtsezkoak diren kasuetan, datuak ez baitira gure ingurune lokaletik ateratzen.
- Abiadura eta latentzia handiagoak izango ditugu. AA lokalean exekutatzean, saihestu egiten dugu datuak prozesatzeko Interneteko konexio baten mendekotasunik egotea, bai eta sarea erortzeagatik eta kontratatutako banda-zabaleran mugengatik izan litezkeen akatsak ere. Horrenbestez, AAk azkarrago erantzun dezake, eta hori onuragarria da erantzuteko denbora azkarrak behar dituzten aplikazioetan.
- Pertsonalizazio- eta doikuntza-maila handiagoak. Malgutasun handiagoa izango dugu eredua pertsonalizatzeko eta gure behar espezifikoetara egokitzeko orduan. Aukera izango dugu sistemaren parametroak egokitzeko eta AA gure datuekin entrenatzeko, eta horrek egokigarritasun handiagoa eta errendimendu hobea emango digu gure helburuen arabera. Izan ere, nahikoa ezagutza badugu, Stable Diffusion programaren gure bertsio lokalaren iturburu-kodea aldatu ahal izango dugu, gure baldintza berezietara egokitzeko.

Instalatzeko eskakizunak

Stable Diffusion lokalean instalatzeak hardwarearen betekizun batzuk ditu, bai eta baliabide konputazionalen betekizun batzuk ere.

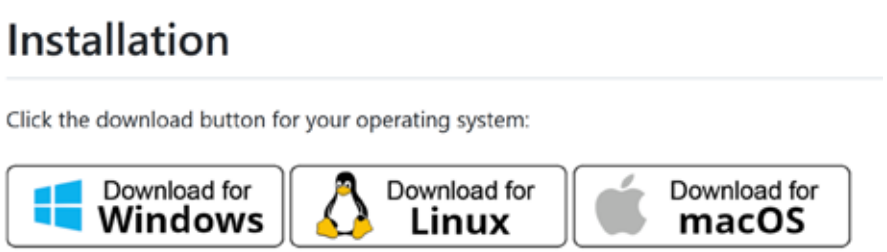
Batetik, ez dugu prozesadore-betekizunik, baina 256 Gb-tik gorako SSD disko gogorra beharko dugu (gomendatua), gutxienez 25 Gb librerekin, eta 8 Gb-ko RAM memoria gutxienez. Bestetik, gomendatzen da NVIDIA txartel grafiko dedikatu bat, 2 Gb-ko VRAMkoa gutxienez, emaitzak modu arinago eta zehatzagoan erakusteko, ordenagailua normaltasunez erabiltzeko behar adinako gaitasunarekin.

Logikoa denez, lan egiten dugun makina zenbat eta indartsuagoa izan, orduan eta azkarrago lortuko ditugu gure emaitzak. Kalkulatzen da Stable Diffusion programak, lokalean instalatuta, 5 segundo baino ez dituela behar 512x512 pixeleko irudi bat sortzeko, 12 GB-ko 3NVIDIA 3060 erabiliz gero.

Instalazioari buruzko tutoriala

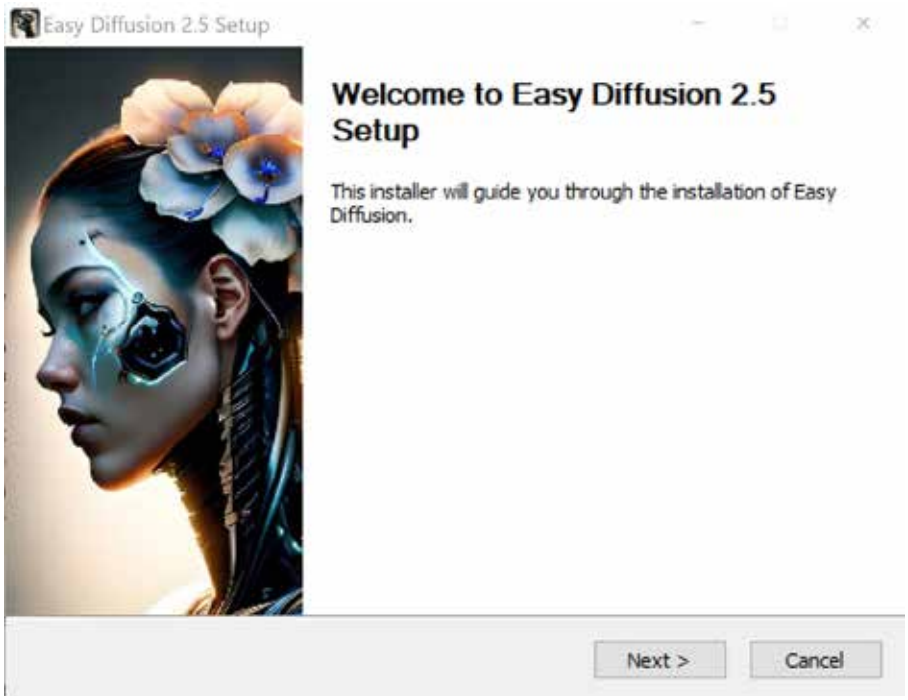
Gaur egun, nabarmen sinplifikatu da Stable Diffusion programa lokalean instalatzeko prozesua. Hasteko, GitHub-en Stable Diffusion UI-ren orrira jo behar dugu, eta zer sistema eragiletan ari garen lanean adieraziko dugu bertan, instalazio-programa konbentzional bat jaisteko (1. irudia).

GITHUB-EN STABLE DIFFUSION UI-REN ORRIA
e.digitall.org.es/diffusion



1. irudia. Stable Diffusion lokalean instalatzeko sistema eragileen aukerak.

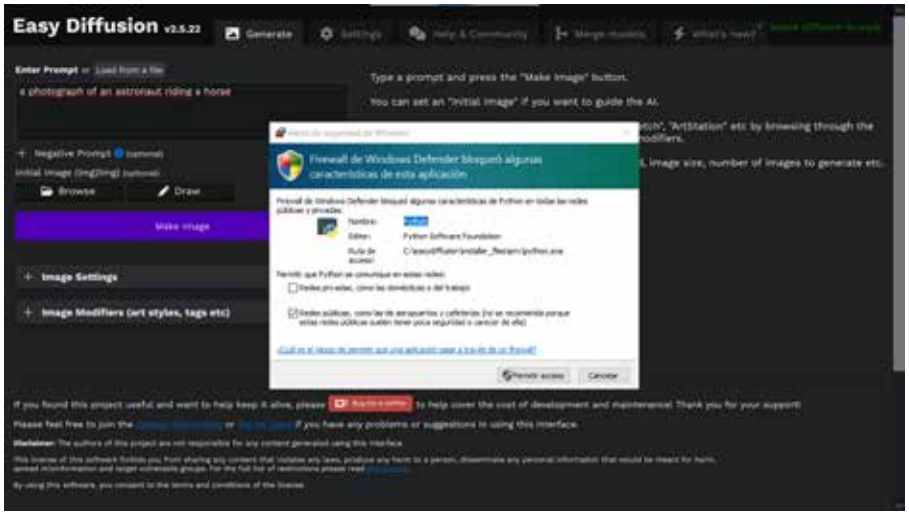
Ondoren, aski dugu instalatzailea exekutatzea eta lizentzia-akordioa onartzea gure PCan Stable Diffusion instalatzeko. Hau da instalazioan iradokitako esteka-bidea: ***C:\EasyDiffusion*** (2. irudia).



2. irudia. Stable Diffusion-en autoinstalatzeko programaren ikuspegia.

Behin instalazioa amaituta, programa (Start Stable Diffusion UI.cmd) exekutatu ahal izango dugu eta CMDKO leiho bat irekiko zaigu; leiho horretan ikusiko dugu sistemak behar bezala funtzionatzeko beharrezkoak diren osagaiak deskargatzen direla oraindik.

Behar diren gainerako fitxategiak deskargatu eta instalatu ondoren, aplikazioaren interfazea irekiko zaigu gure nabigatzailean, helbide honetan lehenetsita: ***http://localhost:9000/***. Windowsen bazaude, suebakiaren abisu bat agertuko da segur aski (3. irudia).



3. irudia. Windowseko nabigatzailearen itxura Stable Diffusion programa lokalean instalatzea exekutatzean.

Behin suebakiaren baimenak ezarrita, arazorik gabe jardun ahal izango dugu Stable Diffusion programaren instalazio lokalarekin. Instalazio lokalaren erabileraren adibideak ikusgai bideo honetan:

ADIMEN ARTIFIZIALEKO TRESNEN TOKIKO ERABILERA, TESTU-INFORMAZIOAN OINARRITUTA IRUDIAK SORTZEKO

e.digitall.org.es/A3C31C2V08

Informazio gehiago

Stable Diffusion zure ingurune lokalean instalatzeari buruz gehiago jakin nahi baduzu, GitHub-en haren orri ofiziala esploratzera animatzen zaitugu. Gainera, kode irekiko softwarea denez, gaur egun GitHub-eko beste bertsio batzuk daude Stable Diffusion lokalean instalatzeko. Azkenik, eta arrazoi beragatik, hau irakurtzen duzunean baliteke orain eskuragarri dauden bertsio horiek guztiak dagoeneko programa berriagoek ordeztu izana.

e.digitall.org.es/diffusion

DigitAll

Eduki digitalak sortzea

3.2

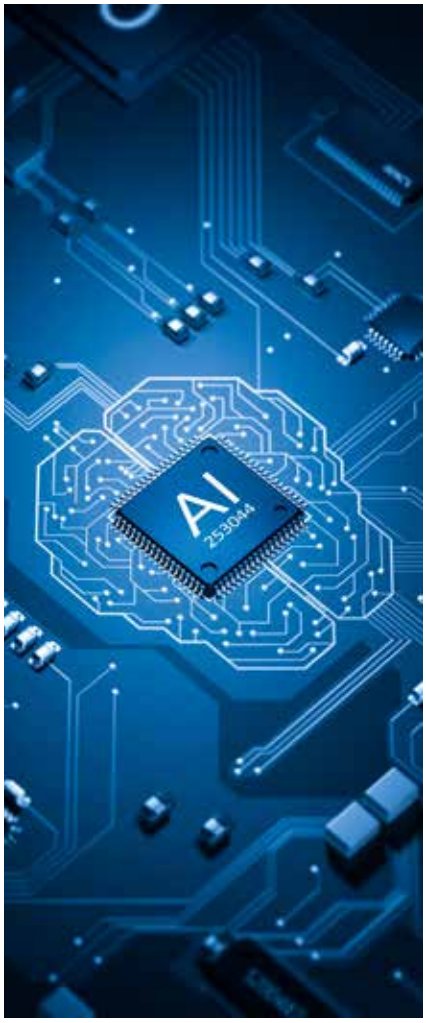
EDUKI DIGITALA INTEGRATZEA ETA BIRLANTZEA

Eduki digitalak sortzea

C2 maila 3.2 Eduki digitala integratzea eta birlantzea

Erabakiak hartzeko adimen artifizialeko tresnak

Erabakiak hartzeko adimen artifizialeko tresnak



Ikaste automatikoa datuetan oinarritutako adimen artifizial mota bat da, eta zenbait teknika barne hartzen ditu, ordenagailuek eta mikrokontrolatzaileek zeregin jakin batzuk egiten ikas dezaten. Teknika horiek hardware-software sistemetan integratzeak arau informatikoak aberasten ditu, eta haiei esker, programatzaileak ez dute programa baten logika zertan idatzi, hau da, **x** sarrera jakin bati **y** erantzuna dagokiola adierazten duen logika. Metodo horiek **kutxa beltz** gisa jarduten dute; **x** sarrera jasotzen dute eta **y** irteera itzultzen dute. Metodo horiek sistemaren ekintza batzuk gidatzen dituzte, bertan nolabaiteko *adimena* gehitzen baitute.

