Enginyeria de Sistemes TIC – *i*TIC

Tecnologia de la Programació
problemes de teoria

Sebastià Vila-Marta
Marta I. Tarrés-Puertas
Escola Politècnica Superior d'Enginyeria de Manresa
Universitat Politècnica de Catalunya
Versió 1: Febrer 2012
Darrera versió:

February 11, 2021

Contents

1 Consolidació curs anterior

L'objectiu d'aquests exercicis és treballar coneixements que ja teniu i progressar cap a un ús més polit del llenguatge. Així doncs heu de procurar que les solucions que proposeu siguin el més netes, simples, entenedores i elegants possible. Podeu usar qualsevol construcció vàlida del llenguatge així com qualsevol recurs de la llibreria estàndard. No oblideu de documentar les funcions i afegir-hi els corresponents doctests.

EXERCICI 1.1 Implementeu una funció tal que, donades una cadena de caràcters `s`, retorna **True** ssi `s` és palíndroma. És a dir, es llegeix igual dreta a esquerra que a l'inrevés.

EXERCICI 1.2 Implementeu una funció tal que, donat un diccionari `d` i una cadena `s` retorna **True** ssi existeix algun valor del diccionari que és una subcadena d'`s`. Podeu considerar que els valors del diccionari sempre són cadenes de caràcters.

EXERCICI 1.3 Sigui `l` una llista de 10 cadenes de caràcters. Sigui `p` una llista de 10 enters x amb $0 \leq x \leq 9$ que representa una permutació. Dissenyeu i implementeu una funció tal que que donats dos paràmetres com `l` i `p` retorna la llista `l` permutada d'acord amb el que indica `p`.

EXERCICI 1.4 Sigui `p` una paraula. Definiu la funcio `desplega(p)` tal que donada una paraula `p` retorna la llista de tots els prefixos de `p`. Per exemple, si `p='hola'` llavors `desplega(p)` ha de tornar la llista `['h', 'ho', 'hol', 'hola']`.

EXERCICI 1.5 Crea la funció en Python, stringToBinary, tal que donat un string corresponent a un nombre binari correcte, retorni el nombre enter al que correspon. Nota: no es pot utilitzar cap funció predefinida de python.

```
def stringToBinary(s):
    """
    retorna enter del binari s
    >>> stringToBinary('0011')
    3
    >>> stringToBinary('1111')
    15
    """
```

Exercici 1.6 Crea la funció en Python, intToBinary, tal que donat un nombre enter positiu, retorni un string corresponent a la conversió a nombre binari. Nota: no es pot utilitzar cap funció predefinida de python.

```
def intToBinary(s):
    """
    retorna nombre binari de enter positiu s
    >>> intToBinary(3)
    '11'
    >>> intToBinary(14)
    '1110'
    >>> intToBinary(0)
    '0'
    >>> intToBinary(1)
    '1'
    """
```

Exercici 1.7 Estudia els operadors a nivell de bit, bitwise operators de Python, i en l'exercici 1.6 i en l'exercici 1.5, substitueix les multiplicacions i divisions per aquests operadors. Exemples d'ús:

- $n * pow(2, 1)$ equival a $n << 1$, i
- $n/pow(2, 1)$ equival a $n >> 1$

Exercici 1.8 Escriu quin serà el resultat de cadascuna de les següents expressions.

```
>>> 32>>5
#TO DO
>>> 166&2**5
#TO DO
>>> 166&1>>0
#TO DO
>>> 4|1<<4
# TO DO
>>> 4&~(1<<2)
#TO DO
```

2 Classes, atributs i mètodes

Exercici 2.1 Cerqueu a la wikipèdia el terme anglès *programming paradigm*. Estudieu-ne el contingut i resumiu en dos o tres paràgrafs què significa aquest concepte.

Exercici 2.2 Definiu la classe Semaf que representa un semàfor de carrer, amb els tres colors habituals. Quants atributs hauria de tenir? Definiu un mètode per inicialitzar instàncies d'aquesta classe que permeti definir el color en que es troba el semàfor just després de crear-lo.

Exercici 2.3 Definiu la classe Vector2d que representa un vector en el pla. Doteu la classe d'un constructor i també d'un mètode que retorni el mòdul del vector en qüestió.

Exercici 2.4 Definiu la classe Fraccio. Creeu un constructor o inicialitzador que instancii un objecte d'aquesta classe donant el numerador i el denominador. Afegiu-hi un mètode que permeti sumar una fracció amb una altra.

Exercici 2.5 Dissenyeu la classe Byte. Un byte és una paraula de 8 bits. Representeu un byte com un tuple de 8 enters, que prendran sempre valor 0 o 1. Definiu un mètode constructor i també un altre mètode que retorna el valor enter corresponent al byte.

Aquesta classe hauria de superar el següent doctest:

```
>>> x = Byte((0,0,0,0,1,0,0,1))
>>> print(x.value())
9
```

Exercici 2.6 Escriviu els missatges que es mostren per pantalla en l'execució del següent script.

```
class Robot(object):
    def __init__(self, name, build_year):
        self.__name = name
        self.setBuildYear(build_year)

    def SayHello(self):
        print("Hi, I'm " + self.__name)

    def setBuildYear(self, build_year):
        self.__build_year = build_year

    def getBuildYear(self):
        return str(self.__build_year)
```

```
    def getName(self):
        return self.__name

    def __repr__(self):
        return "Robot(" + self.getName() + "," + str(self.__build_year) + ")"

if __name__ == "__main__":
    x = Robot("Marvin", 1979)
    y = Robot("Caliban", 1993)
    for rob in [x, y]:
        rob.SayHello()
        print("I was built in the year " + rob.getBuildYear() + "!")
    print y
```

EXERCICI 2.7 Les resistències es marquen amb unes bandes de color per indicar el seu valor nominal i la seva tolerància. En aquesta referència trobareu els detalls d'aquesta codificació: `http://en.wikipedia.org/wiki/Electronic_color_code`.

Es demana que definiu la classe Codif que ha de representar la codificació d'una resistència. A tal efecte seguiu les següents indicacions:

- Codifiqueu cada color amb l'enter que representa. Per exemple, el 0 representa el color negre, el 1 representa el color marró, etc.
- Els objectes de la classe Codif han de tenir tres atributs cadascun dels quals representa una banda de color.
- La classe ha de tenir un constructor al que cal donar els codis corresponents a les tres bandes.
- La classe ha d'estar dotada d'un mètode value que retorni el valor en Ohms corresponent al codi.
- La classe ha d'estar dotada d'un mètode colors que retorni un cadena corresponent als colors del codi.

La classe, una vegada acabada, hauria de poder superar el següent doctest:

```
>>> r = Codif(1,2,5)
>>> print(r.value())
1200000
>>> print(r.colors())
Marro, Vermell, Verd
```

3 Composició i information hiding

Exercici 3.1 Considerar els atributs d'un objecte privats és, segons el principi d'information hiding, quelcom beneficiós. Això no obstant, hi ha casos en que no és necessària aquesta prevenció. En quines circumstàncies no és necessari considerar els atributs d'un objecte privats? Podries donar-ne un exemple?

Exercici 3.2 Dissenya i implementa un nou mètode per a la classe Wallet. Aquest mètode ha de tenir un paràmetre enter x i ha de retornar **True** ssi en el moneder hi ha prou canvi com per pagar x EUR de manera exacta.

Exercici 3.3 Dissenya i implementa un nou modificador per a la classe Wallet. Aquest modificador ha de tenir un paràmetre enter x i ha de:

1. Decrementar x EUR del moneder emprant la menor quantitat de moneda possible.
2. Retornar un tuple amb la quantitat de cada tipus de moneda que ha descomptat.

Noteu que aquesta operació no es pot fer sempre atès que el moneder podria contenir menys moneda de la necessària. Cal comprovar-ho i tenir-ho en compte.

Exercici 3.4 Dissenyeu i implementeu la classe Cotxe. Considereu que els atributs d'un cotxe són:

- La matrícula.
- El model.
- El color.
- El tipus de motor: benzina o dièsel.

Decidiu si els atributs han de tenir natura privada i quins són els mètodes més escaients per a aquesta classe.

Exercici 3.5 Dissenyeu i implementeu la classe Garatge. Un garatge és una col.lecció d'instàncies de Cotxe. Representeu aquesta col.lecció amb un atribut de tipus llista. Afegiu-hi mètodes per:

- Afegir un cotxe al garatge.
- Treure un cotxe d'una matrícula concreta del garatge.
- Calcular l'ocupació del garatge.
- Treure un cotxe arbitrari del garatge.

Tingueu en compte que en el garatge no poden haver-hi dos cotxes amb la mateixa matrícula.

Diríeu que la relació entre una instància Garatge i una instància Cotxe és de composició? Per què?

EXERCICI 3.6 Amplieu la classe Cotxe amb dos mètodes més. Aquests mètodes han de permetre transformar un cotxe en una cadena de caràcters i viceversa.

El primer mètode l'anomeneu tostring i ha de retornar una cadena que contingui les dades de la instància de cotxe. El segon l'anomeneu fromstring(s) i ha de modificar la instància de cotxe d'acord amb les dades que conté s.

Si implementeu com cal els mètodes, el següent doctest hauria de ser correcte:

```
>>> c = Cotxe('2331 DXC', 'VW Polo', 'vermell', True)
>>> c_string = c.tostring()
>>> b = Cotxe()
>>> b.fromstring(c_string)
>>> c == b
True
```

EXERCICI 3.7 Amplieu de nou la classe Cotxe. En aquest cas cal que hi afegiu dos mètodes més que permeten emmagatzemar i recuperar una instància d'un fitxer. A tal efecte cal usar els mètodes tostring i fromstring dissenyats en l'exercici 3.6.

Dissenyeu primer un mètode desa(f) que emmagatzema una representació en forma de cadena de caràcters de la instància en el fitxer f. f ha d'estar obert en mode escriptura. Un ús típic fora el següent:

```
>>> c = Cotxe('2331 DXC', 'VW Polo', 'vermell', True)
>>> f = open("cotxe.dat", "w")
>>> c.desa(f)
>>> f.close()
```

A continuació dissenyeu un mètode recupera(f) tal que donat un fitxer f obert en mode lectura i que contingui una representació d'una instància de cotxe, la recupera. Un ús típic, considerant que el fitxer `cotxe.dat` és el de l'exemple anterior, seria el següent:

```
>>> c = Cotxe()
>>> f = open("cotxe.dat", "r")
>>> c.recupera(f)
>>> f.close()
```

Si la implementació és correcta, hauríeu de poder executar el següent doctest:

```
>>> c = Cotxe('2331 DXC', 'VW Polo', 'vermell', True)
>>> f = open("cotxe.dat", "w")
>>> c.desa(f)
>>> f.close()
>>> d = Cotxe()
>>> f = open("cotxe.dat", "r")
>>> d.recupera(f)
>>> f.close()
>>> c == d
True
```

EXERCICI 3.8 Amplieu la classe Garatge de l'exercici 3.5 amb dos mètodes més que permetin emmagatzemar i recuperar un garatge d'un fitxer de text. A tal efecte cal que useu els mètodes de Cotxe implementats a l'exercici 3.7. Documenteu i afegiu els doctests escaients a la classe Garatge.

EXERCICI 3.9 Dissenyeu la classe Component que modela un component electrònic. Aquesta classe representa un component enregistrant els següents atributs:

preu És un **float** i representa el preu en euros del component.

pes És el pes del component en grams.

id És l'identificador del component, del tipus **str**. S'usa per descriure el component quan es fan llistats.

EXERCICI 3.10 Dissenyeu la classe Circuit. Una instància de Circuit modela un montatge real d'un circuit electrònic format per una col.lecció de components, és a dir, instàncies de Component tal i com s'ha definit a l'exercici 3.9.

Doteu aquesta classe dels següents mètodes:

afegir(self,c) Afegeix el component c al circuit self.

preu(self) Retorna el preu amb euros del circuit. Aquest preu es calcula a base de sumar el preu de cadascun dels components del circuit.

pes(self) Retorna el pes en grams del circuit. Aquest pes es calcula a base de suamr el pes de cadascun dels components del circuit.

num_elem(self) Retorna el nombre de components del circuit.

Afegiu els doctests necessaris per garantir la correctesa de la vostra implementació.

EXERCICI 3.11 Amplieu la classe Circuit de l'exercici 3.10 i afegiu el següent mètode:

llista(self) Aquest mètode retorna una llista dels components del circuit ordenada per identificador del component.

EXERCICI 3.12 El missatge SMS rebut. Suposeu la classe Missatge, que correspon, de manera simplificada, a un missatge rebut amb les següents atributs: *emailOrigen*, *textMissatge* i un atribut que permeti emmagatzemar si el missatge ja ha estat vist de nom *viewed*. Aquests atributs han de ser privats.

[Apartat a] Creeu la classe amb el mètode constructor. Afegiu-hi el valor per omissió per a l'atribut *viewed* a *False*.

[Apartat b] Afegiu-hi els mètodes accessors/modificadors.

[Apartat c] Afegiu-hi a la classe *Missatge* els mètodes necessaris per superar satisfactòriament els següents doctests:

```
>>> m1=Missatge("joan@gmail.cat","avui arribo tard")
>>> m2=Missatge("anna@gmail.com","hem aconseguit la beca!!")
>>> m3=Missatge("joan@gmail.cat","avui arribo tard")
>>> print m1
Message information joan@gmail.cat avui arribo tard
>>> print "m1 == m2 ??",m1==m2
m1 == m2 ?? False
>>> print "m1 == m3 ??",m1==m3
m1 == m3 ?? True
```

Exercici 3.13 La Bústia de missatges rebuts.

Seguint l'exercici 3.12, ara cal implementar la classe *Mailbox* que simula una col·lecció de missatges rebuts.

[Apartat a] Dissenyeu la classe Mailbox i el seu constructor.

[Apartat b] Afegiu-hi el mètode *addNewArrival*, que permet crear una instància objecte de missatge i afegir-la a la col·lecció de missatges de la classe *Mailbox*. Per simplificar suposarem que no arriben missatges repetits.

[Apartat c] Afegiu-hi el mètode *getMessage*, que permet mostrar la informació d'un missatge d'una posició concreta de la col·lecció de missatges. Si la posició existeix, també ha de canviar l'atribut *viewed* del missatge en qüestió a *True*. En cas que la posició sigui errònia, ha de mostrar l'error corresponent.

[Apartat d] Afegiu-hi el mètode *sortByMail*, que permet obtenir la informació dels missatges ordenada per *emailOrigen*, i canvia automàticament l'atribut *viewed* de tots els missatges a *True*.

A continuació segueix un exemple de funcionament de la classe *Mailbox*.

```
>>> bustia=Mailbox()
>>> bustia.addNewArrival("joan@gmail.com","notes entrades")
>>> bustia.addNewArrival("alba@upc.edu","dubtes classe tecpro")
>>> bustia.addNewArrival("jordi@ucp.edu","quin es l'horari de consultes rebut?")
>>> bustia.getMessage(2)
Message information jordi@ucp.edu quin es l'horari de consultes rebut?
>>> bustia.sortByMail()
Message information alba@upc.edu dubtes classe tecpro
Message information joan@gmail.com notes entrades
Message information jordi@ucp.edu quin es l'horari de consultes rebut?
```

Exercici 3.14 El Referèndum.

L'objectiu del problema a resoldre consisteix a desenvolupar un sistema que gestioni les votacions de candidats en diferents regions. Suposant la següent definició de la classe Referendum, dissenyeu els mètodes que manquen i els que permetin el correcte funcionament del joc de proves que segueix.

```
class Referendum(object):
    def __init__(self):
        self.regions=['r1','r2','r3']
        self.candidats=['A','B','C']
        self.votsRegions={}
```

```
    def afegirVots(self,c,v,r):
        if not r in self.votsRegions:
            self.votsRegions[r]={c:v}
        else:
            if not c in self.votsRegions[r]:
                self.votsRegions[r][c]=v
            else:
                self.votsRegions[r][c]+=v

    def votsCandidatRegions(self,c):
        """
        retorna el nombre total de vots pel candidat c, en totes les regions
        """
        #TO DO

    def mesVotatRegio(self,r):
        """
        retorna el candidat mes votat d'una regio
        """
        #TO DO

    def llistaRegions(self,c):
        """
        retorna un llistat amb les regions on ha guanyat el candidat
        """
        #TO DO

if __name__=='__main__':
    v=Referendum()
    v.afegirVots('A',100,'r1')
    v.afegirVots('A',55,'r2')
    v.afegirVots('A',150,'r3')
    v.afegirVots('C',200,'r1')
    v.afegirVots('B',50,'r2')
    v.afegirVots('C',120,'r3')
    print v
    print v.votsCandidatRegions('A')
    print v.mesVotatRegio('r3')
    print v.llistaRegions('A')
```

El resultat de l'execució és el que segueix.

```
r1 {'A': 100, 'C': 200}
r2 {'A': 55, 'B': 50}
r3 {'A': 150, 'C': 120}
305
A
['r2', 'r3']
```

EXERCICI 3.15 Suposeu la classe Contacte, que correspon, de manera simplificada, a un contacte de mòbil amb els següents atributs: *nick*, *email* i un atribut *telefons* que permeti emmagatzemar un nombre indeterminat de telèfons en una llista. Aquest darrer atribut ha de ser privat.

