[3] José Ramón Barroso Rosendo i Santiago Saborido Piñero. *Antonio Hugo de Omerique - estudio crítico*. Fundación Ignacio Larramendi, 2018.

[4] Joaquim Berenguer Clarià. "Wendlingen: a scientist in the eighteenth century spanish court." Dins de: *The algebrization of mathematics during the 17th and 18th centuries*, Davide Crippa, Maria Rosa Massa-Esteve (eds.). Dialogues and Games of Logic (8), 2023.

[5] Henk J. M. Bos. *Redefining Geometrical Exactness: Descartes' Transformation of the Early Modern Concept of Construction*. Sources and Studies in the History of Mathematics and Physical Sciences. Springer, 2012 (edició original 2001).

[6] Albert Dou. "Las matemáticas en la España de los Austrias". Dins de: L.Español González (coord.), *Estudios sobre Julio Rey Pastor (1888-1962)*. Instituto de Estudios Riojanos, 1990.

[7] F. Gómez [et al.]. "The six books of Diophantus' Arithmetic increased and reduced to specious: the lost manuscript of Jacques Ozanam (1640–1718)". Archive for history of exact sciences, 1 Gener 2021, vol. 75, p. 557-611.

[8] Maria Rosa Massa-Esteve. "The role of symbolic language in the transformation of mathematics". Philosophica 87 (2012) pp. 153-193.

[9] Maria Rosa Massa-Esteve. "Viète i la nova àlgebra". SCM/Notícies, núm. 48 (2021), pp.71-76.

[10] Maria Rosa Massa-Esteve. "Niccolò Tartaglia matemàtic i enginyer del Renaixement". SCM/Notícies, núm. 50 (2022), pp.80-86.

[11] Vicente Meavilla i Antonio M. Oller-Marcén. "Ejemplos de análisis-síntesis en un contexto geométrico. El 'Analysis Geometrica' de Antonio Hugo de Omerique". Matemáticas, Educación y Sociedad, 2(1) (2019), pp.29-39.

[12] Martín Fernández de Navarrete. *Biblioteca marítima española, 1*. Imprenta de la Viuda de Calero, 1851.

[13] Víctor Navarro Brotons. "La renovación de la actividad científica en la España del siglo XVII y las disciplinas físico-matemáticas". Dins de: *El siglo de las Luces. De la ingeniería a la nueva navegación*, Manuel Silva Suárez, ed., 2005.

[14] Juan Navarro-Loidi, José Llombart. "The introduction of logarithms into Spain". Historia Mathematica 35 (2008), p. 83-101.

[15] Antonio Hugo de Omerique. *Analysis geometrica sive Nova, et vera methodas resolvendi tam problemata geometrica quam arithmeticas quaestiones*. Cádiz, 1698.

[16] Antonio D. Hugone *Analysis geometrica, five nova & vera methodus resolvendi, tam problemata geometrica, quam arithmeticas quæstiones. Pars prima, de planis*. Phil. Trans. R. Soc. 21: 351-362.

[17] Jean Pelseneer. "Une opinion inédite de Newton sur 'l'Analyse des Anciens' à propos de l'Analysis geometrica de Hugo de Omerique". Isis, vol.14, 1 (1930).

[18] Fàtima Romero Vallhonesta. *L'àlgebra a la Península Ibèrica del segle XVI* (tesi doctoral). CEHIC - UAB, 2018. `http://hdl.handle.net/10803/650339`

[19] José A. Sánchez Pérez. "La matemática", dins de *Estudios sobre la ciencia española del siglo XVII*. Asociación de Historiadores de la Ciencia Española. Madrid, 1935.

[20] Pierre Vilar. *La historia de España*. Ed. Grijalbo, 1978.

