
Table of Contents

Introducció	1.1
Ciència dels ordinadors	1.2
Llenguatges de programació	1.3
Python: com funciona un llenguatge de programació?	1.4
HTML i CSS: disseny web	1.5
SQL: SQL: creació i gestió de bases de dades	1.6
MVC: model, vista i controlador	1.7
Flask: microframework Python	1.8
BiblioTR	1.9
Conclusions	1.10
Bibliografia	1.11

Introducció

Aquest treball es basa en la creació de BiblioTR, una aplicació web on els usuaris poden penjar i visualitzar treballs de recerca amb la possibilitat que aquests siguin puntuats pels professors corresponents (de manera fiable).

D'una banda, tenia clar que la base del meu treball havia d'estar dins l'àmbit de la programació, ja que és un tema que m'interessa molt des que sóc petit i és al que vull dedicar el meu futur. D'altra banda, volia que el resultat del treball tingués alguna utilitat.

Després de moltes idees d'aplicacions poc útils se'm va ocórrer BiblioTR.

Cada any, a Catalunya, s'escriuen milers de treballs de recerca. Són treballs fets amb molt esforç per part dels alumnes de segon de Batxillerat. Una vegada llegits i exposats, molts d'aquests treballs queden guardats i oblidats. Em sembla que oferir la possibilitat d'agrupar els treballs de recerca i fer-los de lliure accés és una manera molt efectiva d'aprofitar l'esforç dels estudiants proporcionant coneixement.

Objectius

Els objectius als quals vull arribar fent aquest treball són:

- Aprendre com fer una aplicació web segura i funcional.
- Fer una aplicació web segura i funcional que sigui útil.
- Millorar les meves habilitats de redactat.
- Millorar les meves habilitats d'exposició oral.

Metodologia

En primer lloc vaig fer la part pràctica i posteriorment la teòrica.

Per fer la part pràctica vaig seguir els passos següents:

- Cercar informació sobre com fer una aplicació web.
- Idear l'aplicació web i la seva funcionalitat així com els seus objectius.
- Programar l'aplicació amb els diferents llenguatges de programació escollits resolent les diferents incidències que anaven sorgint.
- Comprovació i rectificació d'errors.
- Penjar la web en un servidor creat per mi amb una Raspberry Pi.
- Omplir la web amb treballs i perfils de diferents estudiants.

Un cop finalitzada la part pràctica vaig redactar la part teòrica on explico de la millor manera possible els diferents conceptes necessaris per a saber com ha estat creada l'aplicació.

La ciència dels ordinadors

La ciència dels ordinadors és la ciència que s'encarrega de l'estudi dels ordinadors i els conceptes computacionals que els envolten. Estudia tant el programari (conjunt de programes i procediments que fan alguna tasca dins d'un ordinador) com el maquinari (conjunt de parts físiques d'un ordinador), així com l'àmbit de xarxes Internet inclòs.

L'aspecte maquinari en la ciència dels ordinadors es conjuga amb l'enginyeria elèctrica. Inclou el disseny bàsic d'ordinadors i la manera en la que aquests funcionen. Entendre com funciona a nivell físic el maquinari d'un ordinador serveix per entendre en profunditat el funcionament del programari. Per exemple, aprendre les bases del llenguatge binari et permet entendre com un ordinador pot ser capaç de sumar, restar, dividir i multiplicar. Aquests coneixements ajuden als programadors a optimitzar els seus programes.

La part de programari de la ciència dels ordinadors cobreix tant els conceptes de la programació (funcions, algorismes, disseny de codi, etc.) com els diferents llenguatges de programació amb els que aquests es duen a terme. També inclou compiladors, sistemes operatius i aplicacions de software així com aspectes enfocats a l'usuari com podrien ser la representació gràfica i el disseny d'interfícies d'usuari.

Com que actualment gairebé tots els ordinadors estan connectats a internet, la ciència dels ordinadors cobreix també la tecnologia darrera del internet. Això inclou protocols d'internet, telecomunicacions, i conceptes de xarxes així com el disseny de webs i l'administració de xarxes i servidors.

Existeixen dos grans àrees en el món de la ciència dels ordinadors, la ciència dels ordinadors teòrica i la ciència dels ordinadors aplicada.

Ciència dels ordinadors teòrica

La ciència dels ordinadors teòrica tracta idees lògiques i abstractes i està profundament connectada amb les matemàtiques. La programació gira al voltant de les idees que aquesta tracta. El seu objectiu és entendre la naturalesa de la computació i millorar la seva eficàcia ideant i perfeccionant diferents metodologies.

Dintre d'aquesta part teòrica de la ciència dels ordinadors podem trobar molts camps d'estudi, dos dels més importants són l'estudi d'algorismes informàtics i estructures de dades.

Algorismes

Un algorisme és un conjunt finit d'instruccions o passos a seguir que serveixen per a executar una tasca o resoldre un problema. En la vida quotidiana utilitzem algorismes en multitud d'ocasions, per exemple en cuinar o passejar el gos. Un algorisme es pot representar de maneres diferents, dues d'elles podrien ser el pseudocodi o els diagrames de flux.

El concepte d'algorisme és essencial quan parlem de programació, un programa informàtic és en realitat una sèrie d'instruccions ordenades, codificades en un llenguatge de programació que expressen un algorisme.

Un algorisme està sempre format per tres parts:

- Entrada de dades: recopilació de totes les dades necessàries per a l'execució sense errors de l'algorisme.
- Processament de dades: la seqüència de passos i condicions de l'algorisme.
- Sortida de resultats: dades obtingudes després de l'execució de l'algorisme.

En l'àmbit de la programació trobar algorismes ràpids i funcionals és essencial. A continuació veure'm un exemple de demostració.

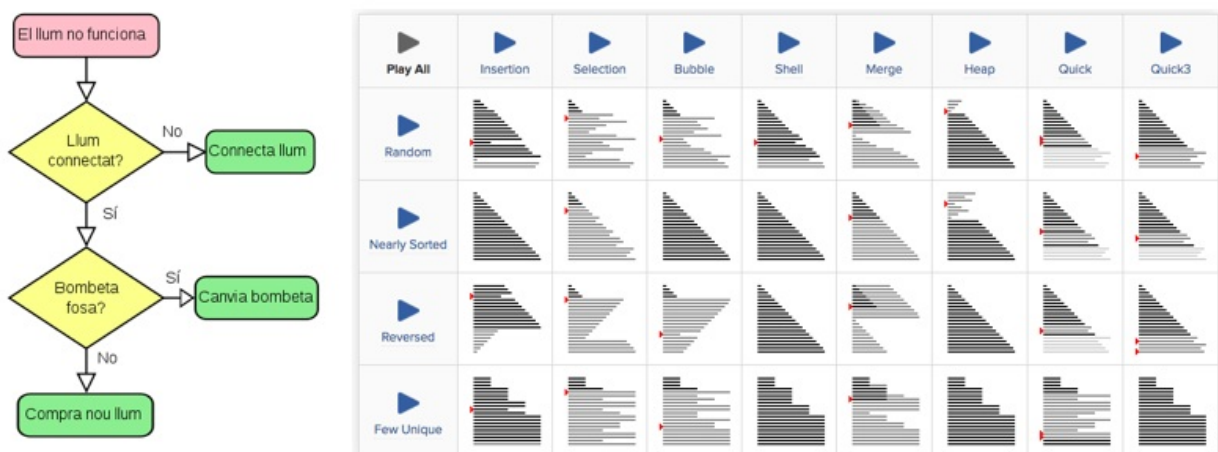


Diagrama de flux de com reparar un llum (<https://ca.wikipedia.org/wiki/Algorisme>) Visualització d'algorismes d'ordenació (<https://www.toptal.com/developers/sorting-algorithms>)

Estructures de dades

Les estructures de dades són les diferents formes d'organització de dades en un ordinador perquè puguin ser utilitzades de manera eficaç per un o més programes.

Generalment, la eficiència d'una estructura de dades serveix per complementar l'eficiència de l'algorisme que l'utilitza.