Creeu la classe amb el mètode constructor i afegiu a la classe *Contacte* només els mètodes necessaris per superar satisfactòriament el següent joc de proves.

```
>>> p=Contacte("juliaR","juliaRoberts@gmail.com")
>>> p.addTelefon("938888888")
>>> p.addTelefon("937777777")
>>> r=Contacte("bradP","bradPitt@gmail.com")
>>> r.addTelefon("666666666")
>>> r.addTelefon("666666666")
Phone already added
>>> s=Contacte("bonJ","bonJovi@gmail.com")
>>> p==r
False
>>> print p
Nick: juliaR Email: juliaRoberts@gmail.com Telefons : 938888888 937777777
>>> print s
Nick: bonJ Email: bonJovi@gmail.com Telefons : No phones yet
```

EXERCICI 3.16 La classe Agenda simula un contenidor de n contactes. Completeu els mètodes de les classes que es requereixen a continuació, i que permetran la correcta execució del main.

```
class Agenda(object):
    def __init__(self):
        self.c=[]

    def __iter__(self):
        #TO DO

    def __len__(self):
        #TO DO

    def add(self,c):
        #TO DO

    def __getitem__(self,key):
        #TO DO

    def __setitem__(self,key,value):
        #TO DO

    def __delitem__(self,key):
        #TO DO

    def __str__(self):
        #TO DO
```

```
    def ordena(self):
        #TO DO

if __name__=='__main__':
    >>> a=Agenda()
    >>> for i in range(3):
            a.add(Contacte(nick=raw_input("Entri nick: "),email=raw_input("Entri email: ")))
    Entri nick: shakira
    Entri email: shak@gmail.com
    Entri nick: ladygaga
    Entri email: ladyg@gmail.com
    Entri nick: justinB
    Entri email: justinB@gmail.com
    >>> print a
    ContactsList
    Nick: shakira Email: shak@gmail.com Telefons : No phones yet
    Nick: ladygaga Email: ladyg@gmail.com Telefons : No phones yet
    Nick: justinB Email: justinB@gmail.com Telefons : No phones yet
    >>> print a.ordena()
    Ordered contactsList
    Nick: justinB Email: justinB@gmail.com Telefons : No phones yet
    Nick: ladygaga Email: ladyg@gmail.com Telefons : No phones yet
    Nick: shakira Email: shak@gmail.com Telefons : No phones yet
    >>> print len(a)
    3
    >>> a[0].addTelefon("666666666")
    >>> a[0].addTelefon("666666667")
    >>> print a[0]
    Nick: shakira Email: shak@gmail.com Telefons : 666666666 666666667
    >>> a[1]=Contacte(nick="Perfecto")
    >>> del a[0]
    >>> print a
    Contacts list
    Nick: Perfecto Email: Telefons : No phones yet
    Nick: justinB Email: justinB@gmail.com Telefons : No phones yet
```

Exercici 3.17 La codificació Run-length.

Run-length encoding, RLE, és una forma molt simple de compressió de dades en què seqüències de dades amb el mateix valor consecutiu són emmagatzemades com un únic valor més el seu recompte. Per exemple, si es considera una pantalla que conté text en negre sobre un fons blanc. Hi ha moltes seqüències d'aquest tipus amb píxels blancs en els marges buits, i altres seqüències de píxels negres a la zona del text. Suposant una línia (o scanline), on N representa les zones en negre i B les de blanc:

BBBBBBBBBBBBNBBBBBBBBBBBBNNNBBBBBBBBBBBBBBBBBBBBBBBBNBBBBBBBBBBBBBB

Si s'aplica la codificació run-length a aquesta línia, s'obtindrà el següent:

12B1N12B3N24B1N14B

Fixeu-vos que en la seqüència resultant, la primera dada correspon al nombre de vegades que el caràcter que ve a continuació està repetit.

L'objectiu del problema és dissenyar una classe de nom RLE que permeti representar i manipular seqüències RLE. En concret, donat el següent esquelet de la classe RLE, es demana implementar els mètodes *encode* i *decode* i els mètodes necessaris per tal que el funcionament del joc de proves sigui l'esperat.

```
class RLE(object):

    def __init__(self,seq):
        self.releSeq=seq
        self.encode()

    def encode(self):
        """
        saves the RLE-encoded string
        """
        #TO DO

    def decode(self):
        """
        returns the string corresponding to the RLE-encoded string for the class instance
        """
        #TO DO

if __name__=='__main__':
    a='BBBBBBBBBBBNBBBBBBBBBB'
    r=RLE(a)
    print "codificacio", r
    print "decodificacio", r.decode()
```

El resultat de l'execució és el que segueix.

```
codificacio 12B1N11B
decodificacio BBBBBBBBBBBNBBBBBBBBBB
```

Exercici 3.18 Escriviu els missatges que es mostren per pantalla en l'execució dels scripts següents. Apartat a)

```
class A(object):
    colour='gold'
    def showA(self):
        return 'A modern '+A.colour+' object'
    def describeA(self):
        return 'A modern '+self.colour+' object'
```

```
if __name__=='__main__':
    objecta=A()
    print objecta.showA()
    print objecta.colour
    objectb=A()
    objectb.colour='plate'
    print objectb.showA()
    print objectb.describeSelf()
    print ojbecta.describeA()
```

Apartat b)

```
class B(object):
    weight=1000
    def __init__(self,weight, driver):
        self.weight=weight
        self.driver=driver
class C(object):
    weight=1000
    def __init__(self,weight):
        self.__info=weight

if __name__=='__main__':
    classeB=B(2000,100)
    classeC=classeB
    print classeB==classeC
    classeD=B(2000,100)
    classeE=C(1000)
    print classeB==classeD
    print classeB.weight==B.weight+1000
    print C.weight==B.weight
    print C.weight==classeE.__info
```

La WebPage. Suposeu la classe *WebPage*, que correspon, de manera simplificada, a un pàgina web els següents atributs: *url*, *títol* i un atribut que permeti emmagatzemar les urls de pàgines web a les que enllaça, *links*. Aquest darrer atribut ha de ser privat.

[Apartat a] Creeu la classe amb el mètode constructor.

[Apartat b] Afegiu a la classe *WebPage* els mètodes necessaris per superar satisfactòriament el següent joc de proves.

```
>>> p=WebPage("http://www.epsem.upc.edu","EPSEM-UPC-Page")
>>> p.addLink("http://www.upc.edu")
>>> p.addLink("http://www.google.es")
>>> r=WebPage("http://www.upc.edu","UPC-Page")
>>> r.addLink("http://www.epsem.upc.edu")
>>> p.containedin(r)
EPSEM-UPC-Page links to UPC-Page
>>> print p.numberofLinks()
EPSEM-UPC-Page has 2 links
```

EXERCICI 3.19 La classe Vector i la classe Space. La classe *Vector* simula un vector de n components. I la classe *Espai* correspon a un contenidor de Vectors. Completeu els mètodes de les classes que es requereixen a continuació, i que permetran la correcta execució del main.

[Apartat a]

```
class Vector(object):
    def __init__(self,components=[]):
        TODO
    def __iter__(self):
        TODO
    def __len__(self):
        TODO
    def __getitem__(self,k):
        TODO
    def __add__(self,other):
        TODO
    def __str__(self):
        TODO
    def __eq__(self,other):
        TODO
    def __mul__(self,other):
        TODO

class Espai(object):
    def __init__(self):
        TODO
    def __setitem__(self,k,v):
        TODO
    def __iter__(self):
        TODO
    def __str__(self):
        TODO

if __name__=='__main__':
    v=Vector([2,4])
    print v
    for c in v:
        print c*2
    h=Vector([1,3])
    print v+h
    print v==h
    print v*4
    j=Espai()
    j[0]=v
    j[1]=h
    print j
```

EXERCICI 3.20 La classe *Videoclub* simula un contenidor de n pel·lícules. Completeu els mètodes de les classes que es requereixen a continuació i que permetran els resultats esperats d'execució.

```
class Videoclub(object):
    def __init__(self):
        self.pelis=[]
    def add(self,peli):
       // TO DO_1
    def __len__(self):
        return len(self.pelis)
    def __getitem__(self,p):
       // TO DO_2
    def __iter__(self):
       // TO DO_3
    def __str__(self):
        // TO DO_4
    def llistaPelisLlargues(self):
        """
        retorna la llista de noms de pelis
        amb durada superior a 120
        """
        // TO DO_5
    def ordena(self):
        """
        mostra el llistat ordenat de pelis
        per durada en ordre descendent
        """
        // TO DO_6
```

```
class Peli(object):
    def __init__(self,titol,any,d=0,g=""):
        self.__titol=titol
        self.__any=any
        self.__durada=d
        self.__genere=g
    def getTitol(self):
        return self.__titol
    def getDurada(self):
        return self.__durada
    def __eq__(self,other):
        // TO DO_7
    def __str__(self):
        return "["+self.__titol+" "+str(self.__durada)+
        " "+self.__genere+"]"
```

A continuació segueix el joc de proves (esquerra) i els resultats d'execució esperats (dreta)

```
if __name__=='__main__':
    v=Videoclub()
    p=Peli("Mechanic: Resurrection",2016,99,
"Thriller")
    p1=Peli("100 metros",2016,108,"Drama")
    p2=Peli("Miss Saigon",2016,195,"Musical")
    p3=Peli("The Right Kind of Wrong",2013,97,
"Love")
    p4=Peli("The Magnificent Seven",2016,132,
"Western")
    p5=Peli(titol="Mechanic: Resurrection",
any=2016)
    print p==p5
    print p2
    v.add(p)
    v.add(p1)
    v.add(p2)
    v.add(p3)
    v.add(p4)
    v.add(p5)
    print v
    print v.llistaPelisLlargues()
    print v[3]
    print "Quantes pelis videoclub",len(v)
    v.ordena()
```

```
True
[Miss Saigon 195 Musical]
not added repeated peli

Llistat pelis
[Mechanic: Resurrection 99 Thriller]
[100 metros 108 Drama]
[Miss Saigon 195 Musical]
[The Right Kind of Wrong 97 Love]
[The Magnificent Seven 132 Western]

List of pelis >120
['Miss Saigon', 'The Magnificent Seven']

[The Right Kind of Wrong 97 Love]

Quantes pelis videoclub 5

Ordered pelis by durada
[Miss Saigon 195 Musical]
[The Magnificent Seven 132 Western]
[100 metros 108 Drama]
[Mechanic: Resurrection 99 Thriller]
[The Right Kind of Wrong 97 Love]
```

EXERCICI 3.21 BankTIC ens demana la gestió dels seus clients i dels números de comptes dels clients. Noteu que cal accedir de manera òptima als clients del banc. Per tant, la clau d'accés dels clients serà el seu dni. Adicionalment, donat un client, cal accedir de manera òptima als comptes d'un client. Per tant, la clau d'accés serà el número de compte.

A continuació segueix l'esquelet de classes proporcionat. Se us demanana implementar **els mètodes que calguin** per tal de que l'execució proporcionada sigui correcta. Se us proporcionen les capçaleres d'alguns dels mètodes necessaris.

```
class Banc(object):
    def __init__(self,n):
        self.__nom=n
        self.clients={}
    def add(self,client):
        if client.getDni() not in self:
            self.clients[client.getDni()]=client
        else:
            print "not added client"
    def __getitem__(self,numero):
       // TO DO_8
    def __str__(self):
        // TO DO_9
    def checkMorosos(self):
        """
        retorna una llista de tuples
        (dni_client,numero_compte) dels
        clients amb numero_compte < 0
        """
        //TO DO_10


if __name__=='__main__':
    b=Banc("BankTIC")
    c1=Client("22222222A")
    c2=Client("33333333B")
    b.add(c1)
    b.add(c2)
    print "Resultat 1 ------------------"
    print b
    co=Compte("1111111111",10)
    co1=Compte("2222222222",20)
    co2=Compte("3333333333")
    co3=Compte("4444444444",100)
    c1.add(co1)
    c1.add(co3)
    c2.add(co2)
    print "Resultat 2 ------------------"
    print c1["2222222222"]
    print "Resultat 3 ------------------"
    print b["22222222A"]
    print "Resultat 4 ------------------"
    print "Comptes del client",c1.getDni(),
    print "Numero comptes:",len(c1)
    c1["2222222222"]=-300
    c1["4444444444"]=-300
    print "Resultat 5 ------------------"
    print c1
    print "Resultat 6 ------------------"
    print b
```

```
class Client(object):
    def __init__(self,dni):
        self.__dni=dni
        self.comptes={}
    def add(self,compte):
        if compte.getNumero() not in self:
            self.comptes[compte.getNumero()]=compte
        else:
            print "already existing count for client"
    def __getitem__(self,numero):
        // TO DO_11
    def __setitem__(self,numero,nousaldo):
        // TO DO_12
    def __str__(self):
        // TO DO_13

class Compte(object):
    def __init__(self,numero,saldo=0):
        self.__numero=numero
        self.__saldo=saldo


    print "Resultat 7 ------------------"
    print b.checkMorosos()
```

Resultats d'execució esperats

```
Resultat 1 ------------------
BankTIC
Llistat clients
Dni client:  22222222A
Client sense comptes

Dni client:  33333333B
Client sense comptes

Resultat 2 ------------------
2222222222 Saldo 20

Resultat 3 ------------------
Dni client:  22222222A
Llistat de comptes
4444444444 Saldo 100
2222222222 Saldo 20

Resultat 4 ------------------
Comptes del client 22222222A Numero comptes: 2
```

```
Resultat 5 ------------------
Dni client:  22222222A
Llistat de comptes
4444444444 Saldo -200
2222222222 Saldo -280

Resultat 6 ------------------
BankTIC
Llistat clients
Dni client:  22222222A
Llistat de comptes
4444444444 Saldo -200
2222222222 Saldo -280

Dni client:  33333333B
Llistat de comptes
3333333333 Saldo 0

Resultat 7 ------------------
Llistat morosos
[('22222222A', '4444444444'), ('22222222A', '2222222222')]
```

EXERCICI 3.22 L'intèrpret de classes. Escriviu els missatges que es mostren per pantalla en l'execució dels scripts següents.

```
#Apartat a)
class A(object):
    colour='gold'
    def showA(self):
        return 'A modern '+A.colour+' object'
    def describeA(self):
        return 'A modern '+self.colour+' object'

if __name__=='__main__':
    objecta=A()
    print objecta.showA()
    print objecta.colour
    objectb=A()
    objectb.colour='plate'
    print objectb.showA()
    print objectb.describeSelf()
    print objecta.describeA()

#Apartat b)
class B(object):
    weight=1000
    def __init__(self,weight, driver):
        self.weight=weight
        self.driver=driver
class C(object):
    weight=1000
    def __init__(self,weight):
        self.__info=weight

if __name__=='__main__':
    classeB=B(2000,100)
    classeC=classeB
    print classeB==classeC
    classeD=B(2000,100)
    classeE=C(1000)
    print classeB==classeD
```

```
print classeB.weight==B.weight+1000
print C.weight==B.weight
print C.weight==classeE.__info
```

EXERCICI 3.23 PlataformaTIC és una eina que ajuda a establir amistats entre usuaris. Coneixedors de que sou professionals TIC experts en la matèria, se us demana la gestió dels seus usuaris i un registre dels usuaris que són amics entre ells. A tal, efecte se us passa l'esquelet de classes amb els mètodes involucrats, i un joc de proves del funcionament esperat. Se us demanana implementar **els mètodes que calguin** per tal de que l'execució proporcionada sigui correcta.

```
class Plataforma(object):
    def __init__(self):
        self.usuaris=[]
    def add(self,usuari):
        #TODO_EX2_1
    def addFriend(self,usuari1,usuari2):
        #TODO_EX2_2
    def __getitem__(self,p):
        #TO DO_EX2_3
    def __iter__(self):
        #TO_DO_EX2_4
    def __str__(self):
        #TO_DO_EX2_5
    def llistaUsuarisFantasma(self):
        """
        obte la llista de emails d'usuaris sense amics
        """
         #TO_DO_EX2_6
    def ordena(self):
        """
        mostra el llistat ordenat de usuaris per email
        en ordre descendent
        """
        #TO_DO_EX2_7

class Usuari(object):
    def __init__(self,email,any=0):
        self.__email=email
        self.__any=any
        self.amics=[]
    def __iter__(self):
        return iter(self.amics)
    def getEmail(self):
        return self.__email
    def getAny(self):
        return self.__any
    def __eq__(self,other):
         #TO_DO_EX2_8
    def __len__(self):
        return len(self.amics)
    def __str__(self):
        """
        >>> u1=Usuari("madonna@gmail.com",1968)
        >>> print u1
        [madonna@gmail.com 50] No friends yet
        """
         #TO_DO_EX2_9
    def addFriend(self,user):
        """
        >>> u1=Usuari("madonna@gmail.com",1968)
        >>> u2=Usuari("bonjovi@gmail.com",1951)
        >>> u1.addFriend(u2)
        >>> print u1
        [madonna@gmail.com 50] Friends:bonjovi@gmail.com
        >>> print u2
        [bonjovi@gmail.com 67] Friends:madonna@gmail.com
        >>> u1.addFriend(u2)
        OOps Friendship was already done...
        bonjovi@gmail.com madonna@gmail.com
        """
        #TO_DO_EX2_10
```

```
if __name__=='__main__':
    p=Plataforma()
    u1=Usuari("madonna@gmail.com",1968)
    u2=Usuari("bonjovi@gmail.com",1951)
    u3=Usuari("shakira@gmail.com",1976)
    u4=Usuari("edshaaran@gmail.com",1980)
    u5=Usuari("fantasma@gmail.com",1999)
    u6=Usuari("pepet@gmail.com",1988)
    p.add(u1)
    p.add(u2)
    p.add(u3)
    p.add(u4)
    p.add(u6)
    print p[2] #Crida 1
    p.addFriend(u1,u2)
    p.addFriend(u1,u3)
    p.addFriend(u1,u4)
    p.addFriend(u2,u3)
    p.addFriend(u5,u4) #Crida 2
    p.addFriend(u1,u2) #Crida 3
    print p #Crida 4
    print p.llistaUsuarisFantasma() #Crida 5
    p.ordena()#Crida 6

#Resultat Crida 1
[shakira@gmail.com 42] No friends yet

#Resultat Crida 2
Users not allowed fantasma@gmail.com edshaaran@gmail.com

#Resultat Crida 3
OOps Friendship was already done...
bonjovi@gmail.com madonna@gmail.com

#Resultat Crida 4
Users:
[madonna@gmail.com 50]
Friends:bonjovi@gmail.com,shakira@gmail.com,edshaaran@gmail.com
[bonjovi@gmail.com 67]
Friends:madonna@gmail.com,shakira@gmail.com
[shakira@gmail.com 42]
Friends:madonna@gmail.com,bonjovi@gmail.com
[edshaaran@gmail.com 38]
Friends:madonna@gmail.com
[pepet@gmail.com 30] No friends yet

#Resultat Crida 5
Ghost users:
[pepet@gmail.com 30] No friends yet

#Resultat Crida 6
Ordered users by email
[shakira@gmail.com 42]
Friends:madonna@gmail.com,bonjovi@gmail.com
[pepet@gmail.com 30] No friends yet
[madonna@gmail.com 50]
Friends:bonjovi@gmail.com,shakira@gmail.com,edshaaran@gmail.com
[edshaaran@gmail.com 38]
Friends:madonna@gmail.com
[bonjovi@gmail.com 67]
Friends:madonna@gmail.com,shakira@gmail.com
```

EXERCICI 3.24 Segur que coneixeu el joc del "tres en ratlla". En aquest exercici es demana que dissenyeu i implementeu una classe per representar el tauler de joc del "tres en ratlla" que anomenareu TresER. Aquesta classe assumirà que:

- El tauler de joc té forma matricial amb 3 files i 3 columnes numerades de 0 a 2.
- Té les caselles identificades per les seves coordenades.
- Els dos jugadors s'identifiquen amb els números 0 i 1 respectivament.
- La classe també controla les fitxes que els jugadors no tenen en el taulell.