Demagun gure hardware-software sistemak **x** sarrerako aldagaien bektore bat duela (adibidez, sentsore-multzo batek bilduak), eta sistemak **y** erantzuneko aldagai bat erabaki behar duela. Aldagai horrek eragingailu multzo baterako erantzuna edo prozesatzen jarraitzeko behar den barne-datu bat jaso dezake. Ikaste automatikoko arazoen sailkapena bi irizpidetan oinarritzen da: I) erantzuneko aldagaia (**y**) ezagutzeko aukera dagoen edo ez, eta II) aldagai horren tipologia. Aldagaien tipologia hau bereizten da:

- **Aldagai kuantitatiboak**. Datuek zenbakizko balioak hartzen dituzte. Era berean, bi hauek bereizten dira: **aldagai jarraituak** (balioak zenbaki errealen tarte batean hartzen dituzte) eta aldagai diskretuak (zenbakizko balioetatik zenbaki finitua hartzen dute).
- **Aldagai ordinalak**. Datuek behaketen arteko ordena-erlazioa adierazten dute. Adibidez, ranking bat egin dezakegu.
- **Aldagai kualitatiboak**. Kasu horretan, objektu baten nolakotasun bat adierazten da. Aldagai horiek balio-multzo bat baino ezin dute hartu, eta ez dute magnitude jakinik neurtzen. Adibide bat aldagai **bitarrak** dira; haietan bi balio baino ezin dira hartu, eta ezaugarri baten presentzia/gabezia adierazten dute.

Ikaste automatikoko algoritmoen taxonomia bat **x** eta **y** sistemaren aldagaien izaeran oinarritzen da. Bi hauek bereizten dira: **ikaste gainbegiratua**; aldez aurretik etiketatutako datu-multzo bat dugu oinarrian, hau da, datu-multzo bat ezagutzen dugu (**x**i,**y**i). Eta ikaste **ez-gainbegiratua**; aldez aurretik etiketatu gabeko datuak ditu oinarri, hau da, informazioa baino ez dago (**x**i). Ikaste ez-gainbegiratuaren barruan arazo hauek agertzen dira: **clustering**, **dimentsionaltasuna murriztea** eta **outlier-ak detektatzea**. Lehenengoak sistemaren patroiak zehazten ditu. Bigarrenak sortutako **x** datuen tamaina murrizten du, ahalik eta informazio gutxien gal dadin. Adibidez, sentsoreek neurketak denbora errealean egiten badituzte, transmititu beharreko datu kopuru handiak sor ditzakete. Adibide nabarmen bat sateliteen bidezko irudi-transmisioa da. Gailuek komunikazio-ahalmen txikia badute, gomendagarria da teknika horiek aplikatzea. Outlierrak detektatzeak esan nahi du software/hardware sistemaren egoera (patroia) ez datorrela bat *a priori* funtzionatzeko planifikatutakoekin, eta ondorioz, alarma edo jakinarazpenen bat bidali behar dela.

Ikaste automatikoan oinarrizko hiru arazo mota agertzen dira:

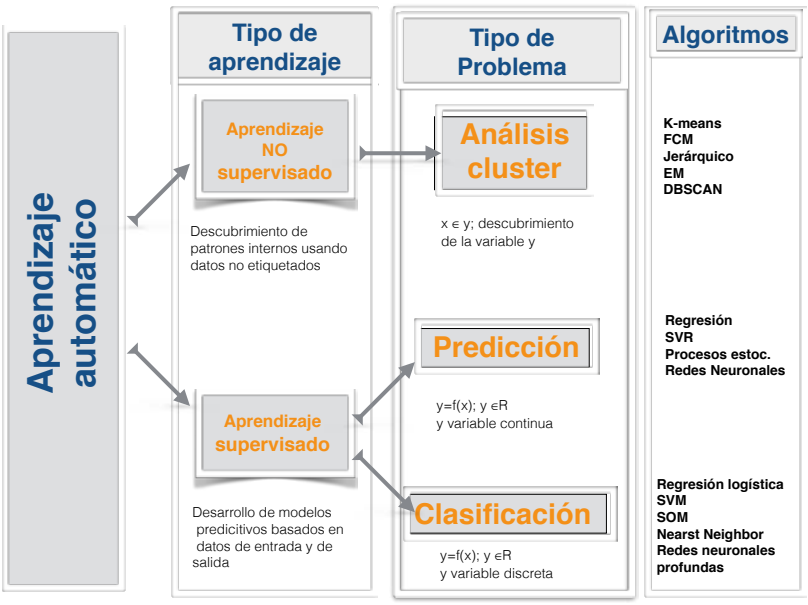
1 | **Clustering**. Arazo honetan patroiak bilatzen dira datuen barruan. Hau da, elkarren antzekoak diren behaketa-azpimultzoak aurkitzea. Matematikoki, honela formulatuko litzateke: datuen y etiketak determinatzea, halako moldez non, bi i eta j behaketa antzekoak badira, etiketa bera lotzen zaien (Yi=Yj). Arazo horren zailtasun bat zera da: ez dago aurretiko etiketa multzo bat, haietatik ikas dezagun. Era horretako arazoen adibide bat datu-taldekatzea da. Horri esker, sistema egongo den patroi multzo finitua identifika daiteke.

2 | **Erregresioa**. Arazo honetan **y** aldagai jarraitu baten balioa iragartzen da **x** aldagaiaren balioa ezagutu ondoren. Erregresio-testuinguruetan, **x** aldagaiari erregresore edo aldagai azaltzaile esaten zaio. Sistema horiek **y**=f(**x**) erregresio-funtzioak kalkulatzen dituzte eta ahalbidetuko lukete, adibidez, eragingailu jakin baten **y** balioa ematea **x** sistemaren oraingo egoerarako.

3 | **Sailkapena**. Arazo honetan, **y** aldagai kualitatiboaren balioa iragarri behar da **x** aldagaia oinarri hartuta. Sailkapenean, **x** aldagaiari **ezaugarri** edo **atributu** esaten zaio, eta **y** aldagaiari, berriz, **etiketa** edo **klase**.

Adibidez, auto autonomoan hiru zutabetan oinarritzen da: *IoT*, *Big Data*ri aplikatutako ikaste automatikoko teknikak eta Interneteko konexioa denbora errealean. Ibilgailuaren *begiak* eraikitzean, ibilgailua inguratzen duten irudiak sailkatzeko arazoa konpondu behar da, era horretan zehaztu dadin zer objektu mota dagoen ibilgailuaren aurrean, atzean eta alboetan.

1. irudiak ikaste automatikoko ereduen sailkapen hori laburbilduta dago. Algoritmo ugari proposatu dira problema mota bakoitzerako. Ez dago egoera guztietarako besteak baino hobea den metodorik. Horrenbestez, *erreminta-kutxa* eduki behar da askotariko soluzioak probatzeko. Adibidez, *K-means* algoritmoa azkarra da eta ondo funtzionatzen du clustering-arazoetan, baldin eta patroiak *linealki* (hiperplanoen bidez) *bereizteko* aukera badago. Beste egoera batzuetan, baliteke algoritmo horrek ondo ez funtzionatzea, eta, ondorioz, bestelako tekniketara jo beharko genuke, hala nola dentsitatean oinarritutako algoritmoetara (DBSCAN). Beste adibide bat **sare neuronal sakonak** dira; errendimendu handia erakusten dute arazo horietan guztietan. Hala ere, gailu jakin batzuen kasuan, eraginkorrak badira ere, baliteke aplikaezinak izatea kostu konputazionala dela eta.



1. irudia. Ikaste automatikoko problemak sailkatzea.

Teknika horiek **ikaste induktiboa** erabiltzen dute; izan ere, adibide zehatzen behaketa eta azterketa oinarri hartuta, ereduak garatzen dira datu horiek azaltzeko eta orokortzeak egiteko. Modelatzaileak **ereduak** entrenatu behar ditu eskura dituen datu-multzoak oinarri hartuta, eta entrenatutako ereduak sistemetan sartzen dira.

1. taulan ikusgai daude ikaste automatikoko ereduak ezartzeko tresna ezagunenetako batzuk. Tresna erabiliko da proiektuaren beharrak eta lehentasunak aintzat hartuta.

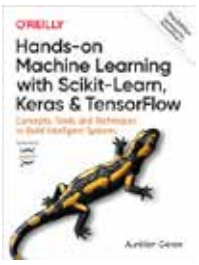
1. taula. IKASTE AUTOMATIKOKO EREDUAK ERAIKITZEKO TRESNAK

Izena	Ezaugarriak	Garatzailea
TensorFlow	Ikaste automatikoko ereduak sortzeko, entrenatzeko eta ezartzeko tresnen bilduma. Python eta C++ erabiltzen ditu programazio-lengoaia gisa.	Google
scikit-learn	Python-erako kode irekiko ikaste automatikoko liburutegi bat da.	
PyTorch	Ikaste sakoneko framework ezaguna. Bere interfazea malguagoa da TensorFlow tresnarena baino.	Facebook
Keras	Keras kode irekiko sare neuronalen liburutegi bat da, Python-en idatzitakoa. TentsoreFlowen tresnaren gainean exekutatzeko gai da.	
Microsoft Azure ML	Hodeiko plataforma, datuak prestatzeko ziklo osorako, ikaste automatikoko ereduak inplementatzeko eta produkzioan monitorizatzeko.	Microsoft

Informazio gehiago

Teknika horiek Python-en integratzeko liburu bikaina: *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow.*

e.digitall.org.es/hands-on



Eduki digitalak sortzea

C2 maila 3.2 Eduki digitala integratzea eta birlantzea

Robot programagarriak garatzeko tresnak

Robot programagarriak garatzeko tresnak

Robot programagarriek protagonismo handia hartu dute azken urteotan, zereginak automatizatzeko eta askotariko eremuetan efizientzia hobetzeko duten gaitasuna dutelako, hala nola industrian, hezkuntzan eta etxean. Robotak diseinatzea eta programatzea ez da lan erraza, beharrezkoa baita ezagutza espezializatuak izatea hainbat arlotan (elektronika, mekanika edo informatika).