Bits de matemàtiques

Programant amb Julia

Laura Brustenga i Moncusí
Odí Soler i Gibert

En aquest número del *SCM/Notícies* us volem presentar *Julia*. Quan parlem de *Julia*, ens referim a un llenguatge de programació i no pas a una persona amb aquest nom. Aquest llenguatge permet treballar amb un intèrpret d'ordres per fer proves senzilles i, a la vegada, escriure programes computacionalment eficients. Aquí no entrarem a valorar si *Julia* és millor o pitjor que alternatives amb capacitats similars com ara Python, C o R. Només explicarem algunes

semblances i diferències amb aquests altres llenguatges, que potser són més coneguts.

Com sempre, animem els lectors a fer-nos arribar propostes per tractar en aquesta secció a les adreces de correu electrònic brust@mat.uab.cat o odisoler@mat.uab.cat. En aquesta ocasió, també convidem els lectors que ja coneguin *Julia* a fer-nos arribar les seves opinions sobre quins llenguatges són millors per a cada objectiu.

El problema dels dos llenguatges

Hi ha moltes raons per les quals ens podem posar a programar com a professionals de les matemàtiques: per fer un càlcul numèric, per analitzar un conjunt de dades, per fer quatre proves *d'estar per casa* per posar a prova la nostra intuïció o perquè ens toca ensenyar a programar.

En alguns casos, com quan fem proves per experimentar, només volem executar unes quantes comandes sense haver de *fer* un programa complet. Ens cal un llenguatge senzill d'entendre i d'executar. Volem poder executar una única comanda i veure'n els resultats al moment. Sobretot, no volem haver de *compilar* un programa que després haurem d'executar. És a dir, no volem haver de traduir les instruccions que tenim al cap a instruccions en el "llenguatge de la màquina" i posteriorment haver d'executar el nou fitxer que obtenim amb aquest procés.

En aquesta situació, molta gent opta per fer servir un *llenguatge interpretat*, com ara Python. És a dir, fem servir un programa anomenat *intèrpret* en què introduïm ordres senzilles que s'executen al moment. Per exemple, si escrivim `sin(pi/4)` (després d'importar les funcions necessàries), l'intèrpret ens dona el valor d'aquesta expressió quan premem la tecla Enter. Els llenguatges interpretats també inclouen estructures més complexes, com ara condicionals (`if`) i bucles (`for`). A més, solen tenir moltes llibreries públiques amb funcions útils per fer tasques habituals, com per exemple tractar cadenes de caràcters o fer gràfics de dades.

Si bé és cert que amb els llenguatges interpretats es poden escriure programes en fitxers anomenats *scripts*, aquests solen ser força lents i ineficients. En aquest punt és on entra el segon llenguatge. Per a projectes que suposen una càrrega molt intensa per a l'ordinador, molta gent recorre a un *llenguatge compilat*, com ara C. És a dir, en aquest cas sí que preparem un fitxer amb les instruccions que volem executar traduïdes al llenguatge propi de la màquina (compilem). Això ho fem perquè, encara que per tasques simples la diferència entre un tipus de llenguatge i l'altre és inapreciable, en projectes més grans la diferència pot ser molt rellevant. Per exemple, amb un programa ben optimitzat amb C podríem estalviar-nos hores, i fins i tot dies, d'execució en càlculs molt intensos!

I tot això, què té a veure amb Julia? Doncs bé, una de les idees dels creadors d'aquest llenguatge era tenir un llenguatge interpretat, per ajudar la intuïció en tasques simples, però que també es pogués compilar i optimitzar amb relativa facilitat, per poder-lo fer servir igualment en projectes amb una càrrega computacional considerable. A més a més, aquesta versatilitat quant a les seves aplicacions és el que fa Julia especialment atractiu per a la docència. El fet que sigui un llenguatge interpretat permet aprendre'l tot experimentant sense complicacions i, a la vegada, aquestes mateixes bases serveixen més enllà del joc amb la línia d'ordres.