Fitxer Edita Visualitza Insereix Eines Finestra Ajuda									
	Id_Client	CognomsNom	Adreça	Població	Provincia	Telefon	DataNaixement	Mail	Observacions
1		Perez, Juan	c/San Diego, 24	Reus	Tarragona	643284736	05/04/72	jperez@hotmail.com	
2		García, Francisco	Plaça Prim, 45	Reus	Tarragona	643284436	26/11/71	fgarcia@hotmail.com	
3		Muñoz, Jose	Avd. Jaume I, 98	Salou	Tarragona	612284736	10/05/69	jmuñoz@hotmail.com	Registre
4		Díaz, Luis	C/ Ample, 88	Barcelona	Barcelona	643294736	20/06/72	ldiaz@hotmail.com	
5		Gimeno, Isabel	C/ de les Tapies, 5	Solsona	Lleida	643284147	23/12/70	igimeno@hotmail.com	
6		Miguel, Mateo	C/ Fira, 32	Reus	Tarragona	643284741	23/01/70	mmateo@hotmail.com	
7		Ricart, Marta	C/ Comte Urgell, 78	Sitges	Barcelona	643284852	13/01/70	mrkart@hotmail.com	
8		Rovira, Laura	C/ Major, 23	Bellver	Barcelona	643284258	03/02/61	lrovira@hotmail.com	
9		Dorca, Carme	Plaça de la Font, 18	Berga	Barcelona	643284369	23/03/62		
10		Donat, Lluís	Avinguda Montserrat, 152	Reus	Tarragona	643284963	30/07/79	Ldonat@gmail.com	
11		Boluda, Anna	C/ Verge del Viñet, 45	Barcelona	Barcelona	678284755	30/12/69	aboluda@hotmail.com	
12		Calahorra, Pere	C/ Cervantes, 91	Salou	Tarragona	698284712	01/08/59	pcalahorra@hotmail.com	
13		Sans, Miquel	C/ Floreal, 28	Sitges	Barcelona	645284731	13/09/78	msans@hotmail.com	
14		Puig, Sara	C/ dels Pins, 30	Maials	Lleida	665284755	30/10/75	spuig@hotmail.com	
15		Cobo, Rosa	C/ Espatller, 11	Salou	Tarragona	632284717	05/11/67	rcobo@hotmail.com	
16		Barrillo, Anna	C/ Miró, 90	Sils	Girona	612284777	19/12/72	Abarrillo@hotmail.com	
17		Camps, Rita	C/ Tapioles, 67	Girona	Girona	615284769	17/01/71	rcamps@gmail.com	
18		Lloberes, Blanca	C/ Bonaire, 22	Bescanó	Girona	695284722	27/02/72		
19		Brugaroles, Miquel	Plaça Espanyola, 34	Salt	Girona	675284713	20/03/62	mbrugarolesrius@hotmail.com	
20		Zurita, Rosina	Avvinguda Güell, 89	Girona	Girona	635284743	21/04/63	rzurita@gmail.com	

Estructura d'una base de dades

(http://ioc.xtec.cat/materials/FP/Materials/0252_AFI/AFI_0252_M05/web/html/WebContent/u5/a1/continguts.html)

Ciència dels ordinadors aplicada

Aquesta àrea està formada per tot el conjunt d'aplicacions pràctiques i útils de la part teòrica de la ciència dels ordinadors. Aquí s'estudia els diferents tipus de productes informàtics i com es creen pas per pas així com les diferents tècniques utilitzades en cada àmbit.

Alguns dels àmbits més rellevants de la ciència dels ordinadors aplicada podrien ser la computació científica, els estudis sobre interacció humà-ordinador o la intel·ligència artificial.

Computació científica

La computació científica o ciència computacional és el camp d'estudi relacionat amb la construcció de models matemàtics, les tècniques d'anàlisi quantitatiu i l'ús dels ordinadors per a analitzar i resoldre problemes científics. De manera pràctica podria ser la simulació per ordinador i altres aspectes de la programació orientats a les diferents disciplines científiques.

Alguns àmbits de la computació científica podrien ser l'anàlisi numèric, la física i la química computacional i la bioinformàtica.

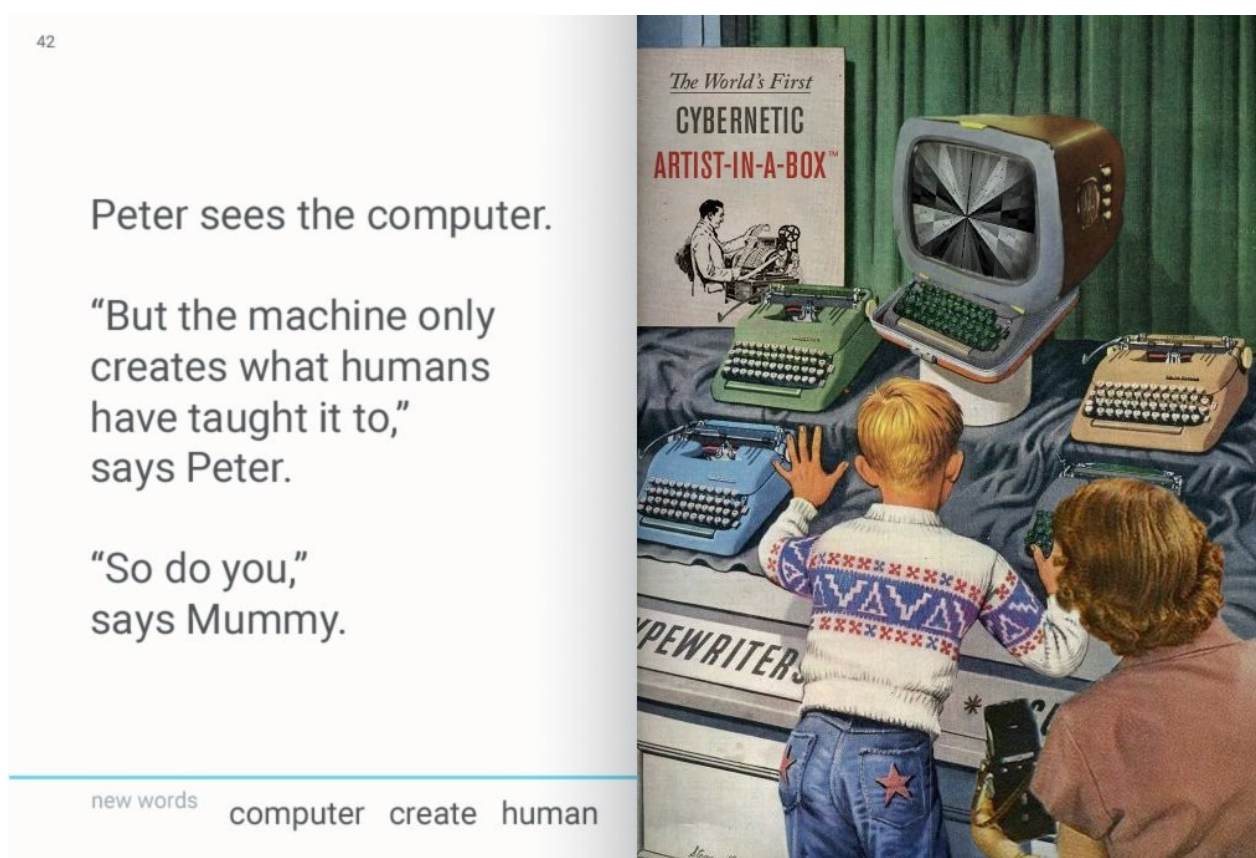
Interacció humà-ordinador

Recerca que desenvolupa teories, principis i guies a seguir per a la creació de millors experiències d'usuari en dispositius com per exemple ordinadors, mòbils o tabletas.

Intel·ligència artificial

La intel·ligència artificial és una part de la ciència dels ordinadors aplicada que es dedica principalment al desenvolupament d'algoritmes que permeten que una màquina (l'ordinador) prengui decisions de manera coherent i intel·ligent, semblant a com ho faria un humà.

El britànic Alan Turing durant la Segona Guerra Mundial i sota els auspicis del servei d'intel·ligència britànic va començar a construir les bases del que avui coneixem com intel·ligència artificial.



Mario Klingemann (<http://mario-klingemann.tumblr.com/>)

En Peter veu l'ordinador.

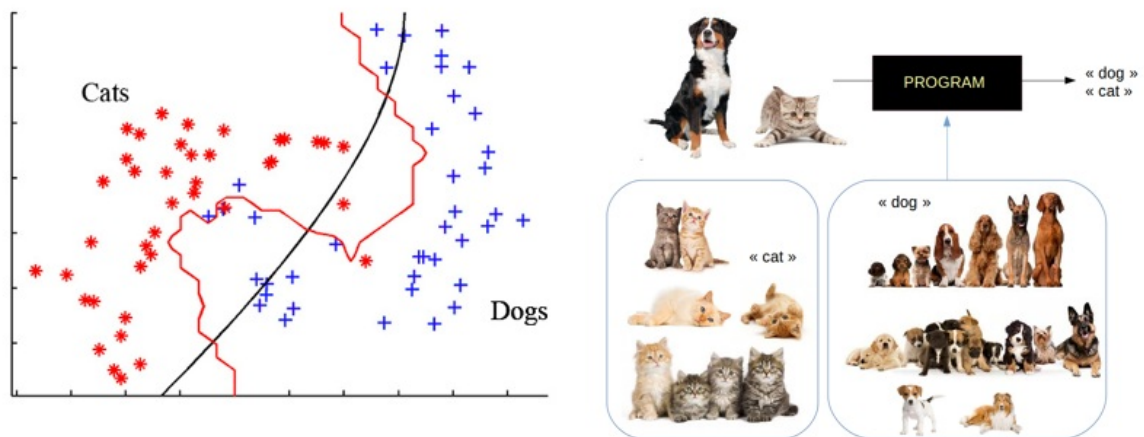
"Però la màquina només crea el que els humans l'hi han ensenyat" diu en Peter

"Igual que tu" diu la mare

Aprenentatge automàtic

L'aprenentatge automàtic (machine learning en anglès), és un àmbit de la intel·ligència artificial que està dedicat al disseny, l'anàlisi i el desenvolupament d'algoritmes i tècniques que permeten que les màquines evolucionin. Es tracta de crear programes capaços de generalitzar comportaments a partir del reconeixement de patrons o la classificació de dades. L'aprenentatge automàtic està, en gran part, relacionat amb el camp de l'estadística i pot estar en contacte amb la informàtica teòrica, doncs es pot donar el cas en que la complexitat computacional de l'algoritme sigui tant alta que ens veiem obligats a estudiar-lo des d'un punt de vista teòric sense poder posar-lo en pràctica.