Alabaina, tresna eta plataforma askok prozesu hori errazten dute, eta aukera ematen diote edozein pertsonari robotak modu erraz eta azkarrean garatzeko, esperientzia-maila edozein dela ere.

Plaka programagarriak deritzonak dira tresna nagusia, hala nola **Arduino** eta **Raspberry Pi**. Plaka horiek gailu txikiak dira, eta kontsumo txikiko mikroprozesadore bat dute, osagai batzuei konektatua; besteak beste: RAM memoriako moduluak, datuak memoria-txartel baten bidez biltegiratzeko konexioa, sarrera-irteerako osagaiak (USB atakak, Ethernet atakak, Wi-Fi eta Bluetooth moduluak, GPIO konexio-pinak, eta abar), baita txartel grafiko txikiak eta irudi-prozesadoreak ere. Plaka horiek zenbait sentsore eta eragingailu desberdinetara konekta daitezke, eta haien funtzionamendua kontrolatzen dute askotariko programazio-lengoaietan idatzitako programen bidez.

Nahi dugun plaka programagarria daukagunean, **programazio-lengoaia** da oinarrizko beste tresna bat. Hizkuntza ezagunenetako batzuk **C, C++, Python** eta **Java** dira. Lengoaia horien bidez kodea idatz daiteke robotari adierazteko nola funtzionatu behar duen eta kanpoko munduarekin nolako elkarreragina izan behar duen. Plaka bakoitzak programazio-lengoaien multzo bat onartzen du:

- Adibidez, Arduino plakek C++ lengoaian oinarritutako programazio-lengoaia propioa erabiltzen dute, Arduino izenekoa. Nolanahi ere, programazio-tresna batzuek bestelako lengoaia alternatiboak erabiltzea ahalbidetzen dute, hala nola C, C# edo Python, eta haiek, ondoren, Arduino bihurtzen dira plakara bidali aurretik.

- Bestalde, Raspberry Pi plakak Linuxen oinarritutako sistema eragile propioa du, Raspbian izenekoa. Horregatik, plakak programazio-lengoaia ugari onartzen ditu, hala nola C, C++, Python, Java, Ruby, Perl eta abar.

Informazio gehiago

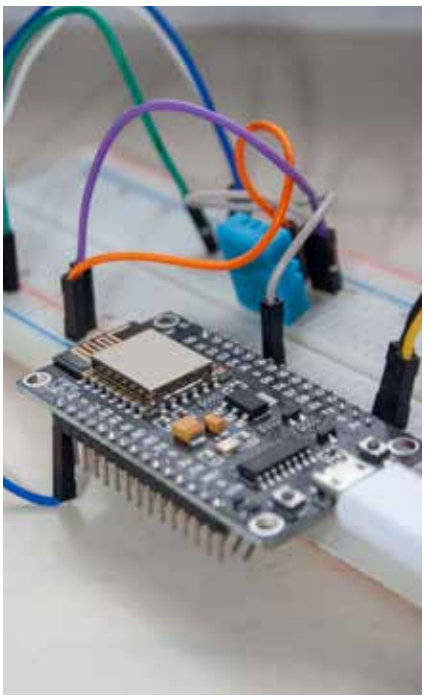
Raspbian da Raspberry Pi-rako sistema eragile ezagunena eta erabiliena, baina badira bestelako sistema eragile bateragarri ugari ere. Adibidez: Windows 10 IoT Core, Ubuntu Core/Desktop/Server, Raspbian, OSMC, Kali Linux, etab.

Programazio-lengoaiez gain, askotariko **garapen-aplikazio eta simulagailuak** daude, eta aukera ematen dute roboten portaera diseinatu eta simulatzeko, haiek eraiki aurretik. Tresna horiek programazio-lengoaiak osatzen dituzte, haien aukerak handituz eta simulazio-gaitasunak gehitzen dituzte. **Simulagailuak** bereziki erabilgarriak dira ingurune birtual batean egoera desberdinak probatzeko eta robotaren funtzionamendua optimizatzeko, fisikoki eraiki eta mundu errealean probatu beharrik gabe. Aplikazio horiei esker, kalteak edo akatsak saihesten dira robotaren garapenean; izan ere, halako akatsak baliteke oso garestiak izatea, robota diseinatzeko, programatzeko eta eraikitzeko etapara itzultzea ekarriko luketelako.

Informazio gehiago

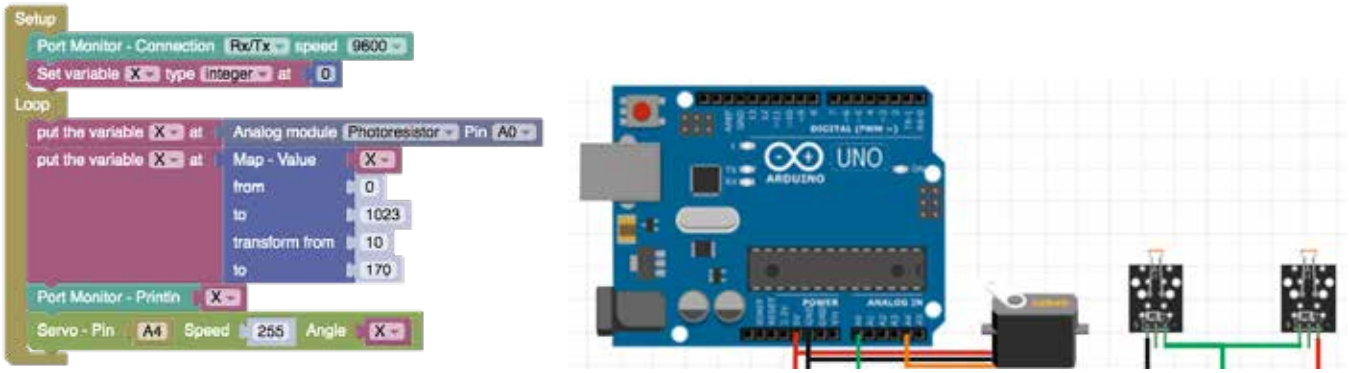
SimulIDEa da Arduino-rako simulagailu erraza eta doakoa, eta hemen dago eskuragai: *simulide.com*. Simulagailu hori Windows, MacOS eta Linuxen erabil daiteke.

Ezagutza eta trebetasun espezifiko asko uztartu behar dira robotak diseinatu eta programatzeko, baina agertu diren garapen-tresna jakin batzuei esker, **edozein pertsonak ikas dezake bere robot programagarria sortzen**. Gainera, tresna horietako batzuen bidez robotak programatzeko aukera dago kodea idatzi edo programatzen jakin beharrik gabe. Arduinorako, adibidez, ArduBlock tresna dago.



ArduBlock tresnak Arduino-n programatzea ahalbidetzen du bloke grafikoak puzzle-piezak izango balira bezala erabiliz.

Tresna horren bitartez programa txikiak egin daitezke programazioko aurretiko esperientzia izan beharrik gabe. Gaur egun, tresna hori hemen dago eskuragai: ***ardublock.ru/en***. ArduBlock tresna erabiliz egindako proiektu baten adibidea dugu 1. irudian.



1. irudia: ArduBlock-eko proiektu baten adibidea, eguzkiaren ibilbideari jarraitzen dion serbomotorra bat bi fotorresistore erabiliz kudeatzeko. *Iturria: ardublock.ru*

Informazio gehiago

Scratch blokeen bidezko programazio-tresna bat da, MITek garatutakoa, programatzen hasiberriak direnentzat. Bideojokoak programatzera bideratuta dago batez ere. Scratch-en aldaketa batek, **S4A** izenekoak (*s4a.cat*), Arduinon programatzeko aukera ematen du Scratch lengoaia erabiliz.

Garapen- edo robotizazio-kitak izenekoak ere badaude; kit komertzial txikiak dira, eta barne hartzen dituzte robot bat eraiki eta programatzeko behar diren osagai guztiak. Kit batzuek plaka programagarriak berak, sentsore- eta eragingailu asko eta behar diren kable guztiak, baita programazio-tresna espezifikoak ere. Kit horiek gidaliburuak eta tutorialak izaten dituzte muntatzeko eta programatzeko prozesua errazte aldera.

Arduino proiektuaren webguneak berak badu **Arduino Starter Kit** (*e.digitall.org.es/arduino*) izeneko hastapenerako kita. Hala ere, Arduino-rako kit ugari daude elektronika-dendetan. Besteak beste, haurrentzako kitak, teknologiazale sutsuentzako kitak eta alorrean hasi nahi dutenentzakoak. Askotariko kitak daude, halaber, Raspberry Pi-rako, eta haietako batzuek zure ordenagailu pertsonala eraikitzeko aukera ematen dute. Adibidez, **Raspberry Pi 400 Personal Computer Kit** (*e.digitall.org.es/raspberry*) izenekoak.

Informazio gehiago

Beste tresna garrantzitsu bat **online foroak eta komunitateak** dira; halakoetan tutorialak, adibideak eta robot-garapenarekiko interesa duten beste erabiltzaile batzuen aholkuak aurkituko ditugu. Komunitate horiek inspirazio- eta ikaskuntza-iturri agortezinak izan ohi dira.

Laburbilduz, hainbat tresnak eta paradigmak errazten dute robot programagarriak garatzea. Besteak beste, plaka horien programazioari aplikatutako helburu orokorreko programazio-lengoaiak, garapen-aplikazioak, simulagailuak eta garapen-kitak. Ikuspegi horietako bakoitzak bere abantailak eta desabantailak ditu, eta garrantzitsua da tresna egokia aukeratzea proiektuaren beharrak aintzat hartuta.

DigitAll

Eduki digitalak sortzea

3.3

EGILE-ESKUBIDEAK ETA JABETZA INTELEKTUALEKO LIZENTZIAK

Eduki digitalak sortzea

C2 maila 3.3 Egile-eskubideak eta jabetza intelektualeko lizentziak

Copyrighta erregistratzea eta AAren laguntzaz sortutako obra bat ustiatzea

Copyrighta erregistratzea eta AAren laguntzaz sortutako obra bat ustiatzea

Dokumentu honen edukiak gaurkotasun handiko gai bati buruzko gogoeta eginaraziko dizu.

Gaur egun, sortzaileek tresna iraultzaileak dituzte Interneten; tresna horiek adimen artifiziala erabiltzen dute era guztietako eduki artistikoak sortzeko.