La classe ha de tenir els següents mètodes:

- __init__(self)

  Crea un tauler inicialitzat de forma que cada jugador té tres fitxes disponibles i cap d'elles és al tauler.

- posa_fitxa(self, jugador, casella)

  Posa la fitxa del jugador jugador en la casella casella. casella és un tuple de coordenades. Si el jugador no té fitxes disponibles o la casella ja estava ocupada, aixeca una excepció.

- treu_fitxa(self, casella)

  Treu la fitxa de la casella casella i la torna al jugador corresponent. casella és un tuple de coordenades. Si la casella no contenia cap fitxa, aixeca una excepció.

- tres_en_ratlla(self, jugador)

  Retorna **True** si el jugador jugador té tres fitxes en ratlla en el taulell.

- disponibles(self, jugador)

  Retorna el nombre de fitxes de que un jugador disposa per posar al taulell.

La classe hauria de superar un doctest com el següent:

```
>>> t = TresER()
>>> t.disponibles(0)
3
>>> t.posa_fitxa(0, (1,1))
>>> t.posa_fitxa(0, (2,2))
>>> t.disponibles(0)
1
>>> t.posa_fitxa(0, (1,2))
>>> t.tres_en_ratlla(0)
False
>>> t.treu_fitxa( (1,2) )
>>> t.posa_fitxa(0, (0,0))
>>> t.tres_en_ratlla(0)
True
```

Solució exercici 3.24 En aquest problema el primer que cal és determinar com representarem el concepte de tauler. Hi ha diverses possibilitats. Una de les més senzilles és representar una tauler associant a cada jugador la llista de les caselles on té col.locades les fitxes. Com hi ha dos jugadors, podem usar una llista de dos elements per representar cada jugador. D'aquesta manera, la llista l=[[],[]] representa un tauler buit i la llista [[(1,2)],[]] representa un tauler en el que el jugador 0 té una fitxa a la casella (1,2) mentre que el jugador 1 no ha posat encara cap fitxa. Noteu que si l representa un tauler, mai pot passar que una mateixa casella formi part de l[0] i de l[1] simultàniament.

El segon escull important és la determinació de quan un jugador té les seves fitxes "en ratlla". Fixeu-vos, però, que això es redueix a determinar quan les tres caselles que ocupa jugador, enteses com a coordenades en el pla, estan sobre la mateixa recta. Això és un problema de geometria a $\mathbb{R}^2$ que de ben segur sabeu resoldre.

Seguint aquesta estratègia, la solució seguiria aquest esquema:

```
class TresER(object):
    def __init__(self):
        self._tauler = [[], []]

    def disponibles(self, jugador):
        return 3 - len(self._tauler[jugador])

    def tres_en_ratlla(self, jugador):
        if self.disponibles(jugador):
            return False
        else:
```

```
            l = self._tauler[jugador]
            f1 = (l[2][0] - l[1][1]) * (l[2][1] - l[0][1])
            f2 = (l[2][0] - l[0][0]) * (l[2][1] - l[1][1])
            return f1 == f2

    def treu_fitxa(self, casella):
        if casella in self._tauler[0]:
            self._tauler[0].remove(casella)
        elif casella in self._tauler[1]:
            self._tauler[1].remove(casella)
        else:
            raise Exception("Casella buida")

    def posa_fitxa(self, jugador, casella):
        if self.disponibles(jugador) == 0:
            raise Exception("No diposa de fitxes")
        elif casella in self._tauler[0] or casella in self._tauler[1]:
            raise Exception("Casella ocupada")
        else:
            self._tauler[jugador].append(casella)
```

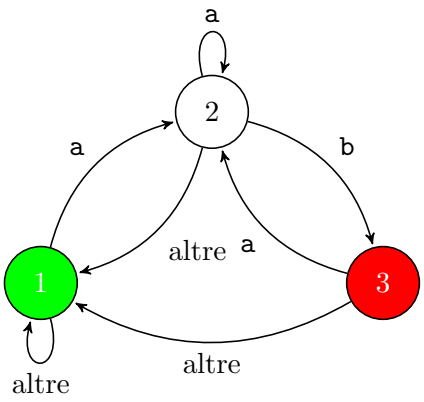
EXERCICI 3.25 Un autòmat finit és un giny que pot usar-se per reconèixer cadenes de caràcters i que està definit per:

1. Un conjunt d'estats $E = \{e_1, \ldots, e_k\}$.
2. Dos estats especials: l'estat inicial $s \in E$ i l'estat final $f \in E$.
3. Una funció de transició δ que indica, donat un estat i un caràcter, quin és el següent estat:

$$\begin{array}{rcl} \delta : E \times C & \longrightarrow & E \\ (e, c) & \mapsto & \delta(e_i, c) = e_j \end{array}$$

Observeu el seu funcionament a través del següent exemple. Imagineu-vos l'autòmat definit pel conjunt d'estats $E = \{1, 2, 3\}$ on l'estat inicial és $s = 1$, l'estat final és $f = 3$ i la funció δ està definida per la següent taula, que també dibuixem en forma de diagrama:

	Entrada	
Estat	Caràcter	Sortida
1	`a`	2
	qualsevol altre	1
2	`a`	2
	`b`	3
	qualsevol altre	1
3	`a`	2
	qualsevol altre	1



Aquest autòmata permet distingir certes cadenes de caràcters. Considerem la cadena `'abcd'`. Per saber si el nostre autòmat la reconeix, simplement comencem en l'estat inicial, 1, i prenem el primer caràcter de la cadena, la `'a'`. Aplicant la funció de transició sabem que el següent estat serà el 2. Considerem el següent caràcter, la `'b'` i apliquem de nou la funció de transició. Ara ens portarà a l'estat 3. Repetiu de nou el procés amb les lletres `'c'` i `'d'`. L'autòmat reconeix la cadena `'abcd'` si després d'aquest procés acaba en l'estat final. Com veieu, la cadena `'abcd'` no és reconeix. En canvi, la cadena `'cdaab'` sí que es reconeix.

En aquest exercici *es demana* que dissenyeu i implementeu la classe Automat. Aquesta classe només té dos mètodes:

- __init__(self, n, t)

  n és un enter que correspon al número d'estats de l'autòmat. Sempre assumim que els estats van de $1, \ldots, n$, i que 1 és l'estat inicial i n l'estat final.

  t és una llista que representa la funció de transició. Aquesta llista està formada per tuples (e1, c, e2) on e1 és un estat, c un caràcter i e2 un altre estat. Un tuple (e1,c,e2) indica que, si estem en l'estat e1 i arriba el caràcter c, llavors hem d'anar a parar a l'estat e2.

  Al caràcter `'.'` li donarem el significat de "qualsevol altre".

- reconeix(self, s)

  Retorna **True** ssi l'autòmat reconeix s.

Seguint amb l'exemple anterior, aquesta classe hauria de passar el següent doctest:

```
>>> a = Automat( 3, [(1, 'a', 2), (1, '.', 1),
                     (2, 'a', 2), (2, 'b', 3), (2, '.', 1),
                     (3, '.', 1), (3, 'a', 2) ])
>>> a.reconeix('abcd')
False
>>> a.reconeix('cdaab')
True
```

Solució exercici 3.25 La solució d'aquest exercici és molt senzilla. L'única consideració que val la pena fer prèviament és fer notar que la funció δ, que defineix el comportament de l'autòmat pot ser representada per un diccionari en el que les claus són parells (estat,caracter) i el valor és el següent estat de l'autòmat. Seguint aquesta idea, la solució és molt senzilla:

```
class Automat(object):

    def __init__(self, n, t):
        self._n = n
        self._d = {}
        for current, char, next in t:
            self._d[(current,char)] = next

    def _next(self,e,c):
        if (e,c) in self._d:
            return self._d[(e,c)]
        else:
            return self._d[(e,'.')]
```

```
def reconeix(self, s):
    state = 1
    for char in s:
        state = self._next(state,c)
    return state == self._n
```

Noteu el mètode privat _next, que calcula el següent estat i té en compte correctament quan cal aplicar les entrades de la funció δ que hem anomenat com "altre caracter".

4 Mètodes especials

EXERCICI 4.1 Definiu una classe Vector3d que representa un vector a $\mathbb{R}^3$. Doteu la classe de les operacions següents i associeu-les als símbols d'operació que s'indiquen usant els mètodes especials:

1. Suma de vectors. Associat a $+$.
2. Resta de vectors. Associat a $-$ binari
3. Producte escalar. Associat a $*$.
4. Producte vectorial. Associat a $**$
5. Mòdul. Sense símbol associat. És un mètode ordinari.

Feu els doctests corresponents i comproveu que la vostra implemenatció funciona correctament.

EXERCICI 4.2 A la classe Vector3d de l'exercici 4.1 afegiu-li una nova operació que implementi el producte per un escalar. Associeu-la al símbol d'operació $*$. Com podeu saber si $*$ fa referència al producte escalar o al producte per un escalar?

Comproveu que aquesta distició es fa correctament mitjançant els doctests pertinents.

EXERCICI 4.3 RespositoryITIC és una plataforma per emmagatzemar les tasques entregades pels alumnes d'una assignatura. A més a més, la aplicació haurà de permetre a l'usuari ordenar les entregues realitzades, així com recuperar determinades entregues usant com localitzador el NIF de l'alumne que ha realitzat l'entrega. I finalment, avaluar les tasques tenint en compte que hi ha diversos professors que imparteixen l'assignatura i que interessa saber-ne qui és el corrector de la tasca en cada moment. A tal, efecte se us passa l'esquelet de classes amb els mètodes involucrats, i un joc de proves del funcionament esperat. Se us demanana implementar **els mètodes que calguin** per tal de que l'execució proporcionada sigui correcta.

```
class Repository(object):
    def __init__(self,maxim):
        self.tasques=[]
        self.maxim=maxim
    def __iter__(self):
        return iter(self.tasques)
    def __str__(self):
        s=""
        for i in self:
            s+=str(i)+"\n"
        return s
    def search(self,tasca):
        #TO_DO_EX1
    def ordena(self):
        #TO_DO_EX2
    def insert(self,tasca):
        #TO_DO_EX3
    def show(self):
        #TO_DO_EX4
    def __len__(self):
        return len(self.tasques)
    def __getitem__(self,p):
        #TO_DO_EX5
class Task(object):
    def __init__(self,nif,nomFitxer):
        self.nif=nif
        self.nomFitxer=nomFitxer
        self.nota=-1
```

```
    def qualifica(self,nota):
        self.nota=nota
    def getNif(self):
        return self.nif
    def getNomFitxer(self):
        return self.nomFitxer
    def __eq__(self,other):
        #TO_DO_EX6
    def getNota(self):
        return self.nota
    def setTeacher(self,teacher):
        self.teacher=teacher
    def getTeacher(self):
        return self.teacher
    def __str__(self):
        #TO_DO_EX7
class Teacher(object):
    def __init__(self,nif):
        self.nif=nif
        self.tasques_corregides=[]
    def qualifica(self,tasca,nota):
        #TO_DO_EX8
    def getNif(self):
        return self.nif
    def __iter__(self):
        return iter(self.tasques_corregides)
    def addTasca(self,t):
        #TO_DO_EX9
    def __str__(self):
        #TO_DO_EX10
```

```
if __name__=='__main__':
    r=Repository(45);  aux=Repository(45)
    r.insert(Task("1111","pract1_tecpro.rar"))
    r.insert(Task("3333","pract1_tecpro.rar"))
    r.insert(Task("5555","pract2_tecpro.rar"))
    r.insert(Task("2222","pract1_tecpro.rar"))
    r.insert(Task("3333","pract3_tecpro.rar"))
    r.insert(Task("3333","pract1b_tecpro.rar"))
    print "Show tasks"
    print r.show()
    aux.tasques=r.ordena()
    print "Show ordered tasks"
    print aux.show()
    nif="3333"
    print "Searching for nif",nif
    m=r.search(nif)
    if len(m)==0:
        print "Not found",nif
    else:
        for t in m:
            print t
    j=Teacher("99")
    for i in range(len(r)/2):
        j.qualifica(r[i],random.randint(0,10))
        j.addTasca(r[i])
    print "Consulta repositori"
    print r
    print "Consulta correccions"
    print j
#Resultats execució
Show tasks
1111 pract1_tecpro.rar Not qualified
3333 pract1_tecpro.rar Not qualified
5555 pract2_tecpro.rar Not qualified
2222 pract1_tecpro.rar Not qualified
3333 pract3_tecpro.rar Not qualified
3333 pract1b_tecpro.rar Not qualified
Show ordered tasks
1111 pract1_tecpro.rar Not qualified
2222 pract1_tecpro.rar Not qualified
3333 pract1_tecpro.rar Not qualified
3333 pract3_tecpro.rar Not qualified
3333 pract1b_tecpro.rar Not qualified
5555 pract2_tecpro.rar Not qualified
Searching for nif 3333
3333 pract1_tecpro.rar Not qualified
3333 pract3_tecpro.rar Not qualified
3333 pract1b_tecpro.rar Not qualified
Consulta repositori
1111 pract1_tecpro.rar 10 Corrected by 99
3333 pract1_tecpro.rar 2 Corrected by 99
5555 pract2_tecpro.rar 6 Corrected by 99
2222 pract1_tecpro.rar Not qualified
3333 pract3_tecpro.rar Not qualified
3333 pract1b_tecpro.rar Not qualified
Consulta correccions
1111 pract1_tecpro.rar 10 Corrected by 99
3333 pract1_tecpro.rar 2 Corrected by 99
5555 pract2_tecpro.rar 6 Corrected by 99
```

5 Herència

EXERCICI 5.1 Defineix tècnicament i posa un exemple dels següents conceptes en Programació Orientada a Objectes.

1. Classe
2. Sobrecàrrega de mètodes
3. Visibilitat privada d'atributs
4. Herència per especialització

EXERCICI 5.2 Dissenyeu i implementeu la classe d'objectes Equacio2g que representa una equació de segon grau en els reals. Per representar aquesta equació, useu els coeficients a, b i c de $ax^2 + bx + c = 0$. Definiu un mètode constructor i també un mètode per avaluar la funció en un punt tal que, donat un valor d'x us torna el valor $ax^2 + bx + c$. Finalment difiniu la igualtat d'equacions. Tingueu en compte que equacions amb coeficients diferents poden ser exactament la mateixa equació. Així, $x^2 + 2x + 3 = 0$ és exactament la mateixa equació que $\frac{1}{2}x^2 + x + \frac{3}{2} = 0$. Coneixeu alguna forma de representar els coeficients que simplifiqui l'operació d'igualtat?

EXERCICI 5.3 A Python, les llistes són tipus de dades predefinits. Com a tals disposen d'una col.lecció de mètodes que permeten treballar amb elles. El mètode index és un d'aquests mètodes. El proposit d'aquest exercici és entendre la relació entre l'operació index i el concepte d'igualtat.

1. Busqueu a la documentació de Python el mètode index. Estudieu-vos amb cura què fa exactament i proveu-lo amb algun petit exemple usant l'intèrpret.
2. El mètode index necessita saber quan dos elements de la llista són iguals (no idèntics). Dissenyeu un petit exemple en el que quedi clar que index busca en una llista seguint el criteri d'igualtat i no pas el d'identitat.
3. Investiguem una mica més afegint classes al joc. Dissenyeu una classe Edat que té un sol atribut edat de tipus **int**. Mitjançant un exemple en que s'usa una lista d'instàncies d'Edat determineu com s'està comportant el mètode index de la llista. Què busca aquest mètode, instàncies iguals o idèntiques? Per què?

   Ara afegiu a la classe Edat els mètodes especials necessaris per definir correctament l'igualtat. Assumiu l'interpretació evident: dues edats són iguals si el atribut edat és igual. Torneu a experimentar amb la llista d'instàncies d'Edat. Com es comporta ara el mètode index? Per què?

EXERCICI 5.4 Els vectors es fan servir molt sovint per representar direccions. En aquest cas, el mòdul del vector és irrellevant sempre que no sigui zero i l'únic interessant és la direcció. Així, per exemple, els vectors $\vec{u} = (1, 2, 3)$ i $\vec{v} = (2, 4, 6)$ són vectors diferents però que representen la mateixa direcció.

Considereu la classe Vector3d de l'exercici 4.1. Dissenyeu una subclasse de Vector3d anomenada Direccio que representa una direcció a $\mathbb{R}^3$. Afegiu, a banda dels mètodes heretats, els següents mètodes:

1. Un mètode anomenat normalitzat(self) que retorna un vector de la mateixa direcció que self però normalitzat, és a dir amb mòdul 1.

2. Un mètode anomenat ortogonal(self,v) que retorna **True** ssi el vector v és perpendicular a self.

3. Afegiu també els mètodes especials necessaris per determinar si dues direccions són iguals.

EXERCICI 5.5 Aquest exercici versa sobre *plotters* i figures geomètriques.

Un plotter és un dispositiu mecànic que pot dibuixar en un full de paper amb un estri de dibuix com una ploma que es desplaça mecànicament. Generalment un plotter considera el paper com una graella de dimensions $X \times Y$. Amb aquesta graella pot referenciar (gairebé) qualsevol punt del paper usant coordenades cartesianes. Les maniobres de les que és capaç un plotter són poques:

puja Separa la ploma del paper. Si la ploma està separada no dibuixa quan es mou.

baixa Arrepenja la ploma en el paper. Qualsevol moviment de la ploma deixarà una traça dibuixada en el paper.

origen La ploma es mou en línia recta a l'orígen de coordenades, que és la cantonada inferior esquerra del paper.

moua Mou la ploma en línia recta a les coordenades absolutes que es donen com a paràmetres.