Característiques de Julia

La primera característica important a destacar de Julia és que és un projecte de *programari obert*. Això vol dir que es pot descarregar i utilitzar de forma gratuïta, sense haver de pagar per la seva llicència. Tot i això, ser programari obert vol dir molt més que ser gratuït (avui dia hi ha nombrosos exemples de programes que són gratuïts però no oberts). Entre altres coses, en ser programari lliure tothom pot accedir al seu codi font, examinar-lo i fins i tot fer propostes per contribuir a millorar-lo. De fet, aquest projecte es manté de manera comunitària. És a dir, hi ha una comunitat de gent que milloren el codi base del llenguatge, creen i mantenen paquets públics per a tasques especialitzades i preparen documentació per aprendre a utilitzar-lo. I el millor de tot és que qualsevol pot interactuar amb aquesta

comunitat, unir-s'hi i implicar-s'hi en la mesura que vulgui.

Pel que fa a la part més tècnica, Julia es va dissenyar tenint present la computació d'alt rendiment, cosa que el fa molt atractiu per a la computació científica. En concret, es pot combinar Julia amb altres eines existents que permeten executar processos en paral·lel per aprofitar les característiques tant dels processadors com de les targetes gràfiques actuals.

Un altre aspecte interessant de Julia és l'ús del paradigma de *dispatch múltiple*. En paraules més entenedores, aquest llenguatge ens permet definir una mateixa funció diverses vegades, però cada nova definició ha de prendre uns arguments diferents, sigui en nombre o en tipus. Per exemple, l'operació producte executa instruccions diferents quan es multipliquen enters i quan es multipliquen nombres decimals. A més, el símbol `*` també s'utilitza per concatenar cadenes. Això vol dir que l'operació producte té una altra definició en què els seus dos arguments són cadenes de caràcters. En el moment en què s'executa un programa, es determina quina de les definicions d'una funció cal fer servir, segons quins arguments acompanyin la crida. Per exemple, podem estendre la funció suma per concatenar cadenes intercalant un espai:

```
import Base:+ # Per a poder estendre
function +(s1::String, s2::String)
    return s1*" "*s2
end
```

Ara, la crida `"abc" + "def"` retorna `"abc def"`.

Per acabar, volem parlar del gestor de paquets de Julia. Abans hem esmentat que aquest llenguatge de programació té el suport d'una comunitat molt àmplia que el millora a cada moment. Moltes d'aquestes millores es posen a disposició pública en forma de paquets que contenen llibreries amb funcions per a tota mena de tasques: funcions matemàtiques de qualsevol àrea, llibreries per al tractament i visualització de dades, eines d'aprenentatge automàtic, etc. Julia incorpora un gestor de paquets per tal d'instal·lar, actualitzar i eliminar els paquets que ens interessi d'una manera còmoda i ordenada.

Estenent Julia, algunes llibreries

Julia ve amb multitud de funcions i objectes ja definits. Com `Strings`, nombres `Int` o `Float`, amb les seves versions de 16, 32 i 64 bits, `Big` per precisió arbitrària... Com a llenguatge d'ús general, incorpora una àmplia gamma de funcions, però a la vegada, les funcions estan limitades a ser d'ús general. Així, per accedir a funcions com `rank`, per calcular el rang d'una matriu, o `norm`, per la norma d'un vector (o qualsevol objecte iterable), hem de carregar el paquet `LinearAlgebra.jl` amb la comanda

```
using LinearAlgebra
```

La comunitat manté el repositori `www.juliahub.com`, amb més de 9.500 paquets per estendre les capacitats de Julia. En cas de voler carregar un paquet que no tenim instal·lat, la mateixa comanda `using` que hem vist fa un moment ens dona l'opció d'afegir-lo al nostre sistema.