Tresna horiei esker, artistek, musikariek, idazleek eta beste sortzaile batzuek muga berriak esploratzen dituzte, eta inspirazioa ere aurkitzen dute horrelako tresnetan.

Musikaren esparruan, aukera ematen dute melodia konplexuak, harmonia nahiko erakargarriak eta orain arte entzun gabeko erritmoak konposatzeko. Bestalde, pinturaren eremuan maisu handien erreprodukzio fidelak edo interpretazio artistiko berriak sor daitezke.

Denok ezagutzen dugu ChatGPT tresnak zer-nolako arrakasta izan duen testu originalak sortzeko. Beste adibide batzuk: DALL E 2 tresna (irudiak sortzen ditu ematen zaizkion deskribapenak oinarri hartuta); AIVA (musika-pista originalak konposatzen ditu); AI Time Machine (erabiltzaileek pertsona baten irudiak sor ditzakete, historian zeharreko aldi desberdinetan).

Baina zer gertatzen da tresna horiek erabiliz sortzen diren obra guztiekin? Lor al daiteke «**adimen artifiziala**»ren laguntzaz sortutako lan baten copyrighta?

Herrialde gehienetan, **egile-eskubidearen edo Copyrightaren bidez babes daitezke gizaki batek sortutako lan originalak**.

Aurreko hamarkadetan, zenbait sortzailek konputagailua behar eta erabiltzen zuten lanak sortzeko tresna gisa, idazle batek boligrafoa eta papera erabiltzen zituen bezala, beste batek bere lana idazteko ordenagailu-programa bat erabiltzen zuen bezala, pintorea pintzelaz eta mihiseaz baliatzen zen bezala, eta musika elektronikoaren sortzaileak sintetizagailu bat behar zuen bezala. Kasu horietan guztietan sortzailearen sormenari ematen zitzaion lehentasuna.

Hala ere, azken urteotan gertatzen ari den iraultza teknologikoak, neurri batean «**ikaste automatikoko softwarea**»ren garapenak bultzatuak, behartzen gaitu konputagailuen eta sorkuntza-prozesuaren arteko interakzioa birpentsatzera aipatutako softwareak esku hartzen duenean.

Informazio gehiago

Ikaste automatikoko softwarea, AAren formetako bat, programa informatiko bat da, eta ikas dezake softwarean bertan gehitzen ditugun datuetan oinarrituta. Datu berriak sartuta eboluzionatu egiten du, eta erabaki berriak hartzen ditu; erabaki horiek bideratuak edo autonomoak izango dira.

Sortzaile batek, adibidez, pintore, idazle edo konpositore batek bere lana egiteko ikaste automatikoko software bat erabiltzen duenean, sorkuntza-prozesuan softwarea bera ikasten ari da sortzaileak edo programatzaileak gehitzen duen informazioaren bitartez, eta datu horiek oinarri hartuta, erabaki independenteak hartzen hasten da, zeinen emaitza artelan berri bat den. Kasu horietan egiaz zera gertatzen da: **sortzaileak sorkuntza-prozesuko parametro batzuk definitzen baditu ere, obra programa informatikoak sortzen du**.

Aurrekoa bezalako kasuetan, ondoriozta genezake programa informatikoa ez dela jada sortzaileak erabiltzen duen tresna bat, arestian aipatu ditugun kasuetan bezala; aitzitik, sortzaileak esku hartu gabe hartzen ditu sormen-prozesuari lotutako erabakiak, eta, ondorioz, uler liteke sortzailea programa bera dela, eta ez gizakia. Espainiako legeria kontuan hartuta, **ikaste automatikoko software** bat (alegia, adimen artifizialaren azpitalde bat) erabiliz sortutako lan edo obra bat ezin da jaso egile-eskubideen erregistroan.

Adimen artifizialak esku hartuta sortutako obrak ezin badira egile-eskubidearen bidez babestu; gizakiak sortu ez dituela iritzita, edonork erabil ditzake libreki, plagiorik egin gabe, eta hori oso arazo larria izango da obra horiek saltzen dituzten sektoreko enpresentzat. Adibidez, demagun filmen ekoiztetxe batek diru asko inbertitzen duela bere filmetarako musika sortzen duen sistema bat garatzeko, eta, ondoren, legeak ez diola babesten uzten; sistema munduko edozein pertsonak mugarik gabe erabil dezake, eta horrek kalte ekonomiko handia eragingo dio enpresari.

Gaur egun, arazo hori ez dago konponduta; hala ere, zantzuen arabera, herrialde askoren legeria ez da gizakiari aplikatzen ez zaion egile-eskubidearen aldekoa, Estatu Batuetan, Australian eta Europako herrialde askotan ebatzi bezala. Hala ere, beste herrialde batzuetan, hala nola Hong Kongen, Indian, Irlandan, Zeelanda Berrian eta Erresuma Batuan, prest egongo lirateke obra sortzeko beharrezkoak diren konponketak egiten dituen pertsonari egiletza emateko».

Etorkizun hurbilean, informatikak aurrera egin ahala, adimen artifizialaren erabilera orokorragoa denean eta sortzaile gehienek ikaste automatikoko softwarea erabiltzen dutenean, ordenagailuek gero eta obra sortzaile hobeak sortuko dituzte, eta zaila izango da bereiztea gizaki batek egindako artelana eta makinak egindakoa, eta, orduan, erabaki beharko da zer-nolako babesa eman beharko zaien gizakiak gutxi edo ezer ez esku hartuta sortutako obrei.

Obraren ustiapen komertzialaren ikuspegitik, zentzuzkoena izango litzateke egile-eskubidea ematea ikaste automatikoko softwarearen funtzionamendua ahalbidetzen duen pertsonari. Horrela, batetik, bermatuko litzateke bai sorkuntza-industriaren jarraitutasuna, bai obrak ustiatzetik bizi den artistaren iraunkortasuna, eta, bestetik, enpresek teknologian inbertitzen jarraitzea, inbertsioari etekinak lortuko dituztela jakiteko segurtasunarekin.

Informazio gehiago

Artikulu honetan, adimen artifizialari (AA) buruzko informazioa aurkituko duzu, modu erraz eta argian azalduta:

e.digitall.org.es/inteligencia-artificial

DigitAll

Eduki digitalak sortzea

3.4 PROGRAMAZIOA

Eduki digitalak sortzea

C2 maila 3.4 Programazioa

Programazio-paradigmak. Printzipio orokorrak

Programazio-paradigmak. Printzipio orokorrak

Gaur egun, estilo desberdinek definitzen dute nola ekin programa baten diseinuari eta idazketari; horri *programazio-paradigma* esaten zaio. Horietako bakoitzak besteengandik bereizten duten ezaugarri espezifikoak ditu (arauak, teknikak eta printzipioak), eta, hala, beharren arabera, egokiagoak izango dira arazo mota batzuk edo beste batzuk konpontzeko. Hauek dira programazio-paradigma erabilienak: inperatiboa, objektuetara bideratua eta funtzionala. Garrantzitsua da bakoitzaren abantailak eta eragozpenak ezagutzea, arazo bati aurre egiteko egokiena hautatzeko, horrela kodea irakurgarriagoa, mantengarriagoa eta eskalagarriagoa izango baita. Ildo horretan, programazio-lengoaiek ere badute zeregin garrantzitsurik, programazio-paradigma bat edo gehiago onartzeko diseinatuta baitaude. Horrek esan nahi du paradigma batzuk ezartzen errazagoak edo eraginkorragoak izango direla programazio-lengoaia batzuetan beste batzuetan baino. Adibidez, objektuetara bideratutako programazioa asko erabiltzen da Java eta C++ lengoaietan, besteak beste, eta programazio funtzionalerako, berriz, Haskell eta Lisp lengoaiak erabiltzen dira. Gainera, programazio-lengoaia batzuk paradigma bakar bateko espezifikoak dira, hala nola Pascal eta C lengoaiak, nagusiki programazio inperatiborako erabiltzen baitira. Beste lengoaia batzuk paradigma anitzekoak dira, eta horrek esan nahi du programa berean paradigma bat baino gehiago erabiltzea ahalbidetzen dutela. Adibidez, Python paradigma anitzeko programazio-lengoaia bat da, eta programazioa inperatiboa, objektuetara bideratua eta funtzionala onartzen ditu. Programazio-lengoaia bat paradigma asko onartzeko gauza bada, garatzaileek malgutasuna eta boterea izango dute proiektuari gehien egokitzen zaion programazio-ikuspegia hautatzeko.

Jarraian, hiru programazio-paradigma ohikoenak deskribatuko ditugu, baita egokienak zaizkien aplikazio motak ere:

- **Programazio inperatiboa**ren ezaugarria da arretagunea jartzen duela arazoa konpontzeko «modua»n, eta, ondorioz, programek zehatz-mehatz deskribatzen dute arazo bat konpontzeko eman beharreko urrats-sekuentzia. Ideia da instrukzioak erabiltzea ***programaren egoera aldatzen*** joateko. Horretarako, aldagaiak daude, hau da, datuak biltegiratzea ahalbidetzen duten elementuak (sarrerakoak zein irteerakoak), eta begiztak eta baldintzak, jarraibideak gauzatzeko ordena kontrolatzen dutenak. Gainera, azpiprogramak (prozedurak edo funtzioak) kodea modularizatzeko eta berrerabiltzeko erabiltzen dira. Paradigma horren abantaila nagusia da oinarrian duen eredua oso intuitiboa dela, «urratsez urratseko jarraibide-eskuliburu» baten oso antzekoa baita. Horregatik erabiltzen da, hain zuzen, programatzen irakasteko eta ikasteko. Hala ere, arazo konplexuagoak ebazteko orduan, baliteke kodea luzeegia izatea, eta horrek zailtzea kodeari eustea.
- **Objektuetara bideratutako programazioak (OBP)** elkarreraginean ari diren objektu-bildumatzat hartzen du programa bat, bizitza errealean gertatzen denaren antzera. **Objektu** bat **klase** bateko ale bat da, eta klasea zera da: objektu multzo baten adierazpide, propietate eta portaera komunak definitzen dituen «txantiloia» edo eredua. Objektuaren adierazpidea **atributu** deritzen aldagaien bidez egiten da. Adibidez, programazio-irakasgaiko ikasleen propietate komunak definitzeko, hala nola izena, abizenak eta kalifikazio alfanumerikoa («gutxiegi», «nahiko», «oso ongi» eta «bikain»), Ikaslea izeneko klase bat defini liteke, Izena, Abizenak eta Kalifikazioa izeneko hiru aldagairekin (atributuak), hurrenez hurren. Objektuen portaera **metodo**ek definitzen dute; horiek dira, hain zuzen, objektuak manipulatzeko eta beste objektu batzuekin elkarreragiteko eragiketak inplementatzen dituzten prozedurak edo funtzioak. Adibidez, Ikaslea klaseak eduki beharko lituzke, gutxienez, ikasle baten izena, abizenak eta kalifikazioa ezagutzeko metodoak.