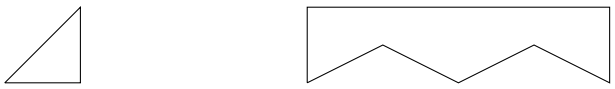
mour Mou la ploma d'acord amb el desplaçament que es dóna com a paràmetre. Si la ploma és a la posició (x, y) i s'executa l'ordre `mour(15,-30)`, la ploma es desplaça en línia recta de (x, y) a la posició $(x + 15, y - 30)$.

1. Dissenyeu i implementeu la classe Plotter les instàncies de la qual representen un plotter. La classe Plotter assumirem que dibuixa sempre en un paper quadrat de dimensions 1000×1000. Cada instància de Plotter té un nom immutable que s'indica en construïr la instància. Els seus mètodes són els que corresponen a les operacions esmentades anteriorment. Com a efecte lateral, tota instància de la classe Plotter escriu per la pantalla una seqüència de missatges que indiquen el que va dibuixant el plotter (no els moviments). Un Plotter acabat de crear té la ploma en l'orígen de coordenades i no toca al paper. Seguidament teniu un exemple d'utilització d'aquesta classe en format `doctest`:

```
>>> p = Plotter('P1')
>>> p.moua(10,10)
>>> p.baixa()
>>> p.mour(5,0)
P1: línia de (10,10) a (15,10)
>>> p.moua(15,5)
```

```
P1: línia de (15,10) a (15,5)
>>> p.puja()
>>> p.origen()
>>> p.baixa()
>>> p.puja()
P1: punt a (0,0)
```

2. Una poligonal tancada és una figura geomètrica tancada formada per una seqüència de segments encadenats que coincideixen en els extrems. En aquest exercici suposarem que els segments no s'auto-intersecten. Les següents figures, per exemple, són poligonals tancades:



Dissenyeu la classe Poligonal que representa una poligonal tancada. La classe ha de permetre:

- Definir una poligonal a base d'afegir vèrtexs a una poligonal buida.
- Dibuixar la poligonal usant un objecte Plotter.
- Calcular la longitud de la poligonal.
- Interrogar sobre el nombre de segments de la poligonal.

Un exemple típic d'ús d'aquesta classe podria ser el següent:

```
>>> p = Poligonal()
>>> p.afegeix((0,0))
>>> p.afegeix((1,0))
>>> len(p)
3
>>> p.afegeix((1,1))
>>> len(p)
4
>>> p.longitud()
3.4142
>>> p.dibuixa(Plotter('P'))
P: línia de (0,0) a (1,0)
P: línia de (1,0) a (1,1)
P: línia de (1,1) a (0,0)
```

3. Un polígon regular és una poligonal formada per N segments de la mateixa longitud de forma que entre dos segments adjacents sempre hi ha el mateix angle. Un triangle equilater o un quadrat són polígons regulars.

Dissenyeu una subclasse de Poligonal anomenada PolReg les instàncies de la qual representen polígons regulars. A banda de les operacions heretades, la classe ha de tenir un constructor que permeti definir el polígon donat el seu centre, un vèrtex del polígon i el número de costats. PISTA: per implementar la classe haureu d'usar una mica de trigonometria i càlcul vectorial.

El següent doctest podria exemplificar l'ús d'aquesta classe:

```
>>> p = PolReg((0,0), (2,0), 4)
>>> len(p)
4
>>> p.longitud()
11.313708499
>>> plot = Plotter('PL1')
>>> p.dibuixa(plot)
PL1: línia de (2,0) a (0,2)
PL1: línia de (0,2) a (−2,0)
PL1: línia de (−2,0) a (0,−2)
PL1: línia de (0,−2) a (2,0)
```

EXERCICI 5.6 Deduiu sense suport del computador què escriurien els següents programes. Posteriorment podeu comprovar la vostra resposta usant el computador.

1. ```
   class C1(object):
     def __init__(self,v):
       self.n = ''
       self.v = v
     def panic(self):
       self.n = 'Eps! ' + str(self.v)

   if __name__ == '__main__':
     c = C1(5)
     c.panic()
     print c.n
   ```

2. ```
   class P(object):
     def __init_(self):
       self.a = 2
       self.v = 4
     def opera(self,v):
       return self.fes_operacio(v)

   class Q(P):
     def fes_operacio(self, s):
       return self.v * s.a

   if __name__ == '__main__':
     o = Q()
     p = Q()
     print o.opera(p)
   ```

3. ```
   class R(object):
     def __init__(self):
       self.x = 10
   ```
   ```

```
class S(R):
  def __init__(self,a):
    super(S, self).__init__()
    self.x *= 2
    self.a = a;
  def get(self):
    return self.x * self.a

if __name__ == '__main__':
  s1 = S(3)
  print s1.get()
```

EXERCICI 5.7 L'intèrpret de classes. Escriviu el resultat dels següents scripts.

Apartat a)

```
class classe1(object):
    def funcio(self,b):
        return 2**b

class classe2(object):
    def __init__(self,valor):
        self.valor=valor
    def funcio2(self,d):
        return self.valor.funcio(d)

if __name__=='__main__':
    a=classe1()
    b=classe2(a)
    print b.funcio2(4)
    print b.funcio2(1/2)
```

Apartat b)

```
class prova(object):
    def funcio(self,b):
        return self.nova(b)+b

class prova2(prova):
    def nova(self,funcio):
        return funcio+2

if __name__=='__main__':
    a=prova2()
    print a.nova(43)
    print a.nova(-3)
```

EXERCICI 5.8 Donada la definició de classes que segueix, responeu les preguntes que trobareu a continuació.

```
class Face1(object):
    def __init__(self):
        print "Whatelse"
    def __str__(self):
        return " Face1 no data"
class Face2(Face1):
    def __repr__(self):
        return " Face2 no data"
class Face3(Face2):
    pass

class Book(object):
    def __init__(self):
        self.c1=Face1()
        self.c2=Face2()
        self.c3=Face3()
        print "Book Class"
    def __str__(self):
        return str(self.c1)+str(self.c2)+str(self.c3)
    def __repr__(self):
        return str(self.c1)+str(self.c2)+str(self.c3)
class Review(Book):
    def __init__(self):
        super(Review,self).__init__()
        print "Review Class"
if __name__=='__main__':
    f=Face3()
    print "From face3",f
    r=Book()
    print r
    j=Review()
    print j
```

Apartat a) Justifiqueu què s'escriu per pantalla després d'executar el main.

Apartat b) En la sentència *f=Face3()* a quin mètode es crida? Justifica si aquest mètode és un exemple de sobrecàrrega/redefinició/herència/delegació de mètodes.

Apartat c) En la sentència *j=Review()* a quin mètode es crida? Justifica si aquest mètode és un exemple de sobrecàrrega/redefinició/herència/delegació de mètodes.

Apartat d) Ara afegim la següent definició de classe contenidora elsBook, i se us demana justificar què s'escriurà per pantalla en executar el main.

```
class elsBook(object):
    def __init__(self):
        self.dades={}

    def add(self,dada):
        if dada not in self.dades:
```

```
            self.dades[dada]=repr(dada)
        else:
            raise Exception("Duplicada")
    def __str__(self):
        print "Els book info",len(self.dades)
        for element in self.dades:
            print element,
        return ""

if __name__=='__main__':
    k=elsBook()
    r=Book()
    j=Review()
    try:
        k.add(r)
        k.add(r)
    except Exception as e:
        print e
    print k
```

Apartat e) Dibuixeu el diagrama UML resultant que contingui les classes anteriors.

EXERCICI 5.9 **Apartat a)** Implementa la classe Esquiador. Tot esquiador ha de tenir un nom (públic), un identificador (privat), i un llistat de puntuacions obtingudes. Cal que en gestioneu el constructor i els mètodes necessaris per tal de tenir una definició de classe completa.

Apartat b) Prenent com a base la classe desenvolupada en l'Apartat a), escriu un exemple de 1) mètode sobrecarregat, 2) mètode redefinit, 3) herència de mètodes i 4) delegació de mètodes.

6 Excepcions

EXERCICI 6.1 Dissenyeu i implementeu una classe que representi una fracció en que el numerador i el denominador són enters. Usant mètodes especials la doteu d'operacions per sumar, restar, multiplicar i dividir fraccions. Afegiu també els mètodes necessaris per tal que, en escriure un valor de la classe fracció, es faci de forma natural.

En un segon pas, contempleu el cas en el que el denominador es fa zero. Si com a resultat de qualsevol operació, el denomninador s'anula, cal aixecar una excepció. Busqueu entre les excepcions pre-definides de Python quina és la més escaient per llençar en aquesta circumstància.

Comproveu el funcionament d'aquesta classe amb un doctest.

EXERCICI 6.2 Deduiu quin és el resultat que escriu el següent programa sense l'ajuda del computador. Després comproveu el resultat usant el computador.

Apartat a)

```
class BeastException(Exception):
  pass

class DoNothing(object):
  def __init__(self):
    self.x1 = 10
  def m1(self,y):
    if y > self.x1:
      raise BeastException('Diabolic error')
    else:
      return self.x1 + y
  def m2(self,z):
    r = self.m1(z)
    print 'Beast number'
    return r

if __name__ == '__main__':
  o = DoNothing()
  try:
    r = o.m2(12)
  except Exception as e:
    print e
  print 'Last day arrived'
```

Apartat b)

```
class TheException(Exception):
    pass
def mesPetit(x,y):
```

```
    try:
        if x<y:
            raise TheException('x<y')
        else:
            return 15+x
    except Exception:
        return 0

if __name__=='__main__':
    try:
        print mesPetit(0,mesPetit(-1,4))
    except Exception:
        print "yes"
```

Apartat c)

```
class TheException(Exception):
    pass
def TheClass(x,y):
    try:
        if x==y:
            raise TheException('x==y')
        else:
            return x+y/2
    except Exception as e:
        return e

if __name__=='__main__':
    try:
        print TheClass(1,TheClass(-1,4))
    except Exception:
        print "yes"
```

EXERCICI 6.3 Escriu els resultats que es mostraran per pantalla després de l'execució de les instruccions del main.

```
class A(object):
    def first(self,x):
        try:
            print "First in"
            self.second(x)
        except ZeroDivisionError as e:
            print "Math"
        except Exception as e:
            print "Firs Except"
        print "First out"

    def second(self,x):
```

```
            print "Second in"
            if x<0: raise Exception()
            y=5/x
            print "second out"
    if __name__=='__main__':
        a=A()
        a.first(0)
        a.first(ord("A"))
```

EXERCICI 6.4 Escriu els resultats que es mostraran per pantalla, després de l'execució del codi que segueix.

```
def first(n):
    x = 0
    try:
        x = second(n)
    except StandardError:
        x = x+1
    return x

if __name__=='__main__':
    print first(0)
    print first(1)
    print first(2)
```

```
def second(n):
    y = 2
    try:
        y = third(n)
    except ValueError:
        y = y+5
    return y
```

```
def third(n):
    if n == 0:
        raise ValueError()
    elif n == 1:
        raise TypeError()
    return n+10
```

EXERCICI 6.5 Donada la següent definició de classes, [**Apartat a**] justifiqueu què s'escriu per pantalla després d'executar el main.

```
class Unknown(object):
    x = 0
    y = 10
    def __init__(self, y=0):
        self.x = y
    def a1(self, y):
        self.x = min(self.x, y)
        return self.x
    def a2(self,y=0):
        self.x = min(self.x, self.y)
        return self.x+y
    def __str__(self):
        return type(self).__name__+" "+str(self.x)
    def __eq__(self,other):
        return self.x==other.x

class Important(Unknown):
    def suma(self,y=0):
        self.y = self.x + self.x
        return self.y+y
```

```
if __name__=='__main__':
    f=Unknown(5)
    print f #ex 1.1
    print f.a1(7) #ex 1.2
    g=Important(3)
    print f.a1(3) #ex 1.3
    print g.a1(5) #ex 1.4
    print g.suma(1) #ex 1.5
    print f.a2() #ex 1.6
    print g.a2() #ex 1.7
    g=Important(0)
    print f.suma() #ex 1.8
```

[**Apartat b**] En la sentència *g=Important(3)* a quin mètode es crida? Justifica si aquest mètode és un exemple de sobrecàrrega/redefinició/herència/delegació de mètodes. [**Apartat c**] En la sentència *g.a2()* a quin mètode es crida? Justifica si aquest mètode és un exemple de sobrecàrrega/redefinició/herència/delegació de mètodes. [**Apartat d**] Ara afegim la següent definició de classe contenidora elsUnknown, i se us demana justificar què s'escriurà per pantalla en executar el main.

```
class elsUnknown(object):
    def __init__(self):
        self.dades={}

    def add(self,dada):
        if dada not in self.dades:
            self.dades[dada]=repr(dada)
        else:
            raise Exception("Duplicada")
    def __str__(self):
        return "Els unknown info "+str(len(self.dades))+"\n"+",".join([str(element) for element in self.dades])

if __name__=='__main__':
    k=elsUnknown()
    r=Unknown()
    j=Important(2)
    try:
        k.add(r)
        k.add(j)
        k.add(r)
    except Exception as e:
        print e
    print k
```

Exercici 6.6 Donat el següent fragment de codi, i per cadascuna de les afirmacions que segueixen, escriviu si l'afirmació és [Certa/Falsa]. **(La no justificació invalida la resposta)**

```
class F(object):
    x = 0
    y = 10
    def __init__(self, y):
        self.x = y
        self.__z=99
    def a1(self, y):
        self.x = max(self.x, y)
        return self.x

    def a2(self):
        self.x = max(self.x, self.y)
        return self.x

    def b(self,a):
        return self.x+a

class G(F):
    def b(self):
        self.y = self.x * self.x
        return self.y

    def a2(self):
        self.x=0
```

```
class H(F):
    def b(self):
        self.y=22
        return self.y

class I(H):
    pass

class J(G,H):
    pass

if __name__=='__main__':
    h1=H(2)
    g1=G(2)
    v=[]
    v.append(h1)
    v.append(g1)
    for a in v:
        print a.b()
```

1. El mètode b de la classe G és un exemple de redefinició de mètodes.
2. El mètode $a2$ de la classe G és un exemple de sobrecàrrega de mètodes.
3. Un objecte de la classe G pot accedir a l'atribut x de la classe F.

4. El resultat de la execució del programa serà 4 4.
5. No es poden crear objectes de la classe I perquè no hi ha definit el mètode constructor.
6. Un objecte de la classe H pot accedir a l'atribut z de la classe F.
7. La crida $i.b(11)$ sobre un objecte de classe I generaria un error d'execució.
8. Les crides

   ```
   j=J(1); print j.b()
   ```

   provocaran un error d'execució.

EXERCICI 6.7 Se us demana la gestió simplificada de les multes per retorn de préstecs fora de termini d'una biblioteca. En concret, se us passa l'especificació de requeriments que ha de complir l'aplicatiu. Noteu que per simplificar, s'usen tipus bàsics en la representació de llibres, persones i dates.

- Un llibre es representa per un string corresponent al seu títol.
- Una persona que usa la biblioteca es representa per un string corresponent al seu nick.
- Una data es representa per un enter, que correspon al número de dies des que es va obrir la biblioteca.

Llegiu atentament les especificacions dels mètodes i els exemples de funcionament proporcionats a continuació per tal d'aconseguir un disseny òptim de l'aplicatiu.

La classe **Biblioteca** ha de contenir un atribut anomenat **diamulta**, inicialitzat amb el valor 0.25, i els següents mètodes, que heu d'implementar en cada apartat.

[**Apartat a**] Mètode **constructor**: Donada una llista de llibres, inicialitza la biblioteca. Se us demana que utilitzeu un **diccionari** per la gestió dels continguts de la biblioteca.

[**Apartat b**] Mètode **lloga**: enregistra el llibre, la persona i la data en la que el llibre es lloga. Cada llibre pot estar en préstec fins a 7 dies. A partir de llavors direm que està endarrerit de préstec. Per exemple, si es lloga el dia x, es troba en préstec fins el dia x+7, i es trobarà endarrerit de préstec el dia x+8. Aquest mètode no retorna res.

[**Apartat c**] Mètode **retorna**: donat un llibre i la data en que el llibre es retorna, modifica el registre del llibre. Retorna un número que representa la multa si el llibre s'ha retornat fóra del termini de préstec i 0.0 si s'ha tornat en el termini previst. La multa correspon al número de dies que s'ha passat del termini de préstec multiplicat per l'atribut de classe diamulta.

[**Apartat d**] Mètode **endarrerimentLlibres**: donada una persona i una data, retorna la llista de llibres tals que la persona els ha llogat i estan endarrerits en el préstec per la data proporcionada.

A continuació segueix un exemple de funcionament. NOTA: Suposeu que les operacions efectuades són legals. Per exemple, els llibres retornats corresponen al llibres prèviament llogats. I els llibres que es lloguen, s'han retornat abans. Suposeu també que només hi ha 1 exemplar de cada llibre. Adoneu-vos que no és necessària cap gestió per als usuaris.

```
if __name__=='__main__':
    lib = Biblioteca(['Tintin', 'Matrix', 'Python', 'Java', 'MySQL', 'Biblia'])
    lib.lloga('Tintin', 'Elena', 1)
    lib.lloga('Python', 'Elena', 1)
    lib.lloga('MySQL', 'Elena', 10)
    print lib.endarrerimentllibres('Elena', 13)
```

```
    #['Python', 'Tintin']
    print lib.retorna('Tintin', 13)
    #1.25
    print lib.retorna('Python', 18)
    #2.5
    print lib.retorna('MySQL', 18)
    #0.25
```

[**Apartat e**] A continuació definiu una nova classe de nom **BibliotecaProrroga** que es comporta com la classe Biblioteca, amb la particularitat que proporciona un període de pròrroga abans de que les multes comencin a comptar. El número de dies de pròrroga s'especifiquen quan s'instancia un objecte de la classe. Per exemple,

```
    lib = BibliotecaProrroga(2, ['Tintin', 'Matrix', 'Python', 'Java', 'MySQL', 'Biblia'])
    lib.lloga('Tintin', 'Elena', 1)
    print lib.retorna('Tintin', 13)
    #0.75
```

Se us demana la implementació completa de la classe **BibliotecaProrroga**. No està permesa la repetició del codi ja definit a la classe Biblioteca. En particular, no cal repetir el càlcul de la multa.