Hi ha molts altres exemples de paquets interessants en el repositori públic, com ara `Plots.jl`, que és la llibreria estàndard per la representació gràfica de dades gràcies a la seva gran versatilitat. Un altre de semblant és `Makie.jl`, per fer gràfiques d'alta qualitat. El paquet `IJulia.jl` permet la interacció amb quaderns Jupyter. Per aplicacions específicament matemàtiques hi ha `OSCAR.jl`, per a les manipulacions algebraiques en un sentit ampli, i `SciML`, per al càlcul numèric. Tots aquests paquets i molts més espremen l'eficiència de Julia a la vegada que ofereixen una interfície còmoda. La veritat és que podríem dedicar un Bit a cadascun d'ells.

Julia a l'ensenyança

El problema dels dos llenguatges també el trobem a l'hora d'escollir llenguatge per introduir els alumnes a la programació. Què és més convenient? Un llenguatge com `C` on els alumnes aprendran el funcionament intern d'un ordinador, i a partir del qual els serà fàcil adaptar-se a altres llenguatges? O un llenguatge com `Python` amb una corba d'aprenentatge més lleugera que permet centrar-se en la part matemàtica i amb una demanda major a la indústria?

Nosaltres, sense entrar en aquesta polèmica, presentem Julia com una opció a mig camí.

L'alumne pot obtenir ràpidament un codi que funcioni centrant-se en la part matemàtica del problema, i a la vegada, després pot centrar-se a optimitzar-lo pel compilador de Julia, aprofundint en el coneixement intern de l'ordinador.

La recerca científica amb Julia

Ja hem vist que Julia es va dissenyar per fer-lo útil per a tasques científiques. En aquest sentit, un dels pilars fundacionals de Julia és simplificar i sistematitzar la reproductibilitat en aplicacions científiques. Un resultat científic ha de poder ser reproduït per qualsevol persona que pugui replicar les condicions en què s'ha obtingut. Per exemple, els resultats d'un experiment per estudiar la caiguda lliure d'objectes a prop de la superfície de la Terra han de poder ser reproduïts per qualsevol que tingui instruments de mesura similars. El problema ve quan uns resultats es basen en l'execució d'un programa, perquè aquesta es pot veure afectada per l'entorn d'execució (sistema operatiu, elements de maquinari, versions de les llibreries que s'utilitzen, etc.).

Julia permet desar totes les variables de l'entorn d'execució en un fitxer. D'aquesta manera, es pot analitzar tant la correcció del codi que s'ha fet servir com totes les variables secundàries que poden afectar-ne l'execució. A més a més, donat un fitxer que contingui la informació de l'entorn d'execució, Julia permet imitar aquest entorn per tal de poder reproduir-la sense preocupar-nos de les característiques de la nostra màquina i el programari instal·lat.

Matemàtiques i música

Musimáticas: a la recerca de científiques històriques i actuals

Laura Farré
Doctoranda a Royal Birmingham Conservatorie, Birmingham City University

La inventora del primer algoritme informàtic de la història, Ada Lovelace (1815-1852), ja va predir en el seu dia que les màquines serien capaces de qualsevol cosa, fins i tot de compondre música, i que les seves úniques limitacions serien les de la imaginació d'aquelles persones que les programessin.

Tot i que diversos matemàtics van establir notables fites en aquest camp, com ara Gottfried Leibniz (1646-1716), Charles Babbage (1791-1871), la mateixa Ada Lovelace, i Alan Turing (1912-1954), Lovelace també va innovar a l'hora de vaticinar que els ordinadors del futur anirien més enllà d'esdevenir pures màquines de calcular.

És per aquest motiu, que amb aquesta matemàtica es va estrenar el programa *Musimáticas* a Radio Clásica, en el qual des de ja fa dues temporades d'estiu, cada setmana s'utilitza la trajectòria d'una científica històrica o actual com a font d'inspiració, servint-se de les figures mitològiques de les muses, per explicar una connexió entre la música i les matemàtiques.

Imatge promocional d'Ada Lovelace del programa radiofònic *Musimáticas* de Radio Clásica, presentat per Laura Farré Rozada

A data d'avui, el programa ha donat a conèixer als oients d'aquesta emissora estatal, les vides