1. irudia. *Ikaslea* klaseko objektuaren adibidea.

Eraikitzailea izeneko metodo berezi bat dago, eta hori deitzen da ***objektu bat sortzeko***. Horrela, Ana Sánchez Megía ikasleari dagokion objektua sortzeko, eraikitzailea deitu genezake izena eta abizenak parametro gisa daudela. Emaitzak dakar hiru eremuko erregistro baten antzeko «aldagaia» sortzea, 1. irudiak erakusten duen bezala.

Behin klase bat definituta, behar adina objektu edo instantzia sor daitezke.

OBPa hiru zutabe hauetan oinarritzen da:

- **Kapsularatzea** objektuen irudikapen zehatza ezkutatzeko erabiltzen da, eta horrek bidegabeko erabileretatik babesten ditu. Adibidez, existitzen ez den kalifikazio bat esleitu ezin izatea edo bi kalifikazio batu ezin izatea.
- **Herentzia** azpiklaseak sortzeko erabiltzen da, hau da, beste klase batetik eratorritako klaseak sortzeko; haien propietateak eta metodoak jasotzen dituzte, baina beste batzuk ere izan ditzakete. Adibidez, IkasleErrepikatzailea klasea sor liteke, Ikaslearen azpiklase gisa, irakasgaia aurreko urte batean egin zuten ikasleak adierazteko, EgiteUrtea atributu espezifikoarekin, heredatzen dituenez gain (Izena, Abizenak, Kalifikazioa).
- **Polimorfismoa**: klase desberdinetako objektuek antzera jokatzea ahalbidetzen du, testuinguruaren arabera. Adibidez, IkasleErrepikatzailea klaseko objektuak ere badira Ikaslea klaseko objektuak, eta, beraz, bi moduetan joka dezakete. Horrela, Ikaslea klaseko objektuez osatutako irakasgaiko ikasle guztien zerrendan egon daitezke.

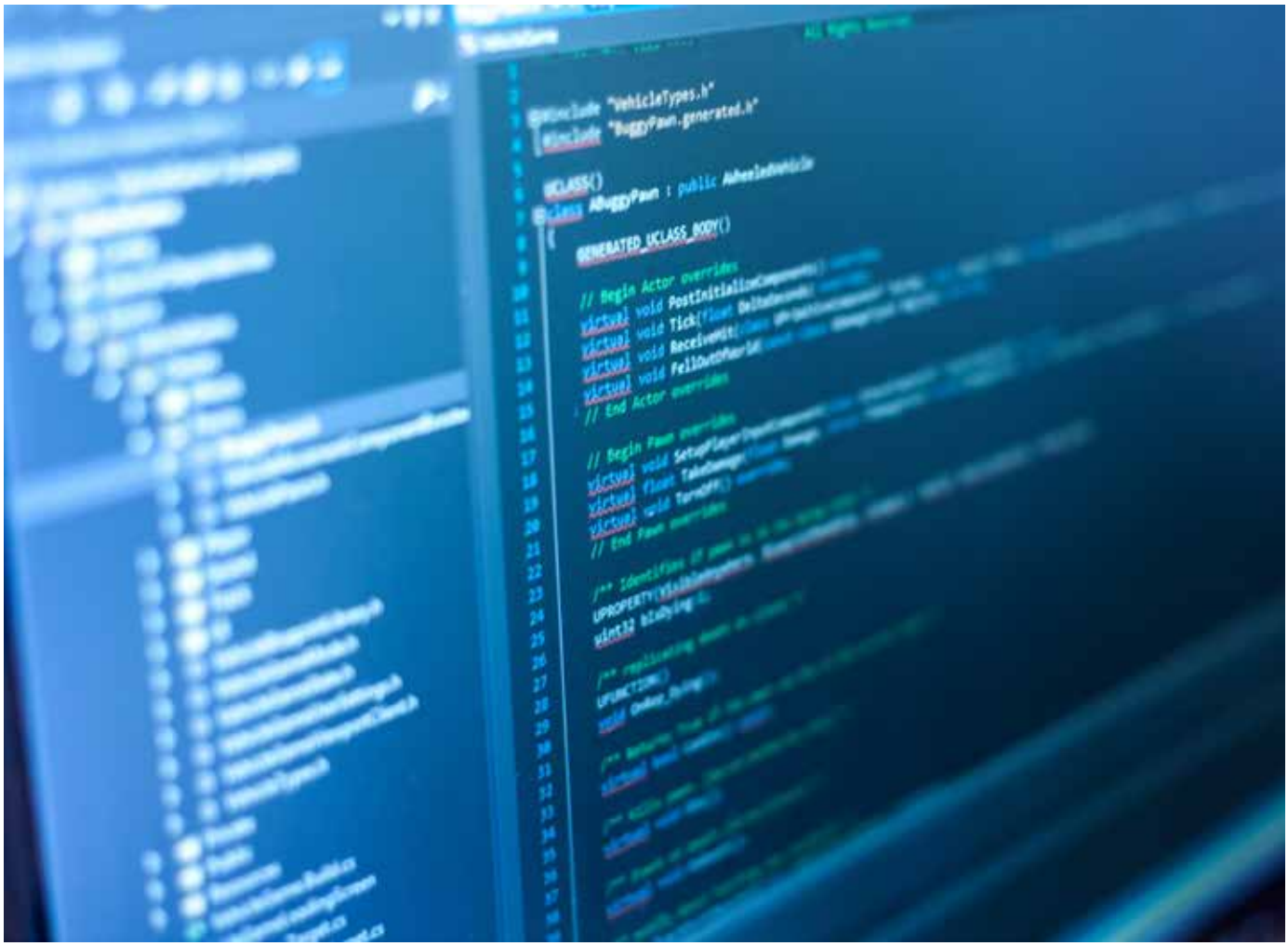
  Polimorfismoak malgutasun handia ematen die programei, eta hori paradigma horren abantaila da; herentziak kodea berrerabiltzea errazten du, eta kapsularatzeak akatsak prebenitzen laguntzen du. Gainera, klaseen diseinuak aukera ematen du programa moduluka lantzeko, eta, gainera, objektu berriak txertatzea eta lehendik daudenak aldatzea errazten du. Horrenbestez, paradigma egokia da aplikazio handiak taldean garatzeko. Hala ere, kontuan izan behar da programak motelago exekutatzen direla, eta balitekeela klaseak diseinatzea konplexuegia izatea.

- **Programazio funtzionala** programazio deklaratiboaren paradigman kokatzen da; haren bidez, programek zer lortu nahi duten zehazten dute, nola egin deskribatu gabe. Paradigma funtzionalean, programak idazteko erabiltzen diren elementuak **funtzio puruak** dira, hau da, sarrera bererako beti emaitza bera sortzen dutenak. Paradigma inperatiboan edo objektuetara bideratutakoan ez bezala, ez da komeni programaren egoera aldatzea, eta, ondorioz, datuak **aldaezinak** dira: programa exekutatzean, datuak ez dira aldatzen; beste batzuk sortzen dira. Horretarako, ezinbestekoa da funtzioak behar bezala parametrizatuta egotea eta balio egokiekin deitzea. Kontuan izan behar da funtzioak beste funtzio baten parametro gisa ere erabil daitezkeela. Gainera, funtzioek ez dute nahi gabeko albo-ondoriorik sortzen. Bestalde, begizten ordez errekurtsioa erabiltzen da. **Errekurtsioa** gertatzen da funtzio batek bere buruari deitzen dionean; horrenbestez, errepikapena sortzen da, eta bertan lehengo balioak aldatu gabe mantentzen dira. Funtzio errekurtsiboaren adibide bat da 1-etik n-ra arteko batura kalkulatzen duena, non n argumentua den. Honela definitzen da:

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

Era horretan, Batura(3)=Batura(2)+3; baina Batura(2)=Batura(1)+2. Batura(1)=1 gisa, orain deiak «desegiten» joango dira, hau da, orain Batura(2)=1+2=3 kalkula daiteke, eta, ondoren, Batura(3)=3+3=6. Lortutako balioa 1+2+3 eragiketaren emaitza da.

Zenbait abantaila ditu programazio funtzionalak: kodea zehatzagoa eta adierazkorragoa izaten da, funtzioak txikiak eta elkarren independenteak izaten baitira, eta horrek programak mantentzea errazten du. Gainera, datuen aldaezintasunak albo-efekturik ez egotea dakar, eta, ondorioz, errazagoa da akatsak detektatzea eta konkurrentzia errazten du. Kontuan hartuta aplikazioen konplexutasuna gero eta handiagoa dela, batez ere prozesatzen dituzten datuen kopuruagatik, beharrezkoa da makina batean baino gehiagoan aldi berean exekutatzeko moduko programak garatzea, konkurrentzia eta paralelismoa onartuta. Horregatik, paradigma hori indar handiz ari da suspertzen enpresa- eta industria-munduan. Hala ere, programazio funtzionala ez da erraza, errekurtsibitatea teknika konplexua delako, eta baliteke errore larriak egitea ondo erabiltzen jakin ezean. Bestalde, programak mantentzea zaila da eta kodea nekez berrerabil daiteke.



Eduki digitalak sortzea

C2 maila 3.4 Programazioa

Arazketa Python-en. Alderdi orokorrak

Arazketa Python-en. Alderdi orokorrak

Sarrera

Programazioan, helburu bat izaten da beti behar bezala eta akatsik gabe funtzionatzen duen kodea sortzeko beharra. Testuinguru horretan, arazketa programazio-prozesuaren funtsezko alderdi bihurtzen da. Arazketa prozedura sistematiko bat da, eta laguntzen du programa batean akatsak edo *bug*ak detektatzen, isolatzen eta zuzentzen. Eta era horretan ziurtatzen da bai behar bezala funtzionatzen dutela, bai errorerik gabeko kode bat sortzen dela.

Python-en eta beste edozein programazio-lengoaiatan programak arazteko trebetasuna funtsezkoa da garatzaile guztientzat. Behar bezala araztutako kodeak detektatu gabeko erroreak egotea minimizatzen du, programak exekutatu bitartean aurreikusi gabeko akatsak eta portaerak izatea prebenitzen du, eta aukera ematen die garatzaileei kodearen logika eta fluxua hobeto ulertzeko, eta horrek softwarearen mantentzea errazten du.

Python-en arazketa errazteko dauden tresnen artean, pdb da nabarmenetako bat. Pdb (*Python DeBugger*en akronimoa) Python lengoaian integratutako araztailea da. Araztaile horrek zenbait funtzionalitate baliotsu ematen ditu, programatzaileek haren kodea urratsez urrats ibil dezaten, aldagaiak ikuska ditzaten eta adierazpenak ebalua ditzaten.