Exercici 6.8 Se us proporcionen les següents definicions (incompletes) de classes. El codi proporcionat preten tracejar les assignatures aprovades per un estudiant, per tal de comprovar si ha satisfet totes les especificacions del grau (GIR) i del departament.

```
class eMITstudent(object):
    GIR = ['mats.111', 'est.011', 'fis.01', 'fis.02', 'electro.01', 'electro.02','rest1', 'rest2', 'instlab',
'prog1', 'prog2', 'prog3', 'prog4','prog5', 'prog6', 'prog7', 'prog8']

    DEPT = []

    def __init__(self, AP = []):
        self.apr = AP

    def aprova(self, assig):
        for a in assig:
            self.apr.append(a)

    def acabaReq(self, req):
        missing = []
        for r in req:
            if not r in self.apr:
                print r +' is missing'
                missing.append(r)
        return len(missing) == 0

    def acaba(self):
        # TO DO

class Course62(eMITstudent):
    DEPT = ['tec.01', 'tec.02', 'electro.03', 'tec.041', 'tec.002', 'tec.003', 'tec.004', 'tec.005', 'tec.011',
'tec.013', 'tec.033', 'tec.115', 'tec.813', 'tec.336','tec.AUT', 'tec.AUP', 'instlab']
```

Adoneu-vos que la definició d'una subclasse de eMITstudent, tal com Course62, defineix un atribut DEPT que especifica els requeriments del departament. Un exemple d'ús es proporciona a continuació.

```
if __name__==’__main__’:
    Albert = Course62([’electro.01’])
    Albert.aprova([’mats.111’, ’electro.02’, ’fis.01’, ’prog1’])
    Albert.aprova([’electro.03’, ’fis.02’, ’tec.01’, ’prog2’])
    Albert.aprova([’tec.02’, ’tec.041’, ’est.011’, ’prog3’])
    Albert.aprova([’tec.002’, ’tec.003’, ’tec.013’, ’prog4’])
    Albert.aprova([’tec.004’, ’tec.005’, ’tec.011’, ’prog5’])
    Albert.aprova([’tec.033’, ’tec.115’, ’tec.336’, ’prog6’])
    Albert.aprova([’tec.813’, ’tec.AUT’, ’tec.AUP’, ’prog7’, ’prog8’])
    print ’acaba? ’, Albert.acaba()
    Albert.aprova([’rest1’, ’rest2’])
    print ’acaba? ’, Albert.acaba()
```

[**Apartat a**] Se us demana la implementació òptima del mètode **acaba**. Aquest mètode ha de comprovar si ha assolit els requeriments del GIR i els requeriments del DEPT, i retorna un booleà (**True** si ha aprovat totes les assignatures requerides pel grau i pel departament).

[**Apartat b**] La implementació prèvia proporcionada no contempla el següent requeriment: Un estudiant que hagi aprovat tec.01 i tec.02, obté automàticament el requeriment del GIR anomenant instlab. Se us demana que implementeu òptimament un nou mètode que gestioni dita funcionalitat correctament. Es penalitzarà la repetició de codi.

Exercici 6.9 L'intèrpret de classes. Escriviu els missatges que es mostren per pantalla en l'execució de l'script següent.

```
class ok(object):
    def posaNota(self,nota=10):
        self.__nota=nota
    def __str__(self):
        return str(self.__nota)

class provando(object):
    def __init__(self):
        self.__a=0
        self.b=22

class hello(provando):
    pass

class dontknow(hello):
    def __str__(self):
        return str(self.b*2)

if __name__==’__main__’:
    o=ok()
    o.posaNota()
    print o
    h=hello()
    print h
    d=dontknow()
    print d
    j=provando()
    j.b=122
    print j.b+h.b
```

7 UML

Exercici 7.1 Dissenyeu un diagrama de classes que modeli el sistema que es descriu a continuació.

Una empresa d'enginyeria es dedica a fer projectes de disseny electrònic. Cada projecte és per un client i, naturalment, un client ho pot ser de diversos projectes simultàniament. Cada projecte és desenvolupat per un equip d'enginyers. Un enginyer pot participar en quatre projectes com a molt.

Exercici 7.2 Dissenyeu un diagrama de classes que modeli el sistema que es descriu a continuació.

En un hospital treballen metgesses, infermers i celadors. Les metgesses i infermers estan organitzats en equips. Cada equip el formen dos metgesses i quatre infermers. L'hospital està distribuït en plantes, i cada a cada planta hi ha un conjunt d'habitacions. Un equip de metgesses és responsable d'un màxim de dotze habitacions, que poden estar en diverses plantes. D'altra banda, els celadors estan assignats a una planta concreta i tota la seva feina se centra en aquesta planta.

Exercici 7.3 Una *Smart City* disposa d'un sistema de gestió d'aparcaments estructurat de la següent forma:

1. Totes les places d'aparcament de la ciutat estan identificades per un identificador enter i agrupades en zones. Hi ha un sensor a cada plaça que detecta si està ocupada o no. Per a cada zona se sap el nombre màxim de places i la ocupació en cada moment.
2. Els carrers poden incloure zones d'aparcament, per exemple a les vores. Ambtot hi ha zones que són específiques com és el cas dels aparcaments dissuassoris.
3. Els carrers estan dividits per barris i una zona mai es compartida entre barris.
4. Cada plaça es localitza amb una referència topogràfica que indica on se situa geogràficament.

Dissenyeu un diagrama de classes que modeli correctament aquest sistema de gestió d'aparcaments.

Exercici 7.4 La relació entre una assignatura i els estudiants matriculats respon al següent diagrama UML:

Assumint que un estudiant té com atributs el nom i el dni i una assignatura el nom i l'aula on es fa classe, implementeu les corresponents classes de Python. Noteu que heu d'implementar també la associació entre Assignatura i Estudiant: feu-ho usant un atribut privat d'Assignatura que emmagatzema en forma de llista els estudiants matriculats. Afegiu a Assignatura els mètodes necessaris per gestionar l'associació.

EXERCICI 7.5 A l'exercici anterior seria molt escaient dotar la classe Assignatura de les eines necessàries per poder recórrer còmodament els estudiants matriculats sense trencar la privacitat de la implementació. Per exemple, seria molt interessant poder escriure codi com el següent:

```
a = Assignatura('mates')
a.add(Estudiant('Pere', 345667))
a.add(Estudiant('Marta',34456167))
a.add(Estudiant('Laura', 34256667))
for estud in a:
    print estud.nom
```

Per implementar això cal que implementeu el mètode especial __iter__() a la classe Assignatura. Estudieu la documentació del mètode i implementeu aquesta millora.

EXERCICI 7.6 La relació entre un equip i els seus jugadors respon al següent diagrama UML:

Com la relació és bidireccional, una instància d'Equip ha de conèixer els seus jugadors i una instància de Jugador ha se saber a quin equip juga. A més, segons el diagrama, no pot existir cap jugador que no tingui equip.

Aquest fet planteja certes dificultats quan es vol implementar aquesta relació:

- D'acord amb el diagrama, un jugador no pot existir sense tenir un equip de referència.
- Quan lliguem un jugador a un equip cal que el jugador tingui constància de l'equip i l'equip del jugador.

Penseu com implementarieu els mètodes corresponents per satisfer aquestes condicions. Després d'haver-hi pensat continueu llegint.

Una solució al problema consisteix a dotar la classe Jugador d'un inicialitzador que, a més d'inicialitzar el nom, inicialitza l'equip per al qual juga el jugador. D'aquesta forma sempre que creem un jugador té obligatòriament un equip de referència:

```
e = Equip('Mollerussa')
j = Jugador('Paula', e)
```

Amb això no és suficient per que podriem fer construccions com aquesta, que és clarament errònia:

```
e1 = Equip('Alcoi')
e2 = Equip('Binissalem')
j = Jugador('Queralt', e1)
e2.add(j)
```

Per evitar això, vincularem la creació d'un jugador a un mètode de la classe Equip de forma que en invocar aquest mètode, add(), es crei el jugador i es vinculi a l'equip:

```
e1 = Equip('Torrent')
j = e1.add('Queralt')
```

Implementeu les dues classes d'aquest exercici i la seva relació usant aquesta estratègia. Recordeu afegir els mètodes necessaris per saber l'equip d'un jugador i també els jugadors d'un equip.

Exercici 7.7 Una empresa que es dedica a la venda i reparació de sistemes de seguretat us ha encarregat una aplicació que gestioni els encàrrecs que rep dels seus clients. L'aplicació ha d'emmagatzemar informació dels sistemes de seguretat que té l'empresa, dels seus clients i dels serveis que aquests requereixen.

Un sistema de seguretat té un codi alfanumèric que l'identifica i està format per un conjunt de sensors dels quals n'hi ha de tres tipus diferents: sensors volumètrics, que s'utilitzen per detectar moviment, sensors d'infraroig, que detecten calor i sensors tàctils, que es disparen si dues superfícies deixen d'estar en contacte (per exemple quan s'obre una porta).

Tots aquests sensors tenen un preu i un codi alfanumèric que els identifica. Els sensors volumètrics es poden configurar amb un valor real que indica la longitud en metres del seu radi d'acció. Els d'infraroigs es poden configurar amb un valor enter de manera que l'alarma es dispara quan la temperatura del lloc on es troba el sensor supera aquest valor. Els sensors tàctils, tenen un LED ocult al client que es posa de color verd si l'alarma va saltar per un canvi de temperatura, o bé vermell per la detecció de moviment. D'aquesta manera és fàcil saber la causa de l'alarma.

Quan un client nou es posa en contacte amb l'empresa, cal emmagatzemar el seu nom, els cognoms, el DNI, l'adreça, la població i el seu telèfon. Els clients poden demanar dos tipus de servei a l'empresa: la compra d'un sistema de seguretat, o bé la reparació d'un que ja tenien instal·lat. Interessa guardar tots els serveis que cada client sol·licita al llarg del temps, la data en què es va sol·licitar cadascun d'ells, el codi numèric que identifica cada servei i el sistema de seguretat que es va vendre o reparar. Cal notar que un sistema de seguretat només es pot vendre un cop i, en canvi, ser reparat moltes vegades al llarg del temps.

Dibuixeu el diagrama UML que doni resposta a les especificacions anteriors. Per cadascuna de les classes creades, afegiu una breu indicació de quins atributs contindria cada classe, per permetre les relacions que heu detectat.

Exercici 7.8 El diagrama UML. Una petita fàbrica de mecanitzats ens ha encarregat un programari per tal de portar a terme la gestió de la seva producció.

La fàbrica consta d'un cert nombre de premses de mecanitzat. D'aquestes premses ens interessa guardar el seu codi de referència, la força (en Tm) i la dimensió de la taula (en mm). A la fàbrica existeixen dos tipus diferents de premses: mecàniques i hidràuliques. Les premses mecàniques tenen un paràmetre que indica el nombre de cops/min que pot fer la premsa. Les hidràuliques tenen dos paràmetres que indiquen, respectivament, la velocitat de treball mínima i màxima de la premsa (en mm/seg.). Aquests paràmetres s'han de tenir en consideració en el càlcul del rendiment (nombre de peces/hora) que la premsa pot donar per a cada tipus de producte, tal com s'explica més endavant.

La fàbrica ja disposa d'una base de dades de les seves premses, així com dels seus clients i dels productes (tipus de peces) que pot fabricar. Dels clients es guarda el nom de l'empresa, el NIF i l'adreça. A la fàbrica es fan dos tipus de productes: plàstics i metàl·lics, que es diferencien pel tipus d'eina que cal fer servir a la premsa. Dels productes es guarda un codi de referència, el preu de venda (/unitat) i un paràmetre de complexitat de fabricació (valors de l'1 al 5) que intervé en el càlcul del rendiment de la premsa en la que es fabriqui aquest producte. La gestió que se'ns ha encarregat consisteix en:

- Entrar comandes: Les comandes venen definides per:
  - Producte/s a fabricar.
  - Quantitat de cada tipus de producte.

  - Client que fa la comanda.
  - Data d'entrega.
  - Estat: rebuda (=0), en procés (=1), o finalitzada (=2).

- Planificar treball: A partir de les comandes rebudes, generar les ordres de treball corresponents i assignar com estat de la comanda "en procés". Una comanda podrà donar lloc a una o més ordres de treball. En una ordre de treball s'especificaran les dates d'inici i de final, així com la premsa en la que es realitzarà el treball. Per estimar la data de final s'ha de calcular el rendiment de la premsa en la que es realitzarà el treball, que depèn del tipus de premsa i del paràmetre de complexitat del producte a fabricar.

- Preparar entregues: Consistiria en generar un llistat de totes aquelles comandes que, a la data en que s'executi aquest mètode, hagin finalitzat totes les seves ordres de treball associades. L'estat d'aquestes comandes s'assignaria com "finalitzada".

NOTA: Òbviament, s'entén que la gestió descrita en aquest enunciat correspon a una versió molt simplificada de la realitat d'una fàbrica, però volem remarcar que en l'exercici heu de considerar, exclusivament, la funcionalitat descrita a l'enunciat. Així, es demana:

Dibuixeu **el diagrama UML** que doni resposta a les especificacions anteriors. Per cadascuna de les classes creades, afegiu una breu indicació de quins atributs contindria cada classe, per permetre les relacions que heu detectat.

Exercici 7.9 Classes, objectes, relacions i excepcions.

[Apartat a] Creeu tres classes, *compA*, *compB*, *compC*, de manera que formin una jerarquia de classes on la classe *compA* és la superclasse i les classes *compB* i *compC* hereten de la classe *compA*. Implementeu un mètode a la classe *compA* anomenat *treballa* que retorni una cadena indicant que és un mètode de la classe *compA*. Posteriorment, creeu el mètode *treballa* a les classes *compB* i *compC*. A continuació segueix un exemple de funcionament.

```
if __name__=='__main__':
    c1=compA()
    c2=compB()
    c3=compC()
    print "The classes are"
    print c1
    print c2
    print c3
    print "The working classes are"
    c1.treballa()
    c2.treballa()
    c3.treballa()
#Resultats de l'execució:
The classes are
compA with no attributes
compB with no attributes
compC with no attributes
The working classes are
Working compA
```

```
Working compB
Working compC
```

[Apartat b] **Justifiqueu** si el mètode constructor de la classe *compB* és un exemple *d'herència de mètodes*, de *sobrecàrrega de mètodes*, de *delegació de mètodes* o bé de *redefinició de mètodes*.

[Apartat c] **Justifiqueu** si el mètode *treballa* de la classe *compC* és un exemple *d'herència de mètodes*, de *sobrecàrrega de mètodes*, de *delegació de mètodes* o bé de *redefinició de mètodes*.

[Apartat d] **Justifiqueu** com heu aconseguit que el print d'un objecte d'instància de la classe *compA* mostri el missatge *compA with no attributes*. Justifiqueu si aquest mètode és un exemple *d'herència de mètodes*, de *sobrecàrrega de mètodes*, de *delegació de mètodes* o bé de *redefinició de mètodes*.

[Apartat e] Es vol modelar una classe nova, *components*, que serà un contenidor d'objectes d'instància de *compA*, *compB* i *compC*. **Justifiqueu** quin tipus de relació (herència/ associació /agregació/ composició) hi ha entre la classe *Components* i *compA*, *compB* i *compC*. **Dibuixeu** el diagrama UML resultant.

[Apartat f] Creeu la classe *components* i afegiu els mètodes necessaris per obtenir els següents resultats.

```
if __name__=='__main__':
    c1=compA()
    c2=compB()
    comp=components()
    try:
        comp.add(c1)
        comp.add(c2)
        comp.add(c1)
    except Exception as e:
        print e
    print "-----Components afegides--------"
    for c in comp:
        print c
#Resultats de l'execució
Component duplicada
-----Components afegides--------
compA with no attributes
compB with no attributes
```

Exercici 7.10 Classes, objectes, relacions i excepcions. Donada la següent definició de classes,

```
class Comp1(object):
    def __init__(self):
        print "component 1"
    def __str__(self):
        return " No attributes in c1"

class Comp2(object):
    def __str__(self):
        return " No attributes in c2"
```

```
class Comp3(object):
    def __init__(self):
        print "component 3"
    def __str__(self):
        return " No attributes in c3"

class Root(object):
    def __init__(self):
        self.c1=Comp1()
        self.c2=Comp2()
        self.c3=Comp3()
        print "Root Class"
    def __str__(self):
        return str(self.c1)+str(self.c2)+str(self.c3)

class Node(Root):
    def __init__(self):
        super(Node,self).__init__()
        print "Node Class"

if __name__=='__main__':
    n=Node()
    print n
```

[Apartat a] Justifiqueu què s'escriu per pantalla després d'executar el main.

[Apartat b] En la sentència *self.c2=Comp2()* a quin mètode es crida? Justifica si aquest mètode és un exemple de sobrecàrrega/redefinició/herència/delegació de mètodes?

[Apartat c] En la sentència *n=Node()* a quin mètode es crida? Justifica si aquest mètode és un exemple de sobrecàrrega/redefinició/herència/delegació de mètodes?

[Apartat d] Ara afegim la següent definició de classe contenidora elsRoot, i se us demana justificar què s'escriurà per pantalla en executar el main.

```
class elsRoot(object):
    def __init__(self):
        self.dades=[]
    def add(self,dada):
        self.dades+=[dada]
    def __str__(self):
        for element in self.dades:
            print element
        return ""

if __name__=='__main__':
    k=elsRoot()
    r=Root()
    j=Node()
    k.add(r)
    k.add(j)
```

```
print k
```

[Apartat e] Afegiu el necessari per tal de gestionar la captura/llençament d'excepcions correctament en invocar el mètode add.

[Apartat f] Dibuixeu el diagrama UML resultant que contingui les classes anteriors.