Encara que no en siguem conscients, estem en contacte amb l'aprenentatge automàtic constantment. Quan Google auto-completa les nostres recerques o Gmail amaga el correu brossa de la nostra safata d'entrada de manera automàtica, ho fan a partir de l'aprenentatge automàtic. Quan Netflix ens recomana una pel·lícula a veure, Amazon un producte a comprar o Facebook un amic a afegir, també ho fan a partir d'algoritmes d'aquest tipus. Els assistents de veu, com per exemple la Siri de Apple o la Cortana de Windows, reconeixen, interpreten i reaccionen a la veu humana gràcies a aquests tipus d'aplicacions.



Identificació automàtica de gossos i gats (<http://freakonometrics.free.fr/bzh/cat-dog2.png>)

Llenguatges de programació

Un llenguatge de programació és un llenguatge informàtic formal que ens permet controlar una màquina (normalment un ordinador). Cada llenguatge de programació té una sèrie de normes sintàctiques i semàntiques que cal seguir. Aquestes normes defineixen l'estil del llenguatge.

Podem classificar els llenguatges de programació en dues classes: els de baix nivell i els d'alt nivell.

- Un llenguatge de baix nivell té una forma críptica, és proper al codi binari i s'utilitza internament en alguns tipus d'ordinador específics.
- Un llenguatge d'alt nivell és senzill, intuïtiu i ràpid d'utilitzar.

Comparació entre un llenguatge d'alt nivell i un de baix:

```
MODEL SMALL
IDEAL
STACK 100H

DATA SEG
HW DB 'Hola!$'

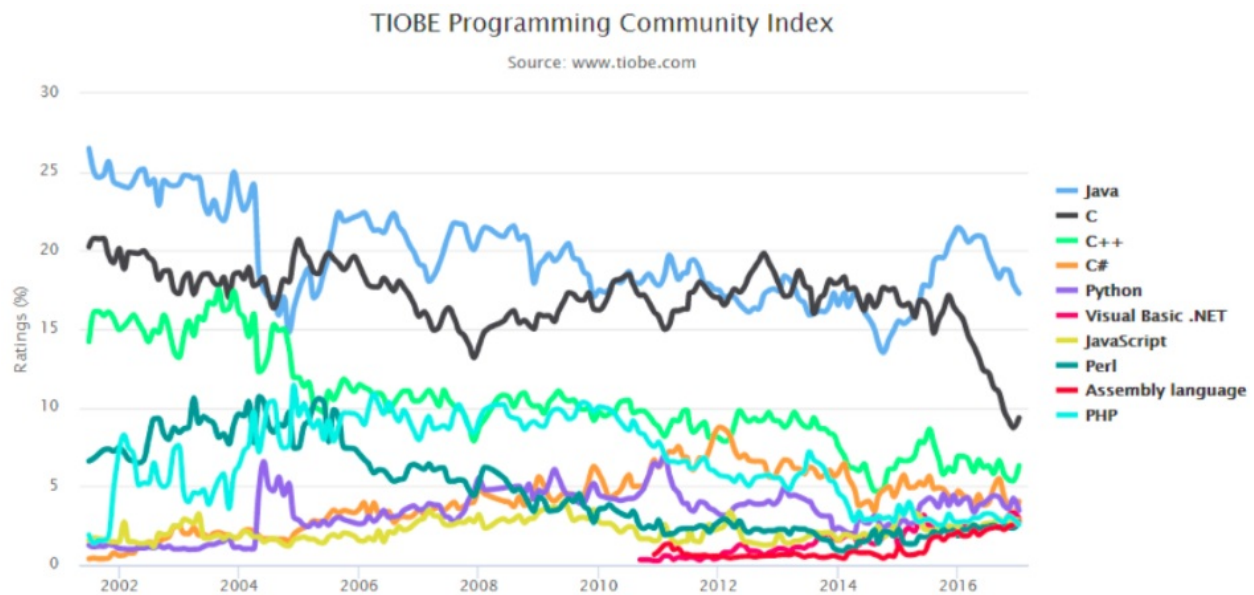
CODE SEG
MOV AX, @data
MOV DS, AX
MOV DX, OFFSET HW
MOV AH, 09H
INT 21H
MOV AX, 4C00H
INT 21H
END
```

```
print ("Hola!")
```

Els dos exemples fan exactament el mateix, mostrar 'Hola!' en pantalla. El primer exemple que ens trobem està escrit en llenguatge d'assemblador per a màquines x86, un llenguatge de baix nivell. El segon està escrit en Python3, un llenguatge d'alt nivell multiplataforma (útil en diversos tipus de màquines).

La diferència és clara. Python és molt més pràctic, fa en una línia el que l'assemblador fa en 14 i, a més, d'una manera molt intuïtiva, doncs, 'print' en anglès, significa 'imprimir' en català.

Segons TIOB, una plataforma que analitza constantment l'ús del software a nivell mundial (902 milions de línies de codi diàries). Els llenguatges de programació més utilitzats en els 14 últims anys són els següents:



Els llenguatges de programació més utilitzats en els 14 últims anys segons TIOBE (www.tiobe.com)

Tots els llenguatges de la llista excepte Assemblador són llenguatges d'alt nivell.

Quins llenguatges de programació utilitzaré en el projecte i perquè

Després de fer recerca i haver valorat un gran nombre d'opcions he arribat a la conclusió que els millors llenguatges de programació pel meu objectiu són els següents:

- Python amb el framework Flask per la part lògica de l'aplicació i el control del servidor. He escollit Python i Flask per la seva simplicitat i el seu alt potencial. Podria haver utilitzat per exemple PHP o Java EE però la meua aplicació no necessita eines tan especialitzades.
- HTML, CSS i JavaScript per la interfície d'usuari (la pàgina web). En aquesta part no hi ha pràcticament altres opcions a considerar.
- SQL amb el sistema SQLite per la gestió i el control de dades (base de dades). He decidit utilitzar SQL perquè és flexible, segur i potent i té molta documentació i SQLite perquè és ràpid i minimalista.

Python: com funciona un llenguatge de programació?

Python és un llenguatge de programació d'alt nivell i de propòsit general molt utilitzat. Va ser creat l'any 1991 per l'informàtic neerlandès Guido van Rossum. La seva filosofia de disseny busca que el codi sigui entenedor i minimalista. Molts programadors experts afirmen que, per la seva simplicitat i el seu alt potencial, Python és una molt bona opció per iniciar-te en l'àmbit de la programació.

Python s'ha utilitzat en moltes aplicacions professionals, algunes d'elles són YouTube, DropBox, Google, Instagram i Reddit.

A continuació veurem, a partir d'exemples, com funciona un llenguatge de programació, concretament, com funciona Python.

Edició i execució de codi

Abans de veure el funcionament de Python haurem d'aprendre com escriure i executar codi.

Per executar codi Python és necessari tindre l'interpret de Python instal·lat en el nostre sistema. Els ordinadors Apple i moltes distribucions Linux ja el tenen integrat per defecte.

Python inclou un mode interactiu que ens permet provar i veure resultats de forma immediata en la consola, sense la necessitat de crear i executar arxius. Es pot accedir al mode interactiu executant la següent línia en la consola del sistema.

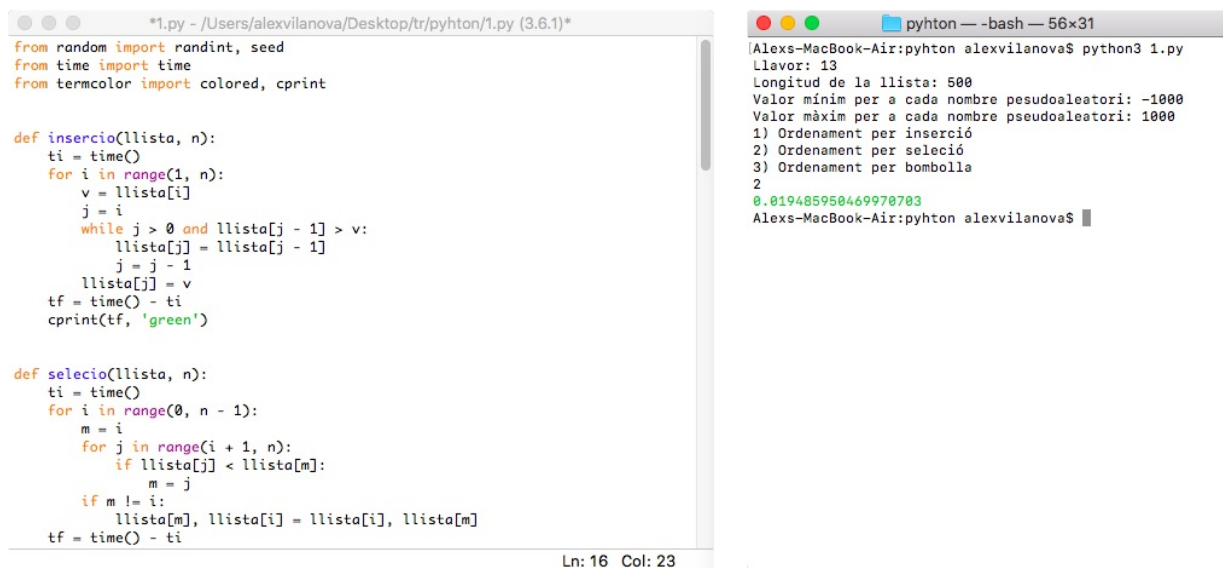
```
python
```

Aquest mode interactiu està molt bé per fer proves puntuals però, a l'hora de crear un programa, ens interessa tindre'l guardat i poder-lo executar quan vulguem sense la necessitat d'escriure tot el codi en la consola. El tipus de fitxer que utilitzen els programes Python és `.py`.