Errore motak

Python-en programak araztean, litekeena da zenbait errore mota aurkitzea. Python-eko erroreak hiru kategoriatan sailkatzen dira: sintaxi-erroreak, salbuespenak eta errore logikoak.

1 | **Sintaxi-erroreak**: kodeak Python lengoaiaren sintaxi-arauak urratzen dituenean gertatzen dira. Era horretako erroreen adibideak, besteak beste: funtzio edo klase baten deklarazioaren amaieran bi puntu (:) jartzea ahaztea, koska-erroreak eta parentesiak eta giltzak ez jartzea. Python-ek sintaxi-errore bat aurkitzen duenean, eten egiten du kodea interpretatzeko prozesua, eta errore-mezu bat erakusten du, zeinak errorearen lerroa eta izaera adierazten dituen.

2 | Salbuespenak: programa exekutatzean gertatzen diren erroreak dira. Kodearen sintaxia zuzena izanagatik ere, baliteke programak errorea sortzea instrukzio bat exekutatzen saiatzean. Salbuespenetan askotariko egoerak daude, hala nola zenbaki bat zati zero egiten saiatzea, existitzen ez den fitxategi bat irekitzea eta definitu gabeko aldagai bat atzitzea. Python oso esplizitua da salbuespenak gertatzen direnean, eta pila-miaketa xehatua ematen du, barne hartuta salbuespena zer lerrotan gertatu zen eta zer salbuespen mota zen.

⚠ ADI

Python-eko pila-miaketa bat (*stack trace*) hauxe da: zure programak puntu jakin batera iritsi arte egin dituen funtzio-deien sekuentziaren adierazpide bat. Zure programa dagoen tokira nora iritsi den adierazten duen aztarna digital bat bezalakoa da; normalean, errore bat gertatzen denean agertzen da. Zure programak huts egiten badu (errore bat), Python-ek erakutsiko dizu zer bideri jarraitu dion eta zer funtzio eta metodo deitu ziren eta zer ordenatan, arazoa non sortu zen diagnostikatzen laguntzeko.

3 | Akats logikoak: hauek dira detektatzen eta zuzentzen zailenak. Errore logikoak gertatzen dira, baldin eta, programa baten logika edo diseinua okerra izanagatik, programa sintaxi-errorerik edo salbuespenik sortu gabe exekutatzen bada. Begizta infinitu bat da errore logikoen adibide komun bat. Begizta sintaktikoki zuzena izan badaiteke ere, begizta amaitzeko baldintza ez bada inoiz betetzen, programak mugarik gabe jarraituko du exekutatzen. Python-en salbuespen-sistemak ez ditu errore horiek atzematen; ondorioz, kodea arretaz berrikusi eta programaren logika funtsez ulertu beharra dago erroreok konponduko baditugu.

pdb araztailea

pdb araztailea funtsezko tresna da kodea Python-en garatzeko. Arazketa-prozesua errazten duten funtzionalitate asko ematen ditu, garatzaileek kodea araka dezaten, haustura-puntuak ezar ditzaten, urratsez urrats aurrera egin dezaten eta abar. Hemen, pdb-rekin arazten hasteko funtsezko kontzeptu batzuk berrikusiko ditugu.

1 | **Haustura-puntuak:** markatzaileak dira, eta jartzen ahal dira programaren exekuzioa geldiaraztea nahi duzun kode-lerro espezifiko batean. Hori baliagarria da baldin badakizu akats bat kodearen atal espezifiko batean gertatzen dela, baina ez bazaude ziur zehazki non eta zergatik gertatzen den. pdb-rekin, «set_trace()» funtzioa erabil dezakezu haustura-puntu bat ezartzeko.

2 | **Lerroz lerro exekutatzea:** programaren exekuzioa haustura-puntu batean gelditu denean, «step» funtzioa (edo «s» forma laburtuan) erabil dezakezu kodean lerroz lerro aurrera egiteko.

3 | **Aldagaiak ikuskatzea:** haustura-puntu batean geratzean edo lerroz lerro aurrera egitean, baliteke aldagai espezifiko baten balioa ikusi nahi izatea. pdb-rekin, horretarako aski duzu aldagaiaren izena pdb-ren kontsolan sartzea. «args» funtzioa ere erabil dezakezu egungo funtzioaren argumentuak ikusteko.

4 | **Betearazten jarraitzea:** exekuzioa haustura-puntu batean eten baduzu eta ondorioztatu baduzu puntu horretaraino dena behar bezala dabilela, «continue» funtzioa (edo «c» forma laburtuan) erabil dezakezu exekuzioari berriro ekiteko, hurrengo haustura-punturaino edo programaren amaieraraino.

5 | **pdb-tik irtetea:**edozein arrazoi dela medio, arazketa gelditu behar baduzu, «quit» funtzioa (edo «q» forma laburtuan) erabil dezakezu pdb-tik irteteko eta programaren exekuzioa amaitzeko.

Programa baten arazketaren adibidea

Abiapuntu hartuko dugu adibide-kode hau, zeinak zenbaki bat zati izendatzaile bat egiten duen eta, ondoren, izendatzailea 0-ra iritsi arte gutxitzen duen; horrek era honetako salbuespena ekarriko du: ZeroDivisionError.

```
def zatiketa_beherakorra(zenbakitzailea, izendatzailea):
  while izendatzailea >= 0:
  emaitza = zenbakitzailea / izendatzailea
    print(f"{zenbakitzailea} zati {izendatzailea} egitearen emaitza {emaitza} da")
    izendatzailea -= 1

zatiketa_beherakorra(10, 3)
```

Kodea arazteko, funtzioan haustura-puntu bat txertatuko dugu gure iritziz arazoa sortzen ari den lerroaren aurretik:

```
import pdb
def zatiketa_beherakorra(zenbakitzailea, izendatzailea):
  while izendatzailea >= 0:
    pdb.set_trace()
    emaitza = zenbakitzailea / izendatzailea
    print(f"{zenbakitzailea} zati {izendatzailea} egitearen emaitza {emaitza} da")

    izendatzailea -= 1

zatiketa_beherakorra(10, 3)
```

Python-en kontsolako arazketaren simulazio posible bat honako hau izan liteke:

```
> python3 nire_programa.py
> /bidea/a/nire_programa.py(6)zatiketa_beherakorra()
-> emaitza = zenbakitzailea / izendatzailea
(Pdb) p izendatzailea
3
(Pdb) c
10 zati 3 eginda, emaitza 3.3333333333333335 da.
> /bidea/a/nire_programa.py(6)zatiketa_beherakorra()
-> emaitza = zenbakitzailea / izendatzailea
(Pdb) p izendatzailea
2
(Pdb) c
10 zati 2 eginda, emaitza 5.0 da.
> /bidea/a/nire_programa.py(6)zatiketa_beherakorra()
-> emaitza = zenbakitzailea / izendatzailea
(Pdb) p izendatzailea
1
(Pdb) c
10 zati 1 eginda, emaitza 10.0 da.
> /bidea/a/nire_programa.py(6)zatiketa_beherakorra()
-> emaitza = zenbakitzailea / izendatzailea
(Pdb) p izendatzailea
0
(Pdb) s
ZeroDivisionError: division by zero
```

Simulazio honetan, «p» komandoa erabiliko dugu begiztaren iterazio bakoitzean izendatzailearen balioa ikuskatzeko. «c» komandoa ere erabiliko dugu exekuzioak hurrengo haustura-punturaino jarraitzeko (begiztaren barruan dago); beraz, iterazio bakoitzean gelditzen da. Azkenik, izendatzailea 0 dela ikusten dugunean, «s» komandoa erabiliko dugu hurrengo lerrora aurrera egiteko, eta horrek ZeroDivisionError sortzen du. Arazketaren bidez, baliteke programatzailea ohartzea errorea zer lerrotan gertatzen ari den eta zer egoerak eragiten duen (zenbakitzailearen eta izendatzailearen balio zehatzak).

Informazio gehiago

Python-eko arazketari buruzko ezagutza handitzeko aukera duzu Python-en gaztelaniazko dokumentazio ofizialaren bitartez, hemen: *e.digitall.org.es/depuracion*

Eduki digitalak sortzea

C2 maila 3.4 Programazioa

Python-en kodea diseinatzea. Jardunbide egokiak

Python-en kodea diseinatzea. Jardunbide egokiak

Sarrera

Kode eraginkor eta efizientea diseinatzea da programatzaile guztiek aurre egin beharreko erronka nagusietako bat. Ildo horretan, Python-ek, sinpletasunean eta irakurgarritasunean oinarritutako ikuspegiari esker, ingurune ezin hobea ematen du garatutako kodea funtzionala ez ezik, erraza ulertzen eta mantentzen ere izan dadin. Kodea diseinatzeko jardunbide egokiak hartzea funtsezkoa da, softwarearen kalitatea hobetzeko, eta horrez gain, baita lankidetza eta kodea epe luzerako mantentzea errazteko ere. Izan ere, jardunbide egokiei jarraitzea epe luzeko inbertsiotzat hartu behar da. Hasieran, baliteke denbora eta ahalegin gehiago behar izatea, baina, zalantzarik gabe, merezi du epe luzera, arazoak gutxitu eta eraginkortasuna handitzen duelako.

Jardunbide egokiak aplikatzeak kodearen konplexutasuna murrizten du; haren mantengarritasuna areagotzen du, eta akatsak hautematea eta zuzentzea errazten du; horrek softwarearen kalitatea hobetzea dakar.

Python-en kodea diseinatzeko jardunbide egokiak

Jarraian, programak diseinatzean eta kodetzean kontuan hartu beharreko jardunbide egoki ohikoenetako batzuk aztertuko ditugu.

Izendatze deskribatzailea

Aldagaien, funtzioen eta klaseen izenek behar bezain deskribatzaileak izan behar dute, programan zer helburu edo erabilera duten adierazteko. Izenek ulergarriak izan behar dute, eta ideia eman behar dute funtzioak zer egiten duen edo aldagaiak zer ordezkatzen duen. Praktika horrek kodea irakurtzea eta ulertzea errazten du, programatzaileak berak ez ezik, baita kodea lantzen edo berrikusten duten beste garatzaile batzuek ere.