EXERCICI 7.11 Classes, objectes, relacions i excepcions. Donada la següent definició de classes, [Apartat a] Escriu el resultat de les expressions contingudes en el main i que estan numerades. Si no retorna res, escriu None. Si es tracta d'un error, escriu Error i una breu explicació del perquè.

```
class F(object):
    x = 0
    y = 10
    def __init__(self, y):
        self.x = y
    def a1(self, y):
        self.x = max(self.x, y)
        return self.x
    def a2(self):
        self.x = max(self.x, self.y)
        return self.x
    def invoca(self):
        try:
            return self.espera(self.x)
        except Exception as e:
            return "not found"
    def __str__(self):
        return self.__class__.__name__+"--"+str(self.x)

class G(F):
    def b(self):
        self.y = self.x * self.x
        return self.y
    def espera(self,x):
        return x+4

class H(F):
    def espera(self,x):
        return x*2

class I(H):
    pass

if __name__=='__main__':
    #1.1 f=F(5)
    #1.2 g=G()
    #1.3 print f.a1(7)
    #1.4 g=G(3)
    #1.5 print f.a1(3)
    #1.6 print g.a1(5)
    #1.7 print f.b()
    #1.8 print g.b()
    #1.9 print f.a2()
    #1.10 print g.a2()
    #1.11 print g.invoca()
    #1.12 h=H(22)
    #1.13 print h.invoca()
    #1.14 print g
    #1.15 print h
    #1.16 i=I(5)
    #1.17 print i.invoca()
```

[Apartat b] En la sentència *i=I(5)* de la línia 1.16, a quin mètode es crida? Justifica si aquest mètode és un exemple de sobrecàrrega/redefinició/herència/delegació de mètodes.

[Apartat c] En la sentència *print h.invoca()* de la línia 1.17, a quin mètode es crida? Justifica si aquest mètode és un exemple de sobrecàrrega/redefinició/herència/delegació de mètodes.

[Apartat d] Ara afegim la següent definició de classe contenidora elsF, i se us demana justificar què s'escriurà per pantalla en executar el main.

```
class elsF(object):
    def __init__(self):
        self.dades={}

    def add(self,dada):
        if dada not in self.dades:
            self.dades[dada]=repr(dada)
        else:
            raise Exception("already")

    def __iter__(self):
        return iter(self.dades)

    def __str__(self):
        return "Els fs info "+">>>".join([str(element) for element in self])
```

```
if __name__=='__main__':
    k=elsF()
    r=I(1)
    j=F(1)
    try:
        k.add(r)
        k.add(j)
        k.add(j)
        k.add(r)
    except Exception as e:
        print e
    print k
```

[Apartat e] Dibuixeu el diagrama UML resultant que contingui les classes anteriors.

Exercici 7.12 Classes, objectes, relacions i excepcions. Donada la següent definició de classes, [Apartat a] Escriu el resultat de les expressions contingudes en el main i que estan numerades. Si no retorna res, escriu None. Si es tracta d'un error, escriu Error i una breu explicació del perquè.

```
import math
class classe1(object):
    x = -3
    y = 100
    def __init__(self, y):
        self.x = y
    def methodA1(self, y):
        self.x = min(self.x, y)*2
    def ask(self):
        try:
            return self.wait(self.x)
        except Exception as e:
            return "not found"
    def __str__(self):
        return self.__class__.__name__+"--"+str(self.x)
class classe2(classe1):
    def methodB(self):
        self.y = math.pow(self.x,2)
    def wait(self,x):
        return x*-2
class classe3(classe1):
    def methodA1(self):
        self.x = max(self.x, classe1.y)
    def wait(self,x):
        return x*2
class classe4(classe3):
    pass
```

[Apartat a] Escriu els resultats que es mostraran per pantalla després de l'execució del main que segueix.

```
if __name__=='__main__':
    classe1=classe1(5)
    classe1.methodA1(-2)
    print "1.1",classe1
    classe1.methodA1(1)
    print "1.2",classe1
    classe2=classe2(3)
    classe2.methodA1(-5)
    print "1.3",classe2
    classe2.methodB(),
    print "1.4",classe2
    print "1.5",classe2.ask()
    classe3=classe3(22)
    classe3.methodA1()
    print "1.6",classe3
    print "1.7",classe3.ask()
    classe4=classe4(5)
    print "1.8",classe4.ask()
```

[Apartat b] Ara afegim la següent definició de classe contenidora elsF, i se us demana justificar què s'escriurà per pantalla en executar el main.

```
class TheClasses(object):
    def __init__(self):
        self.dades={}
    def add(self,dada):
        if dada not in self.dades:
            self.dades[dada]=repr(dada)
        else:
            raise Exception("already")
    def __iter__(self):
        return iter(self.dades)
    def __str__(self):
        return "Classes info "+">>>".join([str(element) for element in self])
```

```
if __name__=='__main__':
    k=TheClasses()
    try:
        k.add(classe1)
        k.add(classe4)
        k.add(classe1)
    except Exception as e:
        print "1.9",e
    print "1.10",k
```

[Apartat c] Dibuixeu el diagrama UML resultant que contingui les classes anteriors.

Exercici 7.13 Donada la definició de la classe Persona que segueix, i que relaciona quina persona és compatible amb una altra,

```
class Persona(object):
    def __init__(self, n):
        self.name = n
        self.compatible = None
    def creaCompatible(self, p):
        self.compatible=p

    def __str__(self):
       #TO DO

def parellaCompatible(n1, n2):
  # TO DO
```

```
if __name__=='__main__':
    s=Persona("Eva")
    print s #1
    t=Persona("Eric")
    s.creaCompatible(t)
    t.creaCompatible(s)
    print s #2
    print t #3
    l=parellaCompatible("Maria","John")
    for e in l:
        print e #4,5

#Resultats d'execució:
#1  Eva has no compatible person
#2  Eva has compatible Persona: Eric
#3  Eric has compatible Persona: Eva
#4  Maria has compatible Persona: John
#5  John has compatible Persona: Maria
```

Apartat a) Implementa el mètode *str* de la classe Persona, que permetrà el funcionament esperat detallat al joc de proves. **Apartat b)** Implementa la funció *parellaCompatible*, tal que, donats els noms de dues persones, crea les 2 persones amb el nom corresponent, i fa les següents assignacions: la persona compatible de la primera és la segona, i la persona compatible de la segona és la primera. Finalment retorna la llista amb les 2 persones. **Apartat c)** Prenent com a base la classe Persona, escriu un exemple de 1) mètode sobrecarregat, 2) mètode redefinit, 3) herència de mètodes i 4) delegació de mètodes.

Exercici 7.14 Dibuixa i justifica el diagrama UML que permetria modelar el següent problema,

Apartat a) Es preten emmagatzemar les dades referents a diversos sensors de temperatura. Cada sensor es classifica en sensors impermeables o sensors no impermeables. Les dades referents a les temperatures es recullen en un històric anual, d'on se'n guarda el dia, la franja horària (format hh:mm:ss) i el sensor que ha recollit la dada.

Apartat b) Justifica com s'implementaria en Python (breu descripció dels atributs / mètodes que les classes contindran).

EXERCICI 7.15 Donats els següents fragments de codi, escriu els resultats que es mostraran per pantalla.

Apartat a)

```
class Forma(object):
    def __cmp__(s1, s2):
        return cmp(s1.area(), s2.area())

class Quadrat(Forma):
    def __init__(self, h):
        self.side = float(h)

    def area(self):
        return self.side**2

    def __str__(self):
        return 'Quadrat with side ' + str(self.side)

class Cercle(Forma):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14159*(self.radius**2)

    def __str__(self):
        return 'Cercle with radius ' + str(self.radius)

def f(L):

    if len(L) == 0: return None
    x = L[0]
    for s in L:
        if s >= x:
            x = s
    return x

s = Quadrat(4)
print s.area()
L = []
shapes = {0:Cercle, 1: Quadrat}
for i in range(10):
    L.append(shapes[i%2](i))
print L[4]
print f(L)
```

Apartat b)

```
class F(object):
    x = 0
```

```
    y = 10
    def __init__(self, y):
        self.x = y
    def a1(self, y):
        self.x = max(self.x, y)
        return self.x
    def a2(self):
        self.x = max(self.x, self.y)
        return self.x
class G(F):
    def b(self):
        self.y = self.x * self.x
        return self.y

f=F(5)
print f
print f.a1(7)
g=G(3)
print f.a1(3)
print g.a1(5)
print g.b()
print f.a2()
print g.a2()
g=G()
f.b()
```

Exercici 7.16

L'objectiu del problema següent consisteix a la gestió d'elements químics, la taula periòdica d'elements, i finalment els elements que composen una molècula. A tal efecte, se us demana:

Apartat a) Implementar la classe *Element*, tal que permeti gestionar el nom de l'element, el número atòmic, el símbol i la massa atòmica. L'accés als atributs cal fer-lo a través de mètodes accessors. Afegiu un mètode tal que permeti obtenir la informació de l'element, en el format del joc de proves que segueix.

Donat el codi que segueix,

```
first=Element("Sodium",11,"Na",22.98977)
print first
```

Cal que es mostri per pantalla la informació de l'element tal i com es mostra a continuació.

```
Element(Sodium,11,Na,22.98)
```

Apartat b) Implementar la classe *Taula*, que ha de gestionar un contenidor d'elements químics de manera òptima, atenent als mètodes que es detallen a continuació. Suposeu que només es faran cerques pel nom de l'element químic.

Les dades dels elements químics es proporcionen a través d'un fitxer de nom *nameFile*, que és un fitxer *.csv* amb el següent format de contingut:

```
Sodium,11,Na,22.98
Hydrogen,1,H,1.01
```

```
Carbon,6,C,12.01
Nitrogen,7,N,14.01
Oxygen,8,O,16
Phosphorus,15,P,30.98
Sulfur,16,S,32.06
Potassium,19,K,39.1
```

En la següent definició de mètodes, cal que gestioneu totes les excepcions que es puguin produir. Utilitzeu, si us cal, altres mètodes addicionals propis de les definicions de classes.

1. Cal implementar el mètode constructor de la classe *Taula*. Aquest mètode ha de cridar al mètode *readFile* per tal d'inicialitzar els *Elements* que ha de contenir.

```
def __init__(self,nameFile):
        """
        Crea una taula amb els elements continguts en el fitxer .csv de nom nameFile
        """
```

2. Cal implementar el mètode *readFile*, tal que, reb com a paràmetre el *nameFile.csv* i ha de llegir el fitxer línia per línia. Aquest mètode ha de ser invocat pel constructor de la classe *Taula*.

```
def readFile(self,nameFile):
        """
        llegeix el fitxer .csv nameFile i gestiona el contenidor d'elements
        """
```

3. Cal implementar el mètode *searchSymbol(name)*, tal que, donat el nom de l'element, en mostri la informació associada. Únicament es faran cerques pel nom de l'element químic.

```
def searchSymbol(self,name):
        """
        Mostra per pantalla la informació associada a l'element de nom name
        """
```

4. Cal implementar el mètode *showAll*, tal que proporcioni tota la informació dels elements emmagatzemats a la taula.

```
def showAll(self):
        """
        Mostra per pantalla la informació associada de tots els elements de la taula
        """
```

A continuació segueix un exemple de funcionament, i dels resultats esperats del mètode *showAll* i del mètode *searchSymbol*.

```
t=Taula("nameFile.csv")
t.showAll()
t.searchSymbol("Sodium")
```

Resultats esperats:

```
List of elements in the table .....
Element(Oxygen,8,O,16.0)
Element(Sodium,11,Na,22.98)
Element(Potassium,19,K,39.1)
Element(Nitrogen,7,N,14.01)
Element(Sulfur,16,S,32.06)
Element(Carbon,6,C,12.01)
Element(Hydrogen,1,H,1.01)
Element(Phosphorus,15,P,30.98)

Info for element Sodium .....
Element(Sodium,11,Na,22.98)
```

Apartat c) Considereu per exemple la molècula aigua, amb fórmula química H_2O. A continuació se us demana gestionar la classe *Molècula*. Per tant, us caldrà gestionar un contenidor òptim que permeti emmagatzemar els àtoms dels elements químics dels quals està composada. En aquest cas, cal gestionar l'accés òptim per símbol.

1. Cal implementar el mètode constructor de la classe *Molècula*

```
def __init__(self,name):
    """
    Crea una molècula de nom name i inicialitza el contenidor d'elements
    """
```

2. Cal implementar el mètode *add*, que afegeix un element a la *Molècula*

```
def add(self,Element):
    """
    afegeix l'Element al contenidor de la molècula
    """
```

3. Cal implementar el mètode *addAtom*, que reb 2 paràmetres: *n* i *symbol*, i ha d'afegir n àtoms de l'element símbol a la molècula

```
def addAtom(self,symbol,n):
    """
    afegeix n àtoms a l'elements amb símbol symbol de la molècula
    """
```

4. Cal implementar el mètode *atoms*, tal que ha de retornar la informació dels elements que composen la molècula, juntament amb el seu nombre d'àtoms

```
def atoms(self):
    """
    mostra per pantalla la informació dels elements que composen la molècula i el seu nombre d'àtoms
    """
```

5. Cal implementar el mètode *weight*, tal que ha de retornar la massa atòmica de la molècula. És a dir, la suma dels pesos de cada àtom en la molècula. Per exemple, donats els pesos

dels Elements Hidrogen i Oxigen són 1.01 i 16.0 respectivament, i donat que en la molècula aigua hi ha 2 àtoms d'hidrogen i un àtom d'oxigen, la massa de la molècula ha de ser 1.01*2+16.0*1=18.02

```
def weight(self):
    """
    retorna la massa atòmica de la molècula
    """
```

A continuació segueix un exemple de crides i el resultat esperat.

Exemple d'execució:

```
if __name__=='__main__':
    Water=Molecula("Water")
    hidrogen=Element("Hidrogen",1,"H",1.01)
    oxigen=Element("Oxigen",8,"O",16.0)
    Water.add(hidrogen)
    Water.add(oxigen)
    Water.addAtom("H",2)
    Water.addAtom("O",1)
    Water.atoms()
    print Water.weight()
```

Resultat esperat:

```
Molecule information Water .....
Water H2 O1
Molecule weight Water .....
18.02
```

Exercici 7.17 Efectuar el diagrama de classes necessari per poder dur a terme aquest operatiu: Dins de l'operatiu de CaixerEPSEM s'emmagatzema com a part de cada compte bancari les dades que pertanyen al titular i el saldo d'aquesta. La informació sobre les dades del titular de cada compte són, entre altres, el seu nom l'adreça, l'edat, el telèfon i el seu NIF, el qual serveix per identificar el client. A més a cada compte hi haurà el saldo d'aquest i el crèdit associat, que és la quantitat de diners que el client pot tenir en negatiu ("Números vermells"). Una vegada creat un client es podran modificar totes les dades d'aquest menys el NIF, que és el que serveix per identificar-lo. Dels comptes es podrà ingressar i treure diners. S'ha de tenir en compte que els menors d'edat poden ser propietaris d'un compte corrent però sols poden ingressar, no treure diners. Finalment, en un compte es poden afegir fins a un màxim de cinc titulars.

Exercici 7.18 Una clínica dental demana assessorament TIC per tal de portar un sistema de registre dels seus pacients. A continuació segueix el llistat de requeriments que han fet arribar.

Per cada pacient cal emmagatzemar el seu nom, adreça i un número de mòbil. A cada pacient se li assigna un número únic de set dígits. El sistema necessita portar un recompte dels pacients que té en tot moment. El pacients poden demanar una visita amb un dentista en concret; en tal cas, el sistema ha d'emmagatzemar la data de la visita i si el pacient va ser atès o no. S'enviarà

al pacient un missatge automàtic de recordatori 2 dies previs a la visita. Un cop finalitzada la visita, el dentista ha de modificar el sistema afegint el cost del tractament.

Hi ha dos tipus d'empleats: els recepcionistes i els dentistes. D'ambdòs, el sistema ha d'emmagatzemar les seves particularitats. Per tots els empleats, cal un número d'empleat de quatre dígits, el nom, adreça, gènere, número de telèfon i el parent més proper. Els dentistes estan qualificats, per tant el sistema ha de guardar la seva qualificació més alta, la data en que es va obtenir i el número de registre del Col·legi de Dentistes.

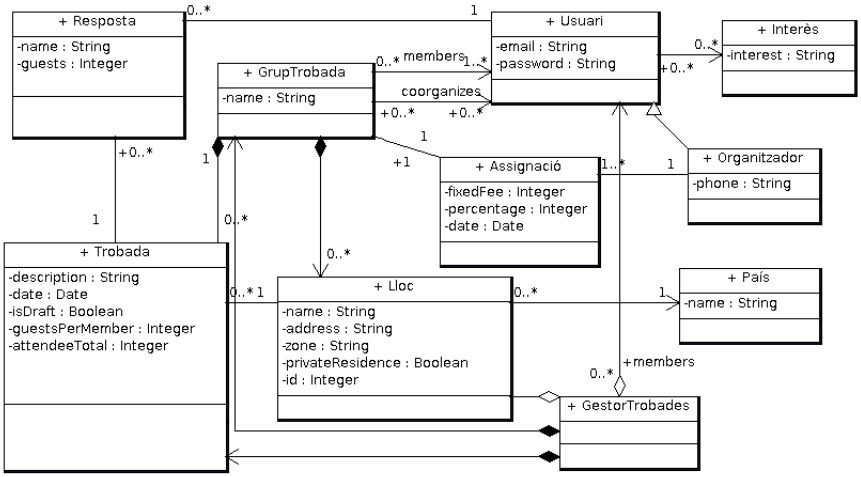
Al final de cada setmana cal generar una llista estadístiques de visites. Aquest llistat ha de ser un resum de quans pacients es van presentar i quants no es van presentar. Si el pacient no es presenta a una visita, se li carregarà una quantitat fixa de diners.

Tots els recepcionistes han de realitzar anualment un curs de primers auxilis. el sistema ha de gestionar la data de l'últim cop que van realitzar el curs i el nom del professor que va impartir el curs.

Se us demana que realitzeu el diagrama de classes que permeti resoldre el problema plantejat, amb especificació dels atributs i la signatura dels principals mètodes. No calen mètodes constructors ni accessors/modificadors. Únicament cal la signatura, no la implementació.

Exercici 7.19 Donat el diagrama UML de la figura 7.1, que segueix, creeu l'esquelet de les classes, on es vegi clarament la materialització de cadascuna de les relacions entre classes, incloent el mètode constructor de cadascuna d'elles.