L'edició del codi es pot fer des de qualsevol editor de text. Tot i així, utilitzar el bloc de notes o Microsoft Word per escriure codi no és gaire pràctic. Per això s'han creat editors de text especialitzats en programació. Alguns d'aquests podrien ser Atom, SublimeText o Emacs. Aquests editors adapten el color del text millorant així la llegibilitat del codi, ens avisen si hi ha errors sintàctics i milloren el rendiment de treball autocompletant codi.

Per executar el codi d'un `arxiu.py` només hem d'escriure el següent codi en la consola del sistema.

python arxiu.py



```
*1.py - /Users/alexvilanova/Desktop/tr/python/1.py (3.6.1)*
from random import randint, seed
from time import time
from termcolor import colored, cprint

def insercio(llista, n):
    ti = time()
    for i in range(1, n):
        v = llista[i]
        j = i
        while j > 0 and llista[j - 1] > v:
            llista[j] = llista[j - 1]
            j = j - 1
        llista[j] = v
    tf = time() - ti
    cprint(tf, 'green')

def seleccio(llista, n):
    ti = time()
    for i in range(0, n - 1):
        m = i
        for j in range(i + 1, n):
            if llista[j] < llista[m]:
                m = j
        if m != i:
            llista[m], llista[i] = llista[i], llista[m]
    tf = time() - ti
```

```
pyhton --bash -- 56x31
Alexs-MacBook-Air:pyhton alexvilanova$ python3 1.py
Llavor: 13
Longitud de la llista: 500
Valor mínim per a cada nombre pseudoaleatori: -1000
Valor màxim per a cada nombre pseudoaleatori: 1000
1) Ordenament per inserció
2) Ordenament per selecció
3) Ordenament per bombolla
2
0.019485950469970703
Alexs-MacBook-Air:pyhton alexvilanova$
```

Edició i execució de codi Python

A la fotografia de l'esquerra podem veure l'editor Python per defecte en acció editant un arxiu `1.py` i, a la de la dreta, l'execució de l'arxiu `1.py` des de la consola d'Apple.

Cal remarcar l'existència dels IDE o entorns de desenvolupament integrat, editors de text amb consola inclosa. Aquests editors de text ens permeten, a part d'editar codi, executar-lo sense la necessitat de programes externs. PyCharm és un exemple d'IDE de codi Python.

Hola, món!

Un programa `Hola, món!` simplement imprimeix el text `Hola, món!` en la consola del sistema. Tradicionalment, és el primer exercici que fas quan estàs començant a aprendre un nou llenguatge de programació.

En Python, el programa `Hola, món!` és molt senzill d'entendre. Simplement necessitem la funció `print` i una constant de text `Hola, món!`.

Codi del programa `hola-món.py`:

```
print("Hola, món!")
```

Sortida (output):

```
$ python hola-món.py  
Hola, món!
```

Comentaris

Els comentaris en un llenguatge de programació són línies de codi que no s'executen. S'utilitzen principalment per informar els lectors del codi sobre que s'està fent en cada part. Molts programadors els utilitzen per apuntar tasques a fer o apunts generals.

```
# això és un comentari  
print("Hola, món!") # això és un altre comentari
```

Constants literals

Una constant literal podria ser un nombre com per exemple `5` o `9.8` o una cadena de text com per exemple `"Hola, món!"` o `'Joan'`.

Es diuen constants literals perquè els valors que representen són invariables i literals. El nombre `5` significarà sempre `5` i el nom `"Joan"` significarà sempre `"Joan"`.

Podríem dividir el conjunt de constant literals en constants literals numèriques i textuais i dividir les numèriques en enteres i decimals.

No hi ha diferència entre les cadenes de text definides entre `"` i les cadenes de text definides entre `'`. Es poden escriure cadenes de text de varies línies utilitzant `"""` o `'''`.

Estructures de dades

En Python hi ha diferents estructures de dades disponibles: llistes, tuples, diccionaris, cadenes, sets i frozensets.

Les llistes, les tuples i les cadenes són seqüències ordenades d'objectes. A diferència de les cadenes que només contenen caràcters, les llistes i les tuples poden contenir qualsevol tipus d'objectes. Les cadenes i les tuples són immutables. Les llistes són mutables, es poden ampliar i reduir en qualsevol moment. Els sets són seqüències d'elements únics mutables. Els frozensets són sets immutables.

Les estructures immutables són més ràpides i requereixen menys memòria que les mutables, per això sempre que no sigui necessari editar l'estructura és millor utilitzar-ne una d'immutable.

Les llistes es defineixen entre `[]`

```
[1, 2.5, "a"]
```

Les tuples es defineixen entre `()`

```
(1, 2.5, "a")
```

Els diccionaris es defineixen entre `{}`

```
{"a":1, "b":2.5}
```

Variables

Les variables d'un programa informàtic són espais de memòria reservats per guardar valors que podran ser recuperats i modificats posteriorment. En Python, les variables es declaren amb el nom de la variable acompanyat del signe `=` i la seva constant literal.

Codi del programa `alex.py` :

```
nom = "Alex"
edat = 17
print(nom)
print(edat)
print(nom, edat)
print("Em dic {0} i tinc {1} anys".format(nom, edat))
print("Tinc {0} anys i em dic {1}".format(edat, nom))
```

Sortida (output):

```
$ python alex.py
Alex
17
Alex 17
Em dic Alex i tinc 17 anys
Tinc 17 anys i em dic Alex
```

En les dos primeres línies de codi el que hem fet és assignar la cadena de text `"Alex"` a la variable `nom` i el valor `16` a la variable `edat` . En les dues últimes línies del codi de sortida, la funció `format` substitueix els elements numerats amb la notació `{n}` partint de `n = 0` amb les variables llistades en ordre. Aquesta funció és molt útil perquè ens permet formar cadenes de text amb variables d'una manera simple i fàcil.

Operadors i expressions

Un exemple d'expressió podria ser `2 + 3`. Una expressió està formada per operadors i operands. En el cas mencionat, l'operador seria `+` i els operands serien les constants numèriques `2` i `3`.

Taula amb els operadors que suporta Python:

Operació	Operador	Expressió (exemple)
Suma	<code>+</code>	<code>2 + 3 == 5</code>
Resta	<code>-</code>	<code>2 - 3 == -1</code>
Multiplicació	<code>*</code>	<code>2 * 3 == 6</code>
Divisió	<code>/</code>	<code>2 / 3 == 0.6666666666666666</code>
Mòdul (residu)	<code>%</code>	<code>2 % 3 == 2</code>
Potència	<code>**</code>	<code>2 ** 3 == 8</code>
Divisió entera	<code>//</code>	<code>2 // 3 == 0</code>

L'ordre d'operació de Python és el mateix que en les matemàtiques.

- parèntesis `()`
- potències `**`
- multiplicació `*`, divisió `/`, divisió entera `//` i residu `%`
- Suma `+` i resta `-`

Quan volem assignar un valor d'una operació a una variable i almenys un dels dos operands de l'operació és la mateixa variable podem abreviar utilitzant l'operador seguit del signe `=`. Exemple:

És el mateix escriure:

```
x = x + 5
x = x - 5
x = x * 5
x = x / 5
x = x ** 2
# etc
```

que escriure:

```
x += 5
x -= 5
x *= 5
x /= 5
x **= 2
# etc
```

A continuació veurem un exemple d'operacions amb variables, un programa senzill que resol equacions de segon grau.

Codi del programa `2grau.py` :

```
a = int(input("Escriu el valor de a: "))
b = int(input("Escriu el valor de b: "))
c = int(input("Escriu el valor de c: "))
r1 = (-b + (b ** 2 - 4 * a * c) ** (1 / 2)) / (2 * a)
r2 = (-b - (b ** 2 - 4 * a * c) ** (1 / 2)) / (2 * a)
print("Solucions de l'equació: {0} i {1}".format(r1, r2))
```

Sortida (output):

```
$ python 2grau.py
Escriu el valor de a: -1
Escriu el valor de b: 7
Escriu el valor de c: -10
Solucions de l'equació: 2.0 i 5.0
```

En aquest exemple veiem per primera vegada les funcions `int` i `input`. El que fan és molt senzill d'entendre.

La funció `input` llegeix el que l'usuari escriu en la consola del sistema i ho retorna en forma de constant textual.

La funció `int` transforma l'argument introduït en un nombre enter (si és possible).

En aquest cas en concret, com podem veure en la sortida del programa, els tres valors introduïts són vàlids per ser transformats en nombres. Les constants textuais `"1"`, `"7"` i `"-10"` esdevenen constants numèriques `1`, `7` i `-10`.