Adibidez, demagun funtzio bat idazten ari zarela zirkulu baten area kalkulatzeko. Funtzioa modu generikoan func1 gisa aipatu beharrean eta x irrati-aldagairako erabili beharrean, izen deskribatzaileagoak erabili:

```
#Izendapen ez oso deskribatzailea

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

```
#Adibide bera aldagai deskribatzaileen izendapenarekin

def kalkulatu_zirkulu_area (erradioa):
  return 3.1416 * (erradioa**2)
```

Iruzkinak egoki erabiltzea

Iruzkinak tresna erabilgarria dira kode-atal baten xedea edo funtzionaltasuna azaltzeko, bereziki konplexua bada edo kodeak berak argi adierazten ez badu. Besterik gabe kodea irakurri eta berehala agerikoak ez diren xehetasunak gehitzeko erabili behar dira. Alabaina, garrantzitsua da kodea alferrikako iruzkinekin asko ez kargatzea, horrek ekarriko baitu kodea irakurtzen zailagoa izatea.

Adibidea: Demagun zenbaki-zerrenda bat burbuilaren metodoa erabiliz ordenatzen duen algoritmo bat ezartzen ari zarela. Hona hemen kode hori behar bezala komentatzeko modua:

```
def ordenamendua_burbuila (zerrenda):
  #Zerrendako elementu bakoitzaren gainean iteratzea
  for i in range(len(zerrenda)):
    # Oraingo elementua hurrengoarekin alderatuko dugu
    for j in range(0, len(zerrenda) - i - 1):
      # Oraingo elementua hurrengoa baino handiagoa bada,
      trukatzen ditugu
      if zerrenda[j] > zerrenda[j + 1] :
        zerrenda[j], zerrenda[j + 1] = zerrenda[j + 1], zerrenda[j]
```

Erantzukizun bakarraren printzipioa aplikatzea

Printzipio horrek iradokitzen du funtzio, modulu eta klase bakoitzak erantzukizun bakarra izan behar duela. Modu sinpleago batez esanda, gauza bakarra egin beharko lukete, baina ondo egin. Printzipio hori betez gero, kodea erabilerrazagoa, mantentzen errazagoa eta akatsetarako joera txikiagokoa izango da, arazorik sortzen bada, zehazki baitakizu non bilatu.

Adibidea: Demagun e-commerce baten eskaerak prozesatzen dituen aplikazio bat duzula.

```
def kudeatu_eskaera(bezeroa, produktua, kopurua):
  # Egiaztatu produktuaren eskuragarritasuna
  ...
  # Eguneratu inbentarioa
  ...
  # Prozesatu ordainketa
  ...
  # Sortu faktura
  ...
  # Bidali berrespen-mezua bezeroari
  ...
```

Eskaera-prozesu osoa kudeatzen duen funtzio bat izan beharrean, hobe da prozesuaren urrats bakoitzerako funtzio bereiziak izatea.

```
def egiaztatu_eskuragarritasuna (produktua, kantitatea):
  ...
def eguneratu_inbentarioa(produktua, kopurua):
  ...
def prozesatu_ordainketa(bezeroa, zenbatekoa):
  ...
def sortu_faktura(bezeroa, eskaera_xehetasunak):
  ...
def bidali_berrespena_postaz(bezeroa, eskaera_xehetasunak):
  ...
```

Funtzioak erabiltzea eta abstrakzioa

Abstrakzioa programazioko funtsezko metodoa da, eta kodearen zati espezifiko baten xehetasun teknikoak haren aplikazio praktikotik urruntzea du oinarritzat. Python lengoaian, funtzioak erabil daitezke eragiketak sinplifikatzeko eta helburu jakin bat betetzen duten kode-atalak biltzeko. Funtzio bidez abstrakzioa erabiltzeak kodearen irakurgarritasuna optimizatzen du, baita berrerabiltzeko aukera ere, eta hura mantentzea errazten du.

Adibidez, kalkulu matematiko konplexuak zenbait aldiz egiten dituen programa bat sortzen ari bazara, behin eta berriz kalkulu-logika bera idatzi beharrean, logika hori funtzio batean abstraitu dezakezu. Orduan, kode bera errepikatu beharrean, aski izango da funtzioa deitzea kalkulu hori egin behar duzun bakoitzean. Horrela, kalkulua egiteko modua aldatu behar baduzu, leku batean baino ez duzu egin beharko (funtzioaren barruan), kode bereko instantzia ugari bilatu eta aldatu ordez.

OHARRA

Python-en funtzioak erabiltzen direnean, aholku ona da printzipio honi jarraitzea: «gutxiago gehiago da». Funtzioak txikiak izan behar dira eta gauza bakarra egin behar dute. Ikusten baduzu funtzio bat asko hazten ari dela edo gauza gehiegi egiten hasi dela, funtzio txikiagotan zatitzeko unea izango da seguruenik.

Erroreak eta salbuespenak kudeatzea

Programa ona izango da sendoa bada eta erroreak eta ustekabeko egoerak kudea baditzake. Python-en erroreak eta salbuespenak kudeatzeak aukera ematen die programei erroreak kudeatzeko eta exekutatzen jarraitzeko, baita ezustekorik gertatzen bada ere. Programatzaileek zehaztu dezakete zer gertatu beharko litzatekeen errore partikular baten kasuan salbuespen bat gertatzen bada.

Adibidea: Pentsatu bi zenbaki zatitzen dituen funtzio bat duzula. Zati zero egiten saiatuz gero programak huts egin dezan utzi beharrean, egoera hori honela atzeman eta kudea dezakezu:

```
def zatitu(zenbakitzailea, izendatzailea):
  try:
    return zenbakitzailea / izendatzailea
  except ZeroDivisionError:
    print("Errorea: Ezin da zati zero egin.")
    return None
```

Adibide honetan, zati zero egiten saiatzen bazara, programak ZeroDivisionError salbuespena atzematen du, errore-mezu bat ematen du eta None itzultzen du. Horrela, programa ez da gelditzen eta exekutatzen jarraitzen du, eragiketa baliogabe bat egiten saiatu izanagatik ere. Horren ondorioz, zure programa sendoagoa da, eta atseginagoa erabiltzailearentzat.

Saihestu kode erredundantea

Softwarea garatzeko funtsezko printzipioa da; haren arabera funtzionalitate oro toki bakar batean ezarri behar da. Kode bera behin baino gehiagotan idazten ari zarela ikusten baduzu, horrek esan nahi du funtzionalitate hori funtzio edo klase batean kapsulatu eta berrerabili behar duzula.

Python-en estilo-konbentzioak erabiltzea (PEP 8)

Python Enhancement Proposal 8 estilo-gidaliburua, PEP 8 izenez ezaguna, Python kodea formateatzen jakiteko gomendio-bilduma bat da. Konbentzio horiei jarraitzeak kodearen sendotasunari eusten eta irakurgarritasuna hobetzen laguntzen du. Arau horietako batzuk itxuraz arbitrarioak badira ere, horiei atxikiz gero, errazagoa izango da beste batzuentzat (eta zeuretzat) zure kodea irakurtzea eta ulertzea.

Hemen duzu eskuragai gidaliburua gaztelaniaz: ***e.digitall.org.es/pep8***

Probatu beti zure kodea behar bezala dabilela

Proba unitarioak programazio osasungarriaren funtsezko osagai bat dira. Aukera ematen diote programatzaileari kode-pieza indibidualak (edo «unitateak») egoera eta sarrera desberdinetan behar bezala funtzionatzen ari direla egiaztatzeko. Python-en, **unittest** modulua tresna ahaltsua da proba horiek egiteko.

Zure koderako proba unitarioak idaztea lagungarria izango da erroreak azkarrago identifikatzeko eta zuzentzeko, kodearen kalitatea hobetzeko, eta kodea etorkizunean aldatu eta zabaltzea errazteko.

Adibidea: Demagun zenbaki baten faktoriala kalkulatzen duen funtzio hau duzula:

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

Proba unitario hau idatz dezakezu, funtzioak behar bezala funtzionatzen duela ziurtatzeko:

```
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()
```

TestFactorial klaseak test_faktoriala izeneko metodoa du, zeinak zenbait test egiten baititu funtzio faktorialaren gainean. Funtzioaren emaitza sarrera-balio batzuetarako egokia dela egiaztatzen da. Test horietakoren batek huts egiten badu, unittestest.main() funtzioak akatsaren berri emango digu, eta orduan, aukera izango dugu arazoa funtzio **faktorialean** zuzentzeko.

Diseinu-patroiak erabiltzea

Diseinu-patroiak softwarea diseinatzean dauden ohiko arazoei emandako irtenbide frogatuak dira. Ohiko egoerei aurre egiteko esparru bat ematen dute, eta kodea hobeto antolatzea ez ezik, kodea mantentzea eta etorkizunean zabaltzea ere ahalbidetzen dute. Diseinu-patroiak erabiltzeak lagunduko dizu kode garbiagoa, modularragoa eta efizienteagoa idazten.

Informazio gehiago

Diseinu-patroiei buruz gehiago jakiteko, gomendagarria honako hau irakurtzea: Gamma, E: *Patrones de diseño*, Espainia: Pearson Educación, 2002.

Dokumentatu kodea

Kodean iruzkinak eta *docstring*ak egiteaz gain, garrantzitsua ere bada dokumentazio zabalagoa eta osoagoa ematea. Horren barnean sartuko dira liburutegi edo moduluetarako APIaren dokumentazioa, prometarako erabiltzailearen eskuliburuak eta softwarea instalatzeko eta konfiguratzeko gidaliburuak. Dokumentazioa funtsezkoa da proiektu bat mantentzeko eta eskalatzeko, baita beste garatzaile batzuekin lankidetzan aritzeko ere.

Kodearen dokumentazio ondo egingo dugu, besteak beste, era honetan: README fitxategiekin, kode-gordailuko wikieklin eta proiektuaren dokumentazio osoa jasotzen duten web-orriekin.

Informazio gehiago

Jardunbide egokiez gain, askoz gehiago ere badaude. Horregatik, liburu hauek irakurtzea gomendatzen dizugu:

Clean Code: Handbook of Agile Software Craftsmanship (Robert C. Martin): Liburu hau Python-erako berariaz idatzita ez badago ere, ematen dituen irakaspenak eta printzipioak aplikagarriak dira programazio-lengoaia guztietan. Liburuak aztertzen du kalitate handiko kodea idazteko modua; alegia, erraza irakurtzen, mantentzen eta zabaltzen den kodea. Lagunduko dizu ulertzen profesionalek nola pentsatzen duten softwarea diseinatzeaz, eta gauzak zergatik egiten dituzten egiten dituzten moduan.

Fluent Python: Clear, Concise, and Effective Programming (Luciano Ramalho): Liburu hau baliabide bikaina da Python-eko programatzaile ertain eta aurreratuentzat. Lengoaiaren ezaugarriak lantzeaz gain, gauzak egiteko «modu pythoniko»ei buruzko ikuspegiak ere ematen ditu. Liburu honen bidez, Python kodea idazten ikasiko duzu, hau da, kode idiomatikoagoa, efizienteagoa eta argiagoa da.

Eduki digitalak sortzea

C2 maila 3.4 Programazioa

Salbuespenak erabiltzea Python-en

Salbuespenak erabiltzea Python-en

Salbuespenak Python-en

Python programazio-lengoaiak bi errore nagusi bereizten ditu: i) sintaxi-erroreak eta ii) salbuespenak. Lehenengoak gertatzen dira lengoaiaren sintaxia erabiltzean eginiko eraikuntza okerren ondorioz. Hurrengo zerrendak adibide sinple bat erakusten du; bertan ikusten da Python-en interpretatzaileak balizko erroretik hurbil dagoen posizio bat *adierazten* duela (kasu honetan, : karakterea falta dela «while True»ren ondoren). Bigarrenak, aurretik adierazi den bezala, **programa bat exekutatzean izaten diren gertaeren** ondorioz dira, eta nahi gabe aldatzen dute haren exekuzio-fluxu *estandarra*.

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

Python-en salbuespen bat gertatzen denean, hura kontrolatu ezean, interpretatzaileak abiarazitako salbuespenaren izena erakutsiko du. Adibidez, zerrenda honek ZeroDivisionError salbuespena erakusten du, zeina lengoaiak berak definitzen baitu zati 0 egiten saiatzen garenean.

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

Zorionez, salbuespenak modu egokian erabiltzen dituzten Python-eko programak idatz daitezke. Horretarako, try **sententzia** erabil daiteke. Sententzia horrek honela funtzionatzen du:

1 | Lehenik eta behin, try klausula exekutatzen da (try eta except gako-hitzen arteko sententziak).
2 | Salbuespenik ez badago, except klausula saltatzen da eta bukatu egiten da try sententziaren exekuzioa.
3 | try klausula exekutatzean salbuespenik gertatzen bada, klausularen gainerakoa saltatzen da. Hortaz, haren mota bat badator except gako-hitzaren ondoren adierazitako salbuespenarekin except klausula exekutatzen da, eta exekuzioak jarraituko du try/except blokearen ondoren.

4 | Salbuespen bat gertatzen bada eta ez badator bat except klausulan adierazitako salbuespenarekin, kanpoko try sententzietara igarotzen da. Salbuespenerako kudeatzaile bat aurkitzen ez bada, orduan kudeatu gabeko salbuespena da eta exekuzioa aurreko zerrendakoa bezalako mezu batekin gelditzen da.

try sententziak baliteke except klausula bat baino gehiago edukitzea, salbuespen desberdinetarako kudeatzaileak zehazteko. Bestalde, try sententziak aukerako else klausula bat dauka, eta, agertzen denean, except klausula guztiei jarraitu behar die. Klausula hau (else) erabilgarria da try klausulak salbuespenik abiarazi ezean exekutatu beharreko kodea jasotzeko. Hurrengo kode-zatian adibide bat dugu; bertan fitxategi bat ireki nahi da, eta haren izena nire_fitxategian gordetzen da. Fitxategia ez bada existitzen, OSError motako salbuespena abiaraziko da, except klausulan atzemandakoa. Fitxategia existitzen bada, orduan, adibide horretan, else klausularen kodea exekutatuko da, eta bertan fitxategiak duen lerro kopurua inprimatzen da.