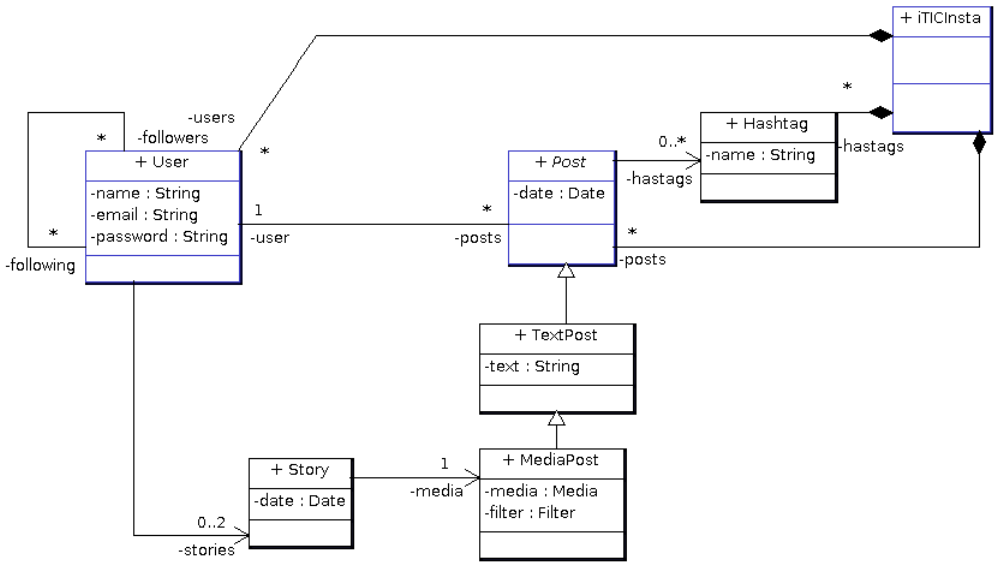
Figure 7.1: Meetings UML



Exercici 7.20 Donat el diagrama UML de la figura 7.2, corresponent a la xarxa social iTICInsta que segueix, creeu l'esquelet de les classes, on es vegi clarament la materialització de cadascuna de les relacions entre classes, incloent el mètode constructor de cadascuna d'elles.

Exercici 7.21 Donat el diagrama UML de la figura 7.3, creeu l'esquelet de les classes incloent el mètode constructor de cadascuna d'elles.

Figure 7.2: iTIC Insta UML



Exercici 7.22 Defineix tècnicament i posa un exemple dels següents conceptes en Programació Orientada a Objectes.

1. Mètode constructor
2. Sobrecàrrega de mètodes
3. Redefinició de mètodes
4. Polimorfisme

Exercici 7.23 Se't proporcionen les següents definicions (incompletes) de classes. El codi proporcionat pretén gestionar estudiants que es matriculen en una entitat.

```
import datetime
class Person(object):
    def __init__(self, name):
        #TO DO
    def getLastName(self):
        return self.lastName
    def setBirthday(self, birthDate):
        self.birthday = birthDate
    def getAge(self):
        #TO DO
    def __str__(self):
        return self.getLastName()
        #TO DO
```

```
me = Person('John Guttag')
him = Person('Barack Hussein Obama')
her = Person('Madonna')
him.setBirthday(datetime.date(1961, 8, 4))
her.setBirthday(datetime.date(1958, 8, 16))
print her.getAge()
print him.getAge()
pList = [me, him, her]
print 'The people in pList ordered by Last Name are:'
for p in pList:
    print '  ' + str(p)
```

```
#Resultats execució:
60.8739726027
57.904109589
The people in pList are:
  Last Name: Guttag Name: John Guttag
  Last Name: Madonna Name: Madonna
  Last Name: Obama Name: Barack Hussein Obama
```

[**Apartat a**] Se't demana la implementació òptima dels mètodes marcats amb TO DO, així com el necessari per obtenir el llistat ordenat de persones per LastName i els resultats d'execució especificats.

[**Apartat b**] Es necessita especialitzar la classe anterior *Person* per tal que emmagatzemi els estudiants i assignar-los un codi automàtic quan es matriculen. Se't demana la implementació de la nova classe **eMITPerson** que satisfaci els requeriments que segueixen.

```
p1 = eMITPerson('Barbara Beaver')
print p1
p2 = eMITPerson('Sue Yuan')
print p2
p3 = eMITPerson('Sue Yuan')
print p3
```

```
#Resultats execució:
Last Name: Beaver Name: Barbara Beaver Code: 0
Last Name: Yuan Name: Sue Yuan Code: 1
Last Name: Yuan Name: Sue Yuan Code: 2
```

[**Apartat c**] Es necessita especialitzar la classe anterior *eMITPerson* per tal que emmagatzemi els estudiants ja graduats (G) i els no graduats encara (UG), tenint constància pels no graduats, dels anys que porten col·laborant amb la universitats. Se't demana la implementació de les noves classes **UG** i **G** que satisfacin els requeriments que segueixen.

```
ug1 = UG('Jane Doe')
ug2 = UG('J Donov')
try:
    ug1.setYear(4)
    ug2.setYear(6)
except Exception as e:
    print e
print ug1
print ug2
g1 = G('Ryan Jackson')
print g1
```

```
#Resultats execució:
Too many. More than 4
Last Name: Doe Name: Jane Doe Code: 0 Years working: 4
Last Name: Donov Name: J Donov Code: 1 Years working: No info
Last Name: Jackson Name: Ryan Jackson Code: 2
```

[**Apartat d**] Se't demana implementar el gestor d'estudiants matriculats en un curs (identificat per un número), que simuli el comportament que segueix. Acaba els mètodes especificats com a TO DO i aquells que calguin per al funcionament correcte (pot implicar afegir mètodes en classes anteriors). Es valorarà la no repetició de codi.

```
class CourseList(object):
    def __init__(self, number):
        self.number = number
        self.students = []
    def addStudent(self, who): #TO DO
    def remStudent(self, who): #TO DO
    def allStudents(self): #TO DO
    def ugs(self): #TO DO
```

```
m1 = eMITPerson('Barbara Beaver')
ug1 = UG('Jane Doe')
ug2 = UG('Jane Doe')
g1 = G('Mitch Peabody')
g2 = G('Ryan Jackson')
g3 = G('Jenny Liu')
ug3 = UG('Only one')
SixHundred = CourseList('6.00')
try:
    SixHundred.addStudent(ug1)
    SixHundred.addStudent(g3)
    SixHundred.addStudent(g1)
    SixHundred.addStudent(ug3)
    SixHundred.addStudent(ug2)
except Exception as e: print e
try:
    SixHundred.remStudent(g3)
    SixHundred.addStudent(m1)
except Exception as e: print e
print 'Students'
for s in SixHundred.allStudents():
    print s
print 'Change Class test'
for s in SixHundred.allStudents():
    print s
    if s == ug1:
        SixHundred.remStudent(ug2)
        SixHundred.addStudent(g2)
print 'Undergraduates'
for u in SixHundred.ugs():
    print u
```

```
#Resultats execució:
Duplicate student....Jane Doe
Not a student----Barbara Beaver
Students
Last Name: Doe Name: Jane Doe Code: 1 Years working:  No info
Last Name: Peabody Name: Mitch Peabody Code: 3
Last Name: one Name: Only one Code: 6 Years working:  No info
Change Class test
Last Name: Doe Name: Jane Doe Code: 1 Years working:  No info
Last Name: one Name: Only one Code: 6 Years working:  No info
Last Name: Jackson Name: Ryan Jackson Code: 4
Undergraduates
Last Name: one Name: Only one Code: 6 Years working:  No info
```

Exercici 7.24 Classes i mètodes. Se t'ha encomanat la versió software d'un popular joc de cartes. L'objectiu de cada jugador és superar nivells. Un jugador supera un nivell cada cop que derrota un altre jugador en la batalla. El guanyador de cada batalla és el jugador que té més força. La força correspon a la suma de: el nivell actual del jugador i la força de cadascun de les eines que aquest tingui. En cas d'empat, guanya el jugador més desafiant. A continuació es proporciona el codi incomplet de les classes a desenvolupar i se't demana que responguis cadascun dels apartats que segueixen.

```
class Player(object):
    name = ""; level = 1
    def __init__(self, name):
        self.name = name
        self.equipment = []
        self.potions = []
    def currentStrength(self):
        s=sum([i.strength() for i in self.equipment])
        return self.level+s
    def battle(self, other):
        my = self.currentStrength()
        others = other.currentStrength()
        print self.name, "challenges", other.name
        print self.name, "has a total strength of", my
        print other.name, "has a total strength of", others
        if my > others:
            print self.name, "wins!"
            self.level += 1; other.level -= 1
        else:
            print other.name, "wins!"
            other.level += 1; self.level -= 1
```

```
class Item(object):
    coreStrength = 0
    def __init__(self, name):
        self.name = name

    def strength(self):
        return self.coreStrength

class BucklerOfSwashing(Item):
    coreStrength = 2
```

Apartat a) Quin és el resultat de l'execució del fragment de codi que es troba a continuació?

```
if __name__=='__main__':
    alyssa=Player("A. Python Hacker")
    ben=Player("Ben Bitdiddle")
    ben.battle(alyssa)
```

Apartat b) Se't demana que implementis la classe Poció (Potion). Les Pocions són Items que només poden ser utilitzades un cop. Si una persona es troba en una batalla i no ha usat una poció, aquesta es consumeix durant la batalla, contribuïnt a la força global. Les Potion tenen un mètode usedUp, tal que retorna un booleà indicant si la poció s'ha utilitzat o no. Cal que la classe Potion hereti de la classe Item. Es vol que el comportament sigui com el que segueix, per tant, és possible que la classe Potion tingui subclasses especificades en la demo.

```
>>> alyssa=Player("A. Python Hacker")
>>> ben=Player("Ben Bitdiddle")
>>> ben.battle(alyssa)
Ben Bitdiddle challenges A. Python Hacker
Ben Bitdiddle has a total strength of 1
A. Python Hacker has a total strength of 1
A. Python Hacker wins!
>>> alyssa.equipment.append(BucklerOfSwashing("excalibyr"))
>>> ben.potions.append(FreezingExplosive())
>>> alyssa.battle(ben)
A. Python Hacker challenges Ben Bitdiddle
A. Python Hacker has a total strength of 4
Ben Bitdiddle has a total strength of 5
Ben Bitdiddle wins!
>>> alyssa.battle(ben)
A. Python Hacker challenges Ben Bitdiddle
A. Python Hacker has a total strength of 3
Ben Bitdiddle has a total strength of 1
A. Python Hacker wins!
```

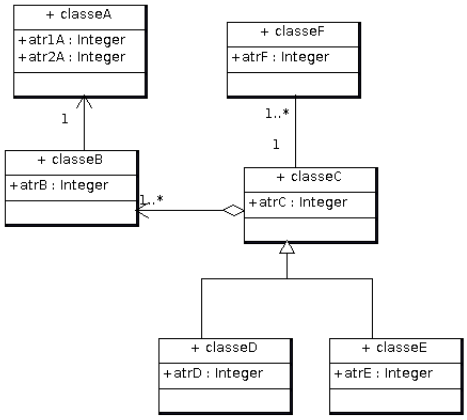
Adona't que en les darreres 2 batalles, Alyssa pert la primera perquè Ben té una Potion. Per això en la següent batalla, Alyssa guanya perquè el Ben ja no té la força de la potion. Se't demana: **Apartat b.1)** La implementació de la classe *Potion*. Ha d'heretar de la classe *Item* i no es pot repetir codi innecessariament. **Apartat b.2)** Modificar el mètode *currentStrength* de la classe *Player* per tal de que realitzi el comportament mostrat.

Apartat c) Encara que algunes pocions s'usen només un cop, estaria bé disposar d'una poció que no s'acabi mai. Se't demana que dissenyis una nova classe, de nom **InfinitePotion**, que hereti de la classe *Potion*. Aquesta ha de ser una *Potion* que mai s'acabi. Implementa aquesta classe i comprova'n el funcionament. És possible que la classe InfinitePotion tingui subclasses especificades en la demo.

```
>>> alyssa=Player("A. Python Hacker")
>>> ben=Player("Ben Bitdiddle")
>>> ben.battle(alyssa)
Ben Bitdiddle challenges A. Python Hacker
Ben Bitdiddle has a total strength of 1
A. Python Hacker has a total strength of 1
A. Python Hacker wins!
>>> alyssa.equipment.append(BucklerOfSwashing("excalibyr"))
>>> ben.potions.append(FreezingCanada())
>>> alyssa.battle(ben)
A. Python Hacker challenges Ben Bitdiddle
A. Python Hacker has a total strength of 4
Ben Bitdiddle has a total strength of 5
Ben Bitdiddle wins!
>>> alyssa.battle(ben)
A. Python Hacker challenges Ben Bitdiddle
```

```
A. Python Hacker has a total strength of 3
Ben Bitdiddle has a total strength of 6
Ben Bitdiddle wins!
```

Figure 7.3: UML Class Diagram



8 Recursivitat

EXERCICI 8.1 Dissenya i implementa una funció recursiva que calcula la suma dels enters entre 1 i m.

EXERCICI 8.2 Dissenyeu una funció recursiva tal que, donada una paraula, retorna la seva longitud.

EXERCICI 8.3 Dissenyeu una funció recursiva tal que, donada una paraula, compta quantes lletres a conté.

EXERCICI 8.4 Dissenyeu una funció recursiva tal que, donada una llista ordenada d'enters l i un enter x contingut a l, retorna la posició que ocupa x en l.

EXERCICI 8.5 Dissenyeu una funció recursiva que determini si una cadena s és palíndroma (és a dir simètrica).

EXERCICI 8.6 Dissenyeu una funció recursiva tal que, donada una llista l, retorna la seva inversa. Per exemple, donada l=['a', 'b', 'c'] retorna la llista ['c', 'b', 'a'].

EXERCICI 8.7 Dissenyeu una funció recursiva per calcular F_n essent n un enter i F_n l'n-èssim valor de la seqüència de Fibbonacci. La seqüència de Fibonacci (mireu a la wikipedia per què és important) es defineix així:

$$F_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F_{n-1} + F_{n-2} & n \geq 2 \end{cases}$$

EXERCICI 8.8 Escriu una funció **recursiva** de nom *exponent*, tal que, donada una base i un exponent positiu, retorni el resultat de calcular recursivament base elevat a exponent.

```
def exponent(base,e):
    """
    Retorna el resultat de calcular recursivament base elevat a e, e>=0
    >>> exponent(0,5)
    0
    >>> exponent(2,4)
    16
    >>> exponent(3,0)
    1
    """
```

EXERCICI 8.9 Escriu una funció **recursiva** tal que, donat un nombre en base decimal, permeti convertir-lo a un nombre binari. No es poden utilitzar piles ni llistes (en cas que s'utilitzin, l'exercici no es puntuarà.)

```
def decimal2binari(n):
    """
    retorna un string resultant de la conversio de l'enter n>=0 a binari
    >>> decimal2binari(22)
    '10110'
    >>> decimal2binari(0)
    '0'
    """
```

EXERCICI 8.10 La llista de k en k. Escriu una funció **recursiva** de nom *novallista*, tal que, donada una llista d'enters, retorni una nova llista on cada element resulta d'agrupar els elements originals agafats de k en k, $k \geq 1$, i representar-los en la nova llista com un nou element que té per valor la mitjana dels originals. Exemples d'execució:

```
def novaLlista(llista,k):
    """
    >>> novaLlista([1,2,3,4,5,6,7,8,9],2)
    [1, 3, 5, 7, 9]
    >>> novaLlista([1,2,3,4,5,6,7,8,9],4)
    [2, 6, 9]
    >>> novaLlista([1,2,3,4,5,6,7,8,9],1)
    [1, 2, 3, 4, 5, 6, 7, 8, 9]
    >>> novaLlista([],3)
    []
    """
```

EXERCICI 8.11 [**Apartat a**] Escriu la funció **recursiva**, de nom *ordenaD*, tal que, donada una llista no ordenada, *laLlista*, retorni la llista resultant d'ordenar-la descendentment, seguint l'estratègia que es detalla a continuació. L'element més gran anirà a la primera posició de la nova llista, i després cal realitzar l'ordenació amb la resta de la llista, de manera successiva fins que tota ella estigui ordenada. Podeu utilitzar els mètodes *len* i *remove* sobre llistes, i també el mètode max (l'ús de qualsevol altre mètode predefinit de Python o la implementació d'una funció no recursiva comportarà puntuació nul·la). [**Apartat b**] Escriviu el resultat dels doctest que segueixen, i afegeix un docstring.

```
def nose(x,param=0,div=0):
    """
    >>> nose(8)
    #3.1
    >>> nose(7)
    #3.2
    """
    if div>=3:
        return False
    else:
        if x-param!=0:
            if x%(x-param)==0:
```

```
            div+=1
            param+=1
            return nose(x,param,div)
        else:
            param+=1
            return nose(x,param,div)
    else:
        return True
```

EXERCICI 8.12 Dissenyeu una funció recursiva tal que, donats dos enters x i y calcula x elevat a y. Naturalment no podeu usar l'operació de potència. Mireu de minimitzar el nombre de multiplicacions.

EXERCICI 8.13 Dissenyeu la funció recursiva sumd(n) que, donat un enter positiu n, retorna la suma dels seus dígits. Per exemple una crida a sumd(234) hauria de retornar 9.

EXERCICI 8.14 Dissenyeu la funció recursiva prefix(l,n) que, donat un enter positiu n i una llista d'enters l, retorna **True** sii existeix una subllista l[0:x] tal que **sum**(l[0:x])==n.