Les tres variables donarien lloc a la següent funció: $-x^2 + 7x - 10 = 0$. Aplicant la fórmula de les equacions quadràtiques $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ obtenim les dues solucions `2.0` i `5.0`. Les assignem a les variables `r1` i `r2` i les mostrem en pantalla.

Amb aquest petit programa de set línies podem resoldre totes les equacions de segon grau que volguem tan sols escrivint els valors de les tres variables de cada equació.

L'operador de suma `+` també és útil en cadenes de text.

Codi del programa `alex-2.py` :

```
nom = "Alex"
edat = 17
print("Em dic {0} i tinc {1} anys".format(nom, edat))
print("Em dic " + nom + " i tinc " + str(edat) + " anys")
```

Sortida (output):

```
$ python alex-2.py
Em dic Alex i tinc 17 anys
Em dic Alex i tinc 17 anys
```

Com es pot observar, la tercera i la quarta línia donen el mateix resultat. Aquí observem el principal avantatge de la funció `format` , simplificar la substitució de variables en cadenes de text.

Comparadors i expressions booleans

Una expressió booleana pot tenir dos valors `True` o `False` , `1` o `0` . Aquesta expressió s'aconsegueix utilitzant els comparadors. Un exemple d'expressió booleana podria ser: `1 + 2 == 3` . En aquesta expressió, el comparador utilitzat és el d'igualtat (`==`) i dona com a resultat `True` , ja que `3 = 3` .

Taula amb els comparadors que suporta Python:

Comparació	Comparador	Expressió (exemple)
Igual	<code>==</code>	<code>1 + 3 == 4 == False</code>
Diferent	<code>!=</code>	<code>"4" != 4 == True</code>
Major	<code>></code>	<code>0.34 > 1 / 3</code>
Major o igual	<code>>=</code>	<code>0.34 >= 1 / 3 == True</code>
Menor	<code><</code>	<code>2 < 2 == False</code>
Menor o igual	<code><=</code>	<code>2 <= 2 == True</code>

Una expressió booleana es pot assignar a una variable.

Codi del programa `si-o-no.py` :

```
a = True
a = 2 == 5
b = 2 >= 1
print(a)
print(b)
```

Sortida (output):

```
$ python si-o-no.py
False
True
```

Decisions

Les expressions `if` , `else` i `elif` , permeten a l'ordinador prendre decisions i executar diferents parts de codi depenent de certes condicions booleans.

Codi del programa `edat.py` :

```
edat = int(input("Edat: "))
if(edat >= 18):
    print("Major d'edat")
elif(edat >= 0):
    print("Menor d'edat")
else:
    print("L'edat no pot ser un nombre negatiu")
```

Sortida (output):

```
$ python edat.py
Edat: 4
Menor d'edat
$ python edat.py
Edat: 50
Major d'edat
$ python edat.py
Edat: -7
L'edat no pot ser un nombre negatiu
```

En l'exemple anterior trobem els tres tipus d'expressions condicionals:

`if` (si): si l'edat és major o igual a `18` , executar `print("Major d'edat")`

`elif` (si no, si): si l'edat no és major o igual a `18` , si és major o menor a `0` , executar `print("Menor d'edat")`

`else` (si no): si cap de les condicions anteriors es compleix, executar `print("L'edat no pot ser un nombre negatiu")`

Normalment en un conjunt de condicions trobaríem un `if`, múltiples `elif` i un `else`.

En la sortida del programa el veiem provat amb tres valors diferents: `4`, `50` i `-7`:

```
4 >= 18 és False i 4 >= 0 és True aleshores print("Menor d'edat")
```

```
50 >= 18 és True aleshores print("Major d'edat")
```

```
-7 >= 18 és False i -7 >= 0 és False aleshores print("L'edat no pot ser un nombre negatiu")
```

Bucle while

Un bucle `while` ens permet executar una part de codi repetidament mentre certa condició sigui `True`. Un cop la condició passa a ser `False`, el programa segueix executant el codi a continuació del bucle amb normalitat.

Codi del programa `10.py`:

```
numero = 0
while(numero < 10):
    numero += 1
    print(numero)
print("Ja s'ha acabat el bucle while")
```

Sortida (output):

```
$ python 10.py
1
2
3
4
5
6
7
8
9
10
Ja s'ha acabat el bucle while
```

En el codi anterior podem veure com una variable `numero` que inicialment val `0` incrementa fins a `10` en un bucle `while`. Un cop la variable `numero` val `10`, `numero < 10 == False`, es surt de bucle i es segueix executant el codi del programa.

Amb un bucle `while` podem executar indefinidament una part de codi:

```
a = 0
while(True):
    a += 1
    print(a)
```

En el codi anterior la variable `a` no pararia mai de créixer.

Bucle for

El bucle `for`, igual que el bucle `while`, serveix per executar repetidament una part de codi. A diferència del bucle `while`, el bucle `for` no para quan certa condició es deixa de complir sinó que para després d'un nombre d'iteracions.

Codi del programa `10_2.py`:

```
for i in range(1, 11):
    print(i)
print("Ja s'ha acabat el bucle for")
```

Sortida (output):

```
$ python 10_2.py
1
2
3
4
5
6
7
8
9
10
Ja s'ha acabat el bucle for
```

El programa `10_2.py` fa exactament el mateix que el programa `10.py`, contar fins a `10`. El bucle `for` i el bucle `while` són molt semblants, moltes vegades podríem utilitzar tant l'un com l'altre que obtindrem el mateix resultat. Normalment s'utilitza `while` quan s'ha de realitzar una tasca indefinidament i `for` quan s'ha de realitzar `i` vegades.

En aquest exemple apareix una funció `range`. Aquesta funció crea una llista tipus `range` amb els límits introduïts.

```
llista = range(1, 11)
list(llista)
```

El codi anterior donaria com a resultat el següent:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Funcions personalitzades

Una funció és un bloc de codi organitzat i reutilitzable que s'utilitza per realitzar una acció única i relacionada. Les funcions proporcionen modularitat a les aplicacions i ens permeten reutilitzar codi sense la necessitat de reescriure'l.

Com ja hem vist, Python ens ofereix moltes funcions integrades com per exemple `print`, `range` o `format`. però també podem crear les nostres pròpies funcions. Aquestes funcions s'anomenen funcions personalitzades. A continuació en veurem un exemple.

Codi del programa `2grau_2.py` :

```
# definir funció
def equacio_segona_grau(a, b, c):
    r1 = (-b + (b ** 2 - 4 * a * c) ** (1 / 2)) / (2 * a)
    r2 = (-b - (b ** 2 - 4 * a * c) ** (1 / 2)) / (2 * a)
    if(r1 == r2):
        return("Solució de l'equació: {1}".format(r1))
    return("Solucions de l'equació: {1} i {2}".format(r1, r2))

# crear llista amb exercicis
exercicis = (
    # equació (a, b, c)
    (1, -5, 6),
    (2, -7, 3),
    (-1, 7, -10),
    (1, -2, 1)
    # etc...
)

# bucle for
for exercici in exercicis:
    print(equacio_segona_grau(exercici[0], exercici[1], exercici[2]))
```

Sortida (output):

```
$ python 2grau_2.py
#1 - Solucions de l'equació: 3.0 i 2.0
#2 - Solucions de l'equació: 3.0 i 0.5
#3 - Solucions de l'equació: 2.0 i 5.0
#4 - Solució de l'equació: 1.0
```

En aquest exemple hi ha una funció personalitzada anomenada `equacio_segona_grau` que depèn de les tres variables: `a`, `b` i `c`. Aquesta funció calcula i retorna a partir del `return` la o les solucions de l'equació de segon grau. Per demostrar la utilitat de la funció es crea una llista amb 4 equacions diferents i amb un bucle `for` s'imprimeixen les 4 solucions.

Classes

Les classes proveeixen una forma d'empaquetar dades i funcionalitat junts. En crear una nova classe, es crea un nou tipus d'objecte, permetent crear noves instàncies d'aquest tipus. Cada instància de classe pot tenir atributs adjunts per mantenir el seu estat. Les instàncies de classe també poden tenir funcions (definits per la seva classe) per modificar el seu estat.

Codi del programa `forma.py` :

```
# definir classe forma
class Forma:
    # funció inicial (definició de variables)
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.descripcio = "No hi ha descripció"
        self.autor = "No hi ha autor"
    # funció area
    def area(self):
        return self.x * self.y
    # funció perimetre
    def perimetre(self):
        return 2 * self.x + 2 * self.y
    # funció descriure
    def descriure(self, descripcio):
        self.descripcio = descripcio
    # funció signar
    def signar(self, autor):
        self.autor = autor
    # funció escalar
    def escalar(self, escala):
        self.x = self.x * escala
        self.y = self.y * escala

# crear classe forma anomenada f1 amb x=10.0 i y=5.0
f1 = Forma(10.0, 5.0)
# mostrar x i y de la forma f1
print("X:{0} Y:{1}".format(f1.x, f1.y))
# mostrar àrea i perimetre de la forma f1
print("Àrea:{0} Perimetre:{1}".format(f1.area(), f1.perimetre()))
# definir l'autor de la forma f1
f1.signar("Alex")
# mostrar descripció i autor de la forma f1
print("Descripció:{0} Autor:{1}".format(f1.descripcio, f1.autor))
# canviar escala de la forma f1
f1.escalar(0.5)
# mostrar x i y de la forma f1
print("X:{0} Y:{1}".format(f1.x, f1.y))
```

Sortida (output):

```
$ python forma.py
X:10.0 Y:5.0
Àrea:50.0 Perimetre:30.0
Descripció:No hi ha descripció Autor:Alex
X:5.0 Y:2.5
```

En l'exemple anterior apareixen dos conceptes nous.