```
try:
  f = open(nire_fitxategia, "r")
except OSError:
  print("Ezin izan da ireki", nire_fitxategia)
else:
  print(nire_fitxategia, "dauka", len(f.readlines()), "lerroak.")
  f.close()
```

Posible da **erabiltzaile-salbuespenak** definitzea; hau da, salbuespenotan semantika lotuta dago arazo jakin baten konponbidea adierazten duen kodearekin. Erabiltzaileak zehaztutako salbuespen-adibide bat, lehen ere aipatutakoa, hau izan liteke: 9 digitu zehazki ez dituen telefono-zenbaki bat gordetzen saiatzea. Hurrengo zerrendak adibidearen kode adierazgarria erakusten du. Bertan, Python-en klase berri gisa definitutako ZenbakiFormatuBalioGabeaException salbuespena sortu dela ikusten da. Klase horrek Exception klase generikotik heredatzen du.

```
class ZenbakiFormatuBalioGabeaException (Exception):
  pass

numero = input("Sartu 9 digituko telefono-zenbaki bat...")
if len(zenbakia) != 9:
  raise ZenbakiFormatuBalioGabeaException
```

Aurreko adibideak erakusten du raise sententzia nola erabili salbuespen bat abiarazteko. Kasu horretan, pentsa liteke kodearen goiko geruzak abiarazitako salbuespena atzemateko eta tratatzeko beharrezkoa den logika barne hartuko lukeela.

Atal hau amaitzeko, finally klausula aipatuko dugu. Funtsean, finally klausula bat badago, azken ataza gisa exekutatuko da try sententzia amaitu baino lehen. finally klausula exekutatzen da try sententziak salbuespena sortu edo ez. Hurrengo adibideak kode-zati osoagoa erakusten du.

```
def zatitu(a, b):
  try:
    emaitza = a / b
  except ZeroDivisionError:
    print("Zati zero egitea.")
  else:
    print("Emaitza da", emaitza)
  finally:
    print("Finally klausula exekutatzen")
```

Eduki digitalak sortzea

C2 maila 3.4 Programazioa

Kode-probak Python-en

Kode-probak Python-en

Sarrera

Probak funtsezkoak dira aplikazioak garatzeko prozesuan. Haiei esker, kodearen funtzionaltasun zuzena egiaztatu eta haren kalitatea berma daiteke. Haien garrantziaren adibide argia Proben bidez Gidatutako Garapena (TDD) da.

Dokumentu honetan, proba unitarioak nola idatzi aztertuko da, zehazki, Python-en garatutako aplikazioetarako. Jarraitu aurretik, komeni da ***A3C34C1D06*** dokumentazioa berrikustea.

KODE-PROBAK. FUNTSEZKO ALDERDIAK

Kode-probetarako eta TDD ikuspegirako sarrera-dokumentua. Erreferentziazko dokumentua: ***A3C34C1D06***

Proba motak. Proba unitarioak

Aplikazioak garatzean erabiltzen diren probak askotarikoak dira, eta batzuk edo beste batzuk erabiliko dira kontuan hartuta haien bitartez zer egiaztatu nahi dugun. Hauek dira gehien erabiltzen diren probak: proba unitarioak, integrazio-probak, proba funtzionalak eta erregresio-probak. Dokumentu honek lehenengoetan jartzen du arreta, hots, proba unitarioetan.

Proba unitarioak kode-unitate indibidualek behar bezala funtzionatzen dutela egiaztatzeko proba fokalizatuak dira. Hau da, ziurtatzea funtzioek, metodoek eta klaseek modu isolatuan behar bezala funtzionatzen dutela.

Python-eko hainbat tresna eta framework-ek proba unitarioak sortzea eta gauzatzea errazten dute. Python-en gehien erabiltzen diren liburutegietako batzuk unittest, pytest eta nose dira.

Python-eko probak: unittest

Unittest da Python-en liburutegi estandarrean integratutako proba-frameworka da. Zenbait modulu eta tresna ematen ditu kodearen funtzioak egiaztatzen laguntzeko eta aplikazioen garapena errazteko.

Adibide praktikoa

Atal honetan, Python-eko proba unitarioen adibide praktiko bat ikusiko dugu. Zehazki, kalkulagailu gisa funtzionatzen duen aplikazio bat garatu da. Klase bat da, eta bertan batuketa, kenketa, biderketa eta zatiketako metodoak ezarri dira:

```
class Kalkulagailua:
  def batuketa(self, a, b):
    return a + b
  def kenketa(self, a, b):
    return a - b
  def biderketa(self, a, b):
    return a * b
  def zatiketa(self, a, b):
    if b == 0:
      raise ValueError ("Errorea: ezin da zati zero zatitu")
    return a / b
```

Probak ezarri aurretik, probatu beharreko egoerak aztertu behar dira. Adibidez, kalkulagailuaren kasuan, beharrezkoa izango da egiaztatzea bi zenbaki positiboren batuketa, bi zenbaki negatiboren kenketa, zati zero egitea eta abar.

Ezarri beharreko kasuak aztertu ondoren, kodetu egiten dira. Kasu honetan, unnitest-ekin, urrats hauek egingo ditugu:

- KalkulagailuTesta izeneko proba klase bat sortu, unittest.TestCasetik heredatzen duena.
- Probatu beharreko modulua (kalkulagailua) inportatu.
- Sortu setUp metodoa, proba bakoitzaren aurretik kalkulagailuaren instantzia bat hasieratzeko.
- Askotariko exekuzio-aukerak barne hartzen dituzten metodoak definitu. Metodo bakoitzean era honetako asertzioak erabiliko ditugu: assertEquals, assertNoEquals, assertTrue edo assertFalse, funtzionaltasuna egiaztatzeko.
- Metodoen izenak test_arekin hasi behar du.

Kode honek Kalkulagailua klasea barne hartzeko proba unitarioen adibide bat erakusten du.

```
import unittest
from Kalkulagailua import Kalkulagailua
class KalkulagailuTesta(unittest.TestCase):
  def setUp(self):
    self.kalkulagailua = Kalkulagailua()

  def batuketa_testa(self):
    result = self.kalkulagailua.batuketa (-1, 3)
    self.assertEqual(result, 2)

  def test_subtraction(self):
    result = self.kalkulagailua.kenketa (10, 4)
    self.assertFalse(result!= 6)

  def test_multiplication(self):
    result = self.kalkulagailua.biderketa (0, 5)
    self.assertTrue(result == 0)

  def test_division(self):
    result = self.kalkulagailua.zatiketa (80, 10)
    self.assertNotEqual(result, 80)

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

Kasu honetan, proba unitario bat ezarri da Kalkulagailua klasearen metodo bakoitzeko. Hala ere, horietako bakoitzaren exekuzio-egoerak barne hartzea litzateke zuzena.

Laburbilduz, dokumentu honetan aztertu dugu probek zer-nolako garrantzia duten aplikazioak garatzean. Zehazki, adibide praktiko batean ikusi dugu proba unitarioak nola sortzen diren Python-en liburutegi estandarreko framework-arekin (unittest). Garrantzitsua da kontuan hartzea proben mundua zabala eta konplexua dela, eta proba unitarioak software-proben panorama osoaren zati bat baino ez direla.

OHARRA

Unitate-probak arinago sortu eta egiaztatzeko, gomendagarria da Python-i euskarria ematen dioten IDE batzuk erabiltzea. Aurreko gaietan Visual Studio Coderen ingurunea ikusi genuenez, interesgarria da ezagutzak areagotzea IDEtik erabil daitezkeen Python-eko proba-liburutegiak erabiliz.

Informazio gehiago

Jarraian, Python-en garatutako aplikazioetarako proba-framework nagusietako batzuk zerrendatu ditugu.

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

DigitAll

Gaitasun
digitaletan
prestakuntza

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