El primer és la funció `__init__`, aquesta funció és una funció definida de python que s'executa només en el moment en què es crea la classe, serveix per definir les variables que tindrà la nostra classe. La classe `Forma` tindrà obligatòriament les variables `x` i `y`, per això estan definides en l'enunciat de la funció mentre que les variables autor i descripció seran opcionals.

El segon és el nom `self`. S'utilitza per referir-se a la classe dins d'ella mateixa. Totes les variables pròpies es defineixen com a `self.variable` i les funcions que depenguin dels paràmetres de la classe han de tenir el nom `self` entre les variables de l'enunciat.

HTML i CSS: disseny web

HTML (llenguatge de marcatge hipertextual - HyperText Markup Language) és el bloc de construcció més bàsic de la web. Descriu i defineix el contingut d'una pàgina web. Altres tecnologies a més de HTML s'utilitzen generalment per descriure l'aparença/presentació (CSS) o la funcionalitat (JavaScript) d'una pàgina web.

Cal recalcar que HTML i CSS no són llenguatges de programació, HTML és un llenguatge de marcatge i CSS un llenguatge de disseny gràfic. Aquests no serviran per realitzar els càlculs ni les operacions lògiques de l'aplicació sinó que servirà per mostrar la interfície d'usuari en forma de pàgina web.

Edició i execució de codi

El codi HTML, CSS i JavaScript igual que el codi Python, es pot editar amb qualsevol editor de codi.

Per executar codi HTML, CSS i JavaScript només necessitem un navegador web i diferents arxius amb les extensions corresponents (`.html` , `.css` i `.js`).

Codi HTML

El llenguatge html funciona amb etiquetes. Cada etiqueta és un element o un bloc de la pàgina.

Una etiqueta pot o no permetre contingut i pot o no tenir atributs:

```
<!-- comentari en HTML -->

<!-- etiquetes que permeten contingut -->

<etiqueta atribut1="">contingut</etiqueta>

<etiqueta atribut1="" atribut2="">
    contingut
</etiqueta>

<!-- etiquetes que NO permeten contingut -->

<etiqueta atribut1="" atribut2="" atribut3=""/>
```

Hi han diferents tipus d'etiquetes. A continuació hi ha una llista amb les més importants.

Etiquetes bàsiques

- Començament i final de la pàgina: `<html> ... </html>`
- Capçalera de la pàgina: `<head>...</head>`
- Títol de la pàgina: `<title>...</title>`
- Cos de la pàgina: `<body>...</body>`

Etiquetes de text

- Capçalera de primer nivell: `<h1>...</h1>`
- Capçalera de segon nivell: `<h2>...</h2>`
- Capçalera de tercer nivell: `<h3>...</h3>`
- Capçalera de quart nivell: `<h4>...</h4>`
- Capçalera de cinquè nivell: `<h5>...</h5>`
- Capçalera de sisè nivell: `<h6>...</h6>`
- Paràgraf: `<p>...</p>`

Estil de text

- Text en negreta: `...`
- Text en cursiva: `<i>...</i>`
- Text subratllat: `<u>...</u>`
- Text tatxat: `<s>...</s>`
- Text centrat: `<center>...</center>`
- Canviar tipus de text:
 - Atribut del tipus de lletra: ``
 - Atribut de mida: ``
 - Atribut de color: ``

Enllaços

- Enllaços (la url pot ser tant d'un fitxer local com d'una pàgina web): `...`

Llistes

- Principi i final d'una llista no numerada: `...`
- Principi i final d'una llista numerada: `...`
- Element d'una llista: `...`

Imatges

- Imatge: ``
 - Atributs de mida: `width="mida"` `height="mida"`

Taules

- Principi i final d'una taula: `<table>...</table>`
- Fila d'una taula: `<tr>...</tr>`
- Cel·la de capçalera d'una taula: `<th>...</th>`
- Cel·la convencional d'una taula: `<td>...</td>`

Formularis

- Principi i final d'un formulari: `<form>...</form>`
 - Atribut d'acció: `action="ruta"`
 - Atribut del mètode d'enviament: `method="get/post"`
- Camp de text: `<input type="text" name="nom" size="amplada" maxlength="max" value="valor" />`
- Camp de text gran: `<textarea name="nombre"> valor </textarea>`
- Camp de contrasenya: `<input type="password" name=" " " size="amplada" maxlength="max" />`
- Botó d'opció (el nom ha de ser igual en grups): `<input type="radio" name="nom" value="valor" />`
- Botó de tic: `<input type="checkbox" name="nom" value="si" />`
- Llista desplegable: `<select name="nom" size="mida" multiple="multiple">... (etiquetas option) ...</select>`
 - `<option value="valor">...</option>`
- Botó de selecció de document: `<input type="file" name="nom" />`
- Botó d'enviar: `<input type="submit" value="text del botó" />`

Etiquetes de disseny

- Selecció d'un bloc gran: `<div>...</div>`
 - Atribut de classe (referència no única): `class="nom"`
 - Atribut de id (referència única): `id="nom"`
- Selecció d'un bloc petit: `...`

Referència a fitxers CSS i JavaScript i incrustació de codi

- Fitxer CSS: `<link rel="stylesheet" href="direcció_fitxer" type="text/css" />`
- Fitxer JavaScript: `<script language="javascript" src="direcció_fitxer" type="text/javascript"> </script>`

- Codi CSS incrustat: `<style type="text/css"> codi css </style>`
- Codi JavaScript incrustat: `<script type="text/javascript"> codi javascript </script>`

Codi CSS

El llenguatge CSS funciona amb selectors i propietats. Un selector podria ser per exemple un element HTML o una classe (que engloba diversos elements), una propietat podria ser per exemple el color o la mida. A continuació hi ha un resum explicatiu amb les bases del llenguatge.

Selectors

- Selector universal: `*`. Afecta tots els elements.
- Selector d'etiqueta: `h1`. Afecta totes les etiquetes amb aquest nom.
- Selector de classe: `.classe`. Afecta totes les etiquetes marcades amb l'atribut `class="classe"`.
- Selector de id: `#id`. Afecta l'etiqueta marcada amb l'atribut `id="id"`.

Propietats

A continuació veurem algunes de las propietats més utilitzades:

Propietats de tipografia

Propietat	Valors	Descripció
color	Color	Color del text
font-family	Nom de la font	Font
font-size	Mida, percentatge	Mida del text
font-weight	Mida	Grossor del text
font-style	Cursiva, normal, etc.	Estil del text

Propietats de text

Propietat	Valors	Descripció
line-height	Mida, percentatge	Espai entre línies
text-decoration	Subratllat, tatxat, etc.	Decoració
letter-spacing	Mida	Espai entre lletres
word-spacing	Mida	Espai entre paraules

Propietats de les vores

Propietat	Valors	Descripció
border-width (border-top-width, border-right-width, border-bottom-width, border-left-width)	Mida	Grossor de les vores (de la vora superior, dreta, inferior, esquerra)
border-color (border-top-color, border-right-color, border-bottom-color, border-left-color)	Color	Color de les vores (de la vora superior, dreta, inferior, esquerra)

Propietats dels marges

Propietat	Valors	Descripció
padding (padding-top, padding-right, padding-bottom, padding-left)	Mida, percentatge	Marge intern (marge intern superior, dret, inferior, esquerra)
margin (margin-top, margin-right, margin-bottom, margin-left)	Mida, percentatge	Marge extern (marge extern superior, dret, inferior, esquerra)

Fons de la pàgina

Propietat	Valors	Descripció
background-color	Color	Color del fons de la pàgina
background-image	URL imatge	Imatge de fons de la pàgina

Tamany dels blocs

Propietat	Valors	Descripció
width i height	Mida, percentatge	Amplada i alçada
max-width i max-height	Mida, percentatge	Amplada i alçada màxima
min-width i min-height	Mida, percentatge	Amplada i alçada mínima

Posicionament

Propietat	Valors	Descripció
position	Estàtica, relativa, etc.	Tipus de posicionament
top, right, bottom, left	Mida, percentatge	Moure a dalt, dreta, baix, esquerra

Visualització

Propietat	Valors	Descripció
display	None, block, inline, etc.	Visualització d'un element
visibility	Visible, amagat	Visibilitat d'un element

Exemple d'una pàgina web amb HTML i CSS

{explicació}

{exemple}

{explicació}

Bootstrap

{explicació}

{exemple}

{explicació}

Disseny responsiu

{explicació}

{exemple}

{explicació}

SQL: creació i gestió de bases de dades

SQL (Structured Query Language o Llenguatge d'interrogació estructurat) és un llenguatge normalitzat de comunicació amb bases de dades relacionals. L'SQL es pot utilitzar dins d'altres llenguatges de programació. La principal característica d'aquest llenguatge és la seva simplicitat, ja que amb pocs coneixements es poden fer consultes bàsiques sobre una base de dades, encara que no per això deixa de ser un llenguatge complet, tant relacionalment com computacionalment.

Hi han moltes bases de dades que estan basades en SQL, dues de les més famoses són MySQL i SQLite, en el nostre cas utilitzarem SQLite.

Característiques principals de SQLite:

- La base de dades completa es troba en un sol fitxer (`.db`).
- Pot funcionar enterament en memòria, la qual cosa la fa molt ràpida.
- Compta amb llibreries d'accés per a molts llenguatges de programació.
- El codi font és de domini públic i es troba molt bé documentat.

Edició i execució de codi

Les bases de dades SQLite es poden editar amb editors d'interfície gràfica o des de la consola o intèrpret.

Estructura d'una base de dades

Les bases de dades estan formades per taules i les taules estan formades per columnes.

Un exemple de base de dades podria ser per exemple `aliments`, dintre de `aliments` trobaríem taules com per exemple `fruites` o `verdures` i dintre d'una taula com per exemple la de `fruites` trobaríem les columnes `mandarina` i `poma`.

Sentències SQL

Les sentències SQL ens permeten actuar amb la base de dades (fitxer `.db`). A continuació veurem les sentències essencials.

Crear i eliminar taula

```
CREATE TABLE nom_taula (  
    columna_1 tipus ATRIBUT_1 ... ATRIBUT_N,  
    columna_2 tipus ATRIBUT_1 ... ATRIBUT_N,  
    ...  
    columna_n tipus ATRIBUT_1 ... ATRIBUT_N,  
    PRIMARY KEY (columna_1),  
    UNIQUE (columna_2),  
    UNIQUE (columna_n)  
)  
  
DROP TABLE nom_taula
```

Els tipus de columna i els atributs de columna els veurem més endavant. Les tres línies finals defineixen la clau primària (identificador únic) i els elements únics (per exemple el correu en una xarxa social).

Modificar taula

```
ALTER TABLE nom_taula acció
```

On acció pot ser:

- Canviar el nom de la taula: `RENAME TO nou_nom`
- Afegir una columna: `ADD COLUMN columna tipus ATRIBUT_1 ... ATRIBUT_N`
- Eliminar una columna: `DROP COLUMN columna`
- Modificar una columna: `MODIFY columna tipus ATRIBUT_1 ... ATRIBUT_N`

Tipus de columnes

Hi han molts tipus de columnes. Algun d'ells són els següents:

- Textuals: `VARCHAR(n_caràcters)`
- Data: `DATE`
- Nombre enter: `INTEGER`
- Nombre decimal: `DECIMAL`
- Conjunt de bytes (imatges, pdf, etc.): `BLOB`

Atributs

Els atributs s'apliquen en la definició d'una columna de la manera que hem vist anteriorment. Dos dels més utilitzats són els següents:

- Essencialitat (ha d'estar definit obligatoriament): `NOT NULL`

- Valor per defecte (si en crear la fila no inserim el valor, s'utilitzarà aquest): `DEFAULT valor`

Insertar i eliminar registres

```
INSERT INTO taula (columna_1, ... , columna_n)
VALUES (valor_1, ... , valor_n)
```

```
DELETE FROM taula
WHERE condicions
```

Un exemple de `condicions` podria ser `columna_x=valor_x OR columna_y=valor_y`

Modificar registres

```
UPDATE taula
SET columna_1=valor_1, ... , columna_n=valor_n
WHERE condicions
```

Seleccionar informació

```
SELECT columna(s)
FROM taula
WHERE condicions
GROUP BY columna_a_agrupar
HAVING condiciones_per_grups
ORDER BY columna_ordenació ASC/DESC
```

`SELECT *` vol dir seleccionar tot.

Exemple de la creació i la gestió d'una base de dades SQLite

En el següent exemple veurem la creació i el control d'una base de dades d'usuaris.

```
sqlite> CREATE TABLE usuari (  
...> id INTEGER NOT NULL,  
...> usuari_nom VARCHAR(20),  
...> correu VARCHAR(254),  
...> contrasenya VARCHAR(80),  
...> nom VARCHAR(40),  
...> centre_educatiu VARCHAR(120),  
...> biografia VARCHAR(240),  
...> data_creacio DATE,  
...> PRIMARY KEY (id),  
...> UNIQUE (usuari_nom),  
...> UNIQUE (correu)  
...> );
```

La taula consta de 8 columnes: `id` , `usuari_nom` , `correu` , `contrasenya` , `nom` , `centre_educatiu` , `biografia` i `data_creacio` . Per evitar problemes, totes les variables textuais tenen límit de text (això vol dir que si el superen, es mostraria un error i no es modificaria la taula). El camp `id` és un identificador general que incrementa automàticament cada vegada que s'afegeix un usuari. Els camps `usuari_nom` i `correu` són únics, si s'intenta afegir un usuari amb el mateix correu o nom d'usuari no seria possible i mostraria un error.

Per afegir un usuari ho farem de la següent manera:

```
sqlite> INSERT INTO usuari (usuari_nom, correu, contrasenya, nom, data_creacio)  
...> VALUES ("alexvilanovab", "alexvilanovab@gmail.com", "1234", "Alex Vilanova", "  
7/1/18");
```

Ara ja hi ha un usuari a la base de dades, veure visualitzar-lo ho podem fer així:

```
sqlite> SELECT * FROM usuari  
...> WHERE usuari_nom="alexvilanovab";  
Output: 1|alexvilanovab|alexvilanovab@gmail.com|1234|Alex Vilanova||7/1/18
```

Com podeu veure aquest usuari no té ni biografia ni centre educatiu assignat.

Així podem modificar l'usuari:

```
sqlite> UPDATE usuari  
...> usuari_nom="alexvb", contrasenya="abcd", centre_educatiu="Institut Eugeni Xammar";  
  
sqlite> SELECT * FROM usuari  
...> WHERE usuari_nom="alexvb";  
1|alexvb|alexvilanovab@gmail.com|abcd|Alex Vilanova||7/1/18
```

Per últim, podem eliminar l'usuari:

```
sqlite> DELETE FROM usuari
...> WHERE usuari_nom="alexvb";
sqlite> SELECT * FROM usuari
...> WHERE usuari_nom="alexvb";
```

Com podem veure, no hi ha resultats en el `SELECT`.

Injecció SQL

La injecció SQL és un mètode d'infiltració de codi intrús que es val d'una vulnerabilitat informàtica present en una aplicació en el nivell de validació de les entrades per realitzar operacions sobre una base de dades. L'origen de la vulnerabilitat radica en la incorrecta revisió de les variables utilitzades en un programa que conté, o bé genera, codi SQL.



Injecció SQL a un radar

(https://www.reddit.com/r/geek/comments/1j9tn3/speed_camera_sql_injection/)

En aquesta foto podem veure un cotxe amb una matrícula falsa que inclou una sentència SQL on posa `DROP DATABASE`. Quan un radar detecta que un cotxe està superant l'excés de velocitat el que fa és analitzar el text de la matrícula i guardar-lo en una base de dades, que podria ser perfectament del tipus SQL. Amb molta sort, si la base de dades del radar no està protegida contra injeccions SQL, quan detecti que aquest cotxe va a massa velocitat executarà el codi de la matrícula `DROP DATABASE` i la base de dades serà eliminada per complet.

L'exemple del cotxe, tot i ser exagerat, no és impossible, tot i així, les injeccions SQL són normalment utilitzades en formularis web de pàgines molt poc segures, doncs avui en dia la gran majoria de pàgines tenen protecció contra aquest tipus d'atacs.

Login Information

Login ID :

Password :

Login successful

Injecció SQL a un inici de sessió (<https://www.codeproject.com/Articles/102284/SQL-Injection-and-Cross-Site-Scripting>)

Aquí hi ha un exemple de com iniciar sessió amb una injecció SQL. A continuació hi ha una simulació amb Python:

```
input_email = "test@test.com' or 1=1'"
input_password = "1234"
sql = "SELECT id FROM users WHERE password='{1}' AND email='{0}'".format(input_password, input_email)
# la sentència SQL seria així
# SELECT * FROM users WHERE password='1234' AND email='test@test.com' or 1=1'
```

La sentència teòricament només ha de donar resultat si la contrasenya i el correu coincideixen tot i així, amb les dades introduïdes, donarà resultat sempre. La clau està en el correu, allà hi ha incrustat una part de SQL `' or 1=1` que modifica el condicional de la sentència fent que doni positiu sempre ja que `1=1` .

Afortunadament, els frameworks (llibries de desenvolupament) actuals, ofereixen protecció i seguretat per aquests tipus d'atacs.

MVC: model, vista i controlador

El model-vista-controlador (MVC) és un patró de l'arquitectura del software creat per l'informàtic noruec Trygve Reenskaug l'any 1979. Aquest patró s'utilitza en la programació d'aplicacions webs i, com bé indica el nom, s'encarrega de separar l'aplicació en tres parts diferents: el model, la vista i el controlador.

Què és una aplicació web?

El concepte d'aplicació web és comunament confós amb el concepte de lloc web. Tant una aplicació web com un lloc web són pàgines web. El que les fa diferents és que l'aplicació web suporta interacció amb l'usuari i el lloc web no.

Així un exemple de lloc web podria ser un blog informatiu on l'usuari tan sols pot llegir les diferents entrades mentre que una aplicació web podria ser una xarxa social on l'usuari pot crear i iniciar sessió i interactuar amb els diferents usuaris.

El model

El model és la representació de la informació amb la qual el sistema opera, per tant, és ell qui gestiona tots els accessos a la dita informació tant les consultes com les actualitzacions, implementant també els privilegis d'accés que s'han descrit en les especificacions de l'aplicació (lògica de negoci).

El model envia la informació necessària a la vista. Les peticions d'accés o modificació d'informació arriben al model a través del controlador.

El controlador

Respon a esdeveniments (usualment accions de l'usuari) i invoca peticions al model quan es fa alguna sol·licitud sobre la informació (per exemple, editar un document o un registre en una base de dades). També pot enviar missatges a la seva vista associada si es sol·licita un canvi en la forma en què es presenta el model. Es podria dir que el 'controlador' fa d'intermediari entre la vista i el model.

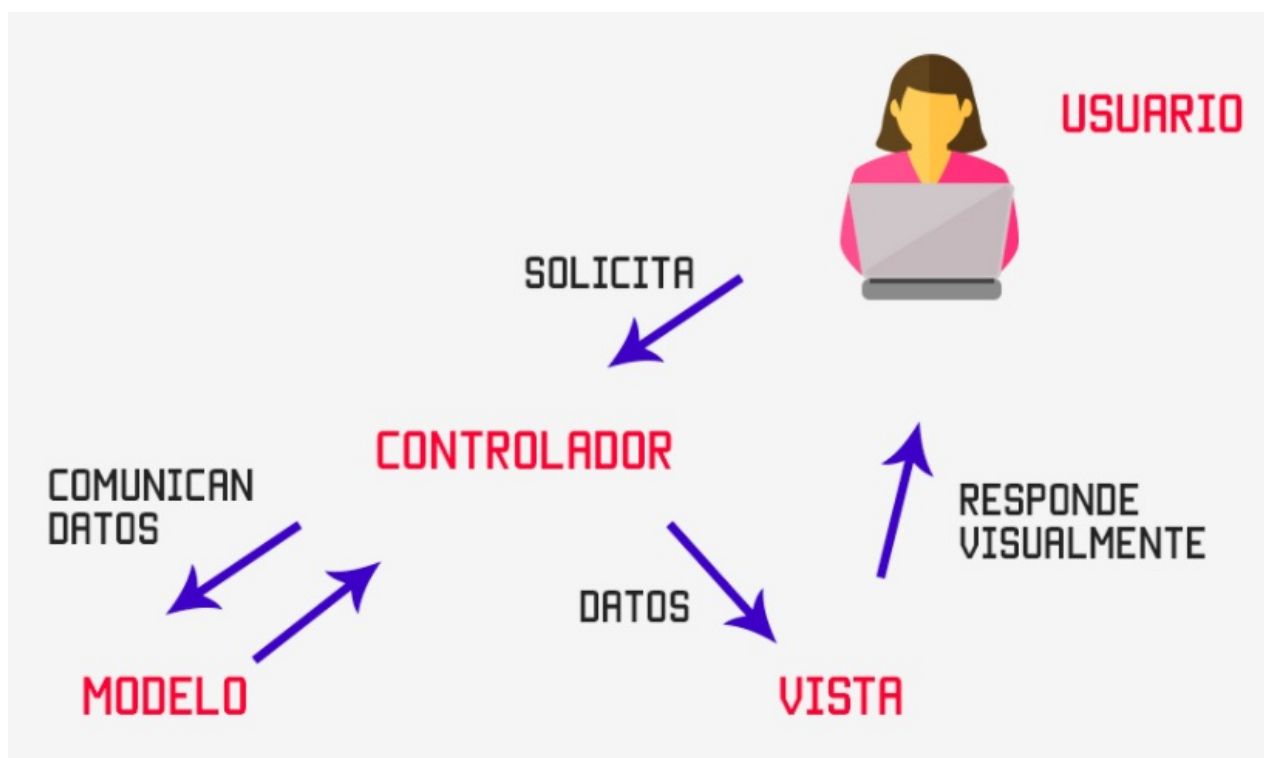
La vista

Presenta el model (informació i lògica de negoci) en un format adequat per interactuar (usualment una interfície d'usuari), per tant requereix d'aquest model la informació que ha de representar com a sortida.

Interacció entre els components

Encara que es poden trobar diferents implementacions de MVC, el flux de control que se segueix generalment és el següent:

1. L'usuari entra a la pàgina http://bibliotr.cat/iniciar_sessió i es troba amb la vista de l'aplicació.
2. L'usuari introdueix les dades necessàries (usuari i contrasenya) i prem el botó 'iniciar sessió'.
3. El controlador rep l'acció sol·licitada per l'usuari, ja que el botó ha estat pres.
4. El controlador accedeix al model i busca si la combinació usuari i contrasenya està en els registres de la base de dades d'usuaris.
5. Si la combinació resulta ser correcta, el controlador inicia la sessió de l'usuari i l'envia a la vista del perfil. Si la combinació resulta ser incorrecte, el controlador envia l'usuari a la vista d'inici de sessió i fa mostrar un missatge d'error.



Representació esquemàtica del MVC (<https://codigofacilito.com/articulos/mvc-model-view-controller-explicado>)

Flask: microframework Python

Flask és un microframework per Python creat per el programador Armin Ronacher l'any 2004. Serveix per controlar les connexions entre el servidor i el client en una aplicació web i està basat en Werkzeug i Jinja 2.

Werkzeug és una llibreria WSGI (Web Server Gateway Interface o Interfície de Porta de Servidor Web) que controla la comunicació entre el client i el servidor.

Jinja 2 és un llenguatge de plantilles per Python inspirat en Django (un altre framework per Python).

Un framework o entorn de treball és una estructura conceptual i tecnològica d'assistència definida, normalment, amb artefactes o mòduls concrets de programari, que pot servir de base per a l'organització i desenvolupament de programari.

Flask es considera un microframework i no un framework perquè la seva base és molt minimalista, només inclou les eines més essencials. Encara que la funcionalitat de Flask per defecte pot estar molt reduïda, Flask és una eina molt poderosa, doncs funciona a partir de mòduls gratuïts descarregables creats per altres usuaris que s'adapten a les diferents necessitats. Per exemple Flask inicialment no suporta el patró MVC, però amb un conjunt de mòduls el podem simular.

Algunes aplicacions famoses que utilitzen Flask són Pinterest i LinkedIn.

A continuació veurem un exemple de com funciona Flask.

Exemple d'un projecte amb Flask

{explicació}

{exemple}

{explicació}

BiblioTR

{explicació: perquè serveix?, etc}

Creació del projecte

Inici: <http://bibliotr.cat/>

{imatges}

{explicació: perquè serveix aquesta pàgina?}

{explicació: com funciona? codi, etc.}

Crear perfil: http://bibliotr.cat/crear_perfil

{imatges}

{explicació: perquè serveix aquesta pàgina?}

{explicació: com funciona? codi, etc.}

Iniciar sessió: http://www.bibliotr.cat/iniciar_sessió

{imatges}

{explicació: perquè serveix aquesta pàgina?}

{explicació: com funciona? codi, etc.}

Perfil d'un usuari: <http://www.bibliotr.cat/alexvilanovab>

{imatges}

{explicació: perquè serveix aquesta pàgina?}

{explicació: com funciona? codi, etc.}

Editar perfil: http://bibliotr.cat/editar_perfil

{imatges}

{explicació: perquè serveix aquesta pàgina?}

{explicació: com funciona? codi, etc.}

Treballs de recerca: <http://www.bibliotr.cat/treballs>

{imatges}

{explicació: perquè serveix aquesta pàgina?}

{explicació: com funciona? codi, etc.}

Verificar treballs: <http://www.bibliotr.cat/verificar>

{imatges}

{explicació: perquè serveix aquesta pàgina?}

{explicació: com funciona? codi, etc.}

Com podria millorar el projecte?

{explicació, idees}

Conclusions

{valoració del treball}

{he complert els objectius?}

{???

Bibliografia

{escriure títols en els links}

https://techterms.com/definition/computer_science

https://en.wikipedia.org/wiki/Computer_science

<https://ca.wikipedia.org/wiki/Algorisme>

https://ca.wikipedia.org/wiki/Aprenentatge_autom%C3%A0tic

https://ca.wikipedia.org/wiki/Llenguatge_de_programaci%C3%B3

<https://www.tiobe.com/tiobe-index/>

<https://ca.wikipedia.org/wiki/Python>

http://www.hartmannsoftware.com/Blog/Articles_from_Software_Fans/Most-Famous-Software-Programs-Written-in-Python

<https://docs.python.org/3/>

http://aprende-web.net/html/html12_1.php

<https://www.sqlite.org/docs.html>

<https://en.wikipedia.org/wiki/Model-view-controller>

<http://flask.pocoo.org/>

<https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world>

<https://www.youtube.com/watch?v=8aTnmsDMldY>

https://www.youtube.com/watch?v=_5OXmXvkU_E