EXERCICI 8.15 Dissenyeu una funció recursiva per calcular la funció $F(n) = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$

EXERCICI 8.16 Donada la següent funció recursiva, justifiqueu el resultat de cadascun dels doctests següents. Pel primer doctest, justifiqueu quantes crides recursives s'efecutaran.

```
def sumatori(llista):
    """
    >>> sumatori([1,3,5,7,9])
    # a contestar
    >>> sumatori([])
    # a contestar
    """
    if len(llista) == 1:
        return llista[0]
    else:
        return llista[0] + sumatori(llista[1:])
```

EXERCICI 8.17 Escriu una funció **recursiva** de nom *nestedListSum*, tal que, donada una llista de llistes niuades, retorni el resultat de calcular recursivament la suma d'elements inclosos en la llista/subllistes.

```
def nestedListSum(NL):
    """
    >>> nestedListSum([2,4,8])
    14
    >>> nestedListSum([[1,4],[2,8],[3,1,1]])
    20
    >>> nestedListSum([[[2],[1]],[2,4],[[3,4],[1,0,-1]]])
    16
    """
```

EXERCICI 8.18 Donades les definicions de les següents funcions recursives, **justifiqueu** el resultat de cadascun dels doctestos.

Apartat a)

```
def misteri(n):
    """
    >>> a = misteri(10)
    Apartat a1)
    >>> a
    Apartat a2)
    """
    print(n)
    if n == 1:
        return 1
    elif n % 2 == 0:
        return 1 + misteri(n / 2)
    else:
        return 1 + misteri(3 * n + 1)
```

Apartat b)

```
def f(s):
    """
    >>> f('mat')
    Apartat b1)
    >>> f('math')
    Apartat b2)
    if len(s) <= 1:
        return s
    return f(f(s[1:])) + s[0]
```

Apartat c)

```
def esProva(m):
    """
    >>> esProva(5)
    Apartat c1)
    >>> esProva(24)
    Apartat c2)
    >>> esProva(0)
    Apartat c3)
    """
    def prova(m,j):
        if j==1:
            return True
        else:
            return m%j!=0 and prova(m,j-1)
    return prova(m,m-1)
```

Apartat d) Escriviu el resultat d'execució

```
def f(L):
```

```
    result = []
    for e in L:
        if type(e) != list:
            result.append(e)
        else:
            return f(e)
    return result

print f('3')
print f([1, [[2, 'a'], ['a','b']], (3, 4)])
print f(3)
```

EXERCICI 8.19 Apartat a) Escriu una funció **recursiva** que obtingui el valor mes petit d'una llista d'elements. Òbviament, no es pot utilitzar la funció *min* de Python.

```
def minim(llista):
    """
    >>> minim([2,34,-1,-4])
    -4
    >>> minim([])
    'no data'
    """
```

Apartat b) Escriu una funció **recursiva**, de nom *ordena*, tal que, donada una llista no ordenada, *laLlista*, retorni la llista resultant d'ordenar-la ascendentment, seguint l'estratègia que es detalla a continuació. L'element més petit anirà a la primera posició de la nova llista, i després cal realitzar l'ordenació amb la resta de la llista, de manera successiva fins que tota ella estigui ordenada. Si ho considereu necessari, podeu utilitzar els mètodes *len* i *remove* sobre llistes (l'ús de qualsevol altre mètode predefinit de Python comportarà puntuació nul·la). Podeu utilitzar la funció *minim* que heu implementat en l'Apartat a).

```
def ordena(laLlista):
    """
    >>> ordena([-4,22,11,0,2,8,99,-2])
    [-4, -2, 0, 2, 8, 11, 22, 99]
    >>> ordena([7, 6, 5, 4, 3, 2, 1])
    [1, 2, 3, 4, 5, 6, 7]
    """
```

EXERCICI 8.20 Donada les següents funcions recursives, a) Justifiqueu el resultat de cadascun dels seus doctestos b) Pel *doctest2* i *doctest5*, justifiqueu quantes crides recursives s'efecutaran c) Digueu quin és el cost assimptòtic en cas pitjor de dita funció.

```
Apartat a)
def f(L):
    """
    >>> f('3')
    #doctest1
    >>> f([1, [[2, 'a'], ['a','b']], (3, 4)])
    #doctest2
    >>> f(3)
    #doctest3
    """
```

```
    result = []
    for e in L:
        if type(e) != list:
            result.append(e)
        else:
            return f(e)
    return result
```

```
Apartat b)
def f(s):
    """
    >>> f('electrode')
    #doctest4
    >>> f('sensor')
    #doctest5
    """
    if len(s) <= 1:
        return s
    return f(f(s[1:])) + s[0]
```

Exercici 8.21 Donada les següents funcions recursives, a) Justifiqueu el resultat de cadascun dels seus doctestos b) Pel *doctest2* i *doctest5*, justifiqueu quantes crides recursives s'efecutaran

```
Apartat a)
def mistery(s):
    """
    >>> mistery('program')
    #doctest1
    >>> mistery('python')
    #doctest2
    """
    if len(s) <= 1:
        return s
    return mistery(mistery(s[1:])) + s[1]
```

```
Apartat b)
def noIdea(L):
    """
    >>> noIdea('88')
    #doctest3
    >>> noIdea([11, [[22, 'ab'], ['ba','bx']], (31, 41)])
    #doctest4
    >>> noIdea(44)
    #doctest5
    """
    result = []
    for e in L:
        if type(e) != list:
            result.append(e)
        else:
            return noIdea(e)
    return result
```

Exercici 8.22 Dissenya òptimament la funció recursiva *suma*, tal que donat un nombre natural, retorni la suma dels seus dígits. Només es pot utilitzar la funció str/int/len. Qualsevol altra funció predefinida comportarà una puntuació nul·la.

```
def suma(x):
    """
    >>> suma(371)
    11
    >>> suma(3)
    3
    """
```

Exercici 8.23 Dissenya òptimament la funció recursiva *compress*, que permeti comprimir un text, de la següent manera, cada vegada que una lletra es repeteix, es guarda la indicació de quants cops s'ha repetit (si és superior a 1).

```
def compress(s):
    """
    >>> compress('abc')
    'abc'
```

```
    >>> compress('abbcddd')
    'a2bc3d'
    """
```

EXERCICI 8.24 Recursivitat. [Apartat a] Dissenya òptimament la funció booleana recursiva *esPrimer*, tal que donat un nombre natural, retorni True si es tracta d'un nombre primer, i False en cas contrari. Un nombre és primer, si i només si, és divisible per 1 i per ell mateix.

```
def esPrimer(n, i = 2):
    """
    >>> esPrimer(15)
    False
    >>> esPrimer(13)
    True
    """
```

[Apartat b] Dissenya òptimament la funció recursiva *crack*, tal que, donada una llista de caràcters, s'encarrega de generar totes les contrasenyes possibles a partir d'aquests caràcters. És a dir, permet generar totes les permutacions possibles de paraules utilitzant els caràcters de la llista, amb repeticions i fins a una longitud donada. Es poden utilitzar funcions addicionals.

```
Exemple 1:
Input : llista = [a, b],
        len = 2.
Output :
a b aa ab ba bb

Exemple 2:
Input : llista = [a, b, c],
        len = 3.
Output :
a b c aa ab ac ba bb bc ca cb cc aaa aab aac aba abb
abc aca acb acc baa bab bac bba bbb bbc bca bcb bcc
caa cab cac cba cbb cbc cca ccb ccc
```

EXERCICI 8.25 Donada les següents funcions recursives, a) Justifiqueu el resultat de cadascun dels seus doctestos b) Pel *doctest2* i *doctest5*, justifiqueu quantes crides recursives s'efectuaran

```
Apartat a)
def mistery(s):
    """
    >>> mistery('program')
    #doctest1
    >>> mistery('python')
    #doctest2
    """
    if len(s) <= 1:
        return s
    return mistery(mistery(s[1:])) + s[1]
```

```
Apartat b)
def noIdea(L):
    """
    >>> noIdea('88')
    #doctest3
    >>> noIdea([11, [[22, 'ab'], ['ba','bx']], (31, 41)])
    #doctest4
    >>> noIdea(44)
    #doctest5
    """
    result = []
```

```
    for e in L:
        if type(e) != list:
            result.append(e)
        else:
            return noIdea(e)
    return result
```

Exercici 8.26 Dissenyeu una funció recursiva tal que donada una llista l d'elements diferents i un natural n menor o igual que la mida de la llista, torna la llista de totes les subllistes possibles d'l de mida n.

Exercici 8.27 (Prof. Michael Main, U. Colorado) Dissenyeu una funció recursiva que dibuixi un dibuix fractal de mida n de la següent forma:

```
*
* *
  *
* * * *
    *
    * *
      *
* * * * * * * *
        *
        * *
          *
        * * * *
            *
            * *
              *
```

Penseu en una funció pinta(n,i) en la que n és una potència de 2 no nuł.la, i una crida a la funció dibuixa un patró basat en l'exemple anterior tal que la línia més llarga té n estrelles i comença a la columna i. Per exemple, la figura anterior s'ha obtingut amb la crida pinta(8,0).

Pistes: Penseu recursivament. La funcio és molt curta. Penseu sobre l'estructura recursiva de la figura i determineu quin és el patró base de recursió.

Exercici 8.28 iTICInsta està gestionant una aplicació per tal de gestionar els fans que segueixen un usuari, bloquejar-los si cal i gestionar els posts amb hashtags de les teves publicacions. A continuació et passen el prototipus de classes per iTICInsta per gestionar la informació i un exemple de funcionament. Cada Usuari té un nick, un conjunt de fans i un conjunt de posts. Cada post té un contingut i un conjunt de hashtags. Aquests hashtags poden incloure noms de nick (usuaris etiquetats).

```
class User(object):
    def __init__(self,nom,email):
        self.nick=nom
        self.email=email
        self.fans={}
        self.posts=[]
        self.__blocked=[]
    def __str__(self):
        #TO DO
```

```
    def addFan(self,p):
        self.fans[p.nick]=p
        #TO FINISH
```

```
    def checkFanInfo(self,fan):
        #TO DO
    def listFans(self):
        #TO DO
```

```
    def blockUser(self, user):
        #TO DO
    def listBlocked(self):
        #TO DO
```

Apartat a) Implementar els mètodes que permeten gestionar els usuaris fans d'un usuari i bloquejar-los si és el cas. A tal efecte, haureu de gestionar els mètodes

1. str: Mostra la informació completa d'un usuari i dels nicks dels seus fans
2. listFans: permet obtenir tota la informació dels fans d'un usuari
3. blockUser: ha de permetre bloquejar un usuari, tan si és fan com si no. Ha d'esborrar un fan de la collection de fans (si és fan). Caldrà afegir-ne el nick a la collection de fans bloquejats.
4. listBlocked: permet obtenir els nicks dels fans bloquejats
5. addFan: ha de permetre afegir un fan a la collection de fans de l'user. No es pot afegir un fan que s'ha bloquejat prèviament (per tant apareix a la collection de blocked de l'usuari en qüestió).
6. checkFanInfo: permet consultar tota la informació d'un fan de l'usuari que no s'hagi bloquejat, a partir del nick del fan. I el fan a consultar ha d'estar a la seva collection de fans.

```
if __name__=='__main__':
    g=User("pere","pere@itic.com")
    h=User("anna","anna@itic.com")
    l=User("maria","maria@itic.com")
    k=User("dimoni","dimoni@itic.com")
    g.addFan(h)
    print g #1
    g.addFan(l)
    g.listFans() #2
    g.blockUser(k) #3
    k.addFan(g) #4
    print k #5
    print g.listBlocked() #6
    g.checkFanInfo("maria") #7
```

```
#Resultats execució
#1  nick --> pere
    email --> pere@itic.com
    Fans_list_nicks --> anna
    No posts yet
#2  nick --> anna
    email --> anna@itic.com
    No fans yet No posts yet
    nick --> maria
    email --> maria@itic.com
    No fans yet No posts yet
#3  Not existing fan
#4  Not allowed
#5  nick --> dimoni
    email --> dimoni@itic.com
    No fans yet No posts yet
#6  Blocked users --> dimoni
#7  nick --> maria
    email --> maria@itic.com
    No fans yet
    No posts yet
```

Apartat b) Afegir el mètode $post(self)$ a la classe User, tal que permeti gestionar un post.

També us cal implementar el mètode *str* de la classe *Post*, per visualitzar-lo correctament.

Un post consisteix en un string, el qual conté a la part final els hashtags amb la sintaxi #nomHashtag i els usuaris etiquetats amb la sintaxi @*usuari*. A continuació segueixen els jocs de prova.

Es pot etiquetar qualsevol usuari (no cal que siguin fans), excepte aquells que hàgim bloquejat.

```
import time
class Post(object):
    def __init__(self,text,date):
```

```
        self.content=text
        self.date=date
        self.hashtags=[]
        self.friends=[]

    def __str__(self):
      #TO DO

    def addHashtag(self,h):
        self.hashtags+=[h]

    def addFriends(self,f):
        self.friends+=[f]

class User(object):
    def post(self,p):
       dataActual = time.strftime("%c")
       #TO FINISH

    def printPostsDate(self):
        for p in self:
            print p

if __name__=='__main__':
    g=User("pere","pere@itic.com")
    l=User("maria","maria@itic.com")
    g.addFan(l)
    g.blockUser(l)
    g.post("Next starting project #newproject #newversion #workingcopy @eva @maria @jordina")
    g.post("Holidays are comming #finishing_project")
    g.post("End of exams @jordi @eva @maria")
    g.post("no way")
    g.printPostsDate()

#Resultats d'execució
Content -->Next starting project
Date--> Mon Jun 12 10:29:47 2017
Hashtags--> #newproject  #newversion  #workingcopy
Labeled users--> @eva @jordina

Content -->Holidays are comming
Date--> Mon Jun 12 10:29:47 2017
Hashtags--> #finishing_project

Content -->End of exams
Date--> Mon Jun 12 10:29:47 2017
Labeled users--> @jordi @eva

Content -->no way
Date--> Mon Jun 12 10:29:47 2017
```

Exercici 8.29

Has acceptat treballar a Facebook-EPSEM i estàs treballant en una nova app de nom "TotSobreMi". L'objectiu de dita aplicació és mostrar-te els comentaris de les fotos de tu mateix que han pujat altres persones. Hauràs de resoldre el problema en tres passos/apartats. L'apartat b) usa el mètode de l'apartat a), i l'apartat c) usa el b). A continuació et passen el prototipus de classes per Facebook-EPSEM per gestionar la informació i un exemple de funcionament. Cada Usuari té un nom, un conjunt d'amics i un conjunt de fotografies. Cada foto té una imatge i un conjunt de tags. Aquests tags poden incloure noms d'usuari (informació rellevant per la nova app).

```
class User(object):
    def __init__(self,nom):
        self.name=nom
        self.friends=[]
        self.photos=[]

    def showPhotos(self):
        print "Photos user",self.name
        for p in self.photos:
            print "Name of image",p.image,"list of tags"
            for t in p.tags:
                print t

    def addPhoto(self,p):
        if p not in self.photos:
            self.photos.append(p)

class Photo(object):
    def __init__(self,image):
        self.image=image
        self.tags=[]

    def addTag(self,t):
        self.tags.append(t)

    def __eq__(self,other):
        return self.image==other.image

if __name__=='__main__':
    g=User("pere")
    p=Photo("a.jpg")
    p1=Photo("b.jpg")
    g.addPhoto(p)
    g.addPhoto(p1)
    p.addTag("pere Fantastic")
    p1.addTag("wow pere")
    p1.addTag("quants anys...")
    g.showPhotos()

#Resultat execució
Photos user pere
Name of image a.jpg list of tags
pere Fantastic
Name of image b.jpg list of tags
wow pere
quants anys...
```

Apartat a) Implementar el mètode $myTags(self)$ a la classe User, tal que ha de retornar una

llista dels tags tals que contenen el name de l'user en les photos d'aquest user. Seguint l'exemple anterior, a continuació segueix quin hauria de ser el resultat d'invocació del mètode.

```
print g.myTags()
```

```
#Resultat d'execució
['pere Fantastic', 'wow pere']
```

Apartat b) Implementar el mètode $tagsAmics(self)$ a la classe User, tal que retorni un llistat de tags tals que contenen el nom de l'user en els tags de les seves fotos, juntament amb aquells tags on apareix el seu nom en el llistat de fotos dels seus amics directes. Cal utilitzar el mètode definit en l'Apartat a), i si us cal, podeu canviar-ne la signatura del mètode. A continuació segueix exemple d'execució.

```
a=User("maria")
t=Photo("c.jpg")
a.addPhoto(t)
t.addTag("holidays with pere")
g.addFriend(a)
print g.tagsAmics()
```

```
#Resultat d'execució
['pere Fantastic', 'wow pere', ['holidays with pere']]
```

Apartat c) Implementar el mètode $totSobreMi(self)$ a la classe User, tal que retorni un llistat de tags tals que contenen el nom de l'user en els tags de les seves fotos, juntament amb aquells tags on apareix el seu nom en el llistat de fotos dels seus amics, i amics d'amics, i amics d'amics successivament. Seguint amb la seqüència d'instruccions anterior i la que segueix, cal obtenir el resultat que es mostra a continuació.

```
c=User("joanC")
f=Photo("d.jpg")
c.addPhoto(f)
f.addTag("holidays with maria and pere")
a.addFriend(c)
r=User("esteve")
s=Photo("e.jpg")
r.addPhoto(s)
s.addTag("holidays with maria and pere and joan")
s.addTag("pere always smiling")
c.addFriend(r)
print g.totSobreMi()
```

```
#Resultat d'execució
['pere Fantastic', 'wow pere', ['holidays with pere'], ['holidays with maria and pere'],
['holidays with maria and pere and joan', 'pere always smiling']]
```

EXERCICI 8.30 El Pere té una motxila que pot carregar n kilograms i vol fer un viatge amb la motxila tant plena com sigui possible. La motxila l'ha d'emplenar triant objectes el pes dels quals consta en una llista l. Dissenyeu una funció motxila(l,n) tal que donada la llista de pesos l torna la subllista dels elements d'l que cal posar a la motxila per omplir-la tant com es pugui sense sobrepassar la capacitat n. A tall d'exemple fixeu-vos en el següent exemple:

```
>>> motxila([10,1,5,4,4,8,6], 35)
[6, 8, 4, 5, 1, 10]
>>> motxila([10,1,5,4,4,8,6], 12)
[6, 5, 1]
>>> motxila([10,1,5,4,4,8,6], 0)
[]
>>> motxila([10,1,5,4,4,8,6], 23)
[6, 8, 4, 5]
```

EXERCICI 8.31 En un teclat de telèfon que segueixi l'estàndard Touch-ToneTM, els dígits es mapegen en l'alfabet (excepte les lletres Q i Z) tal i com mostra la Figura 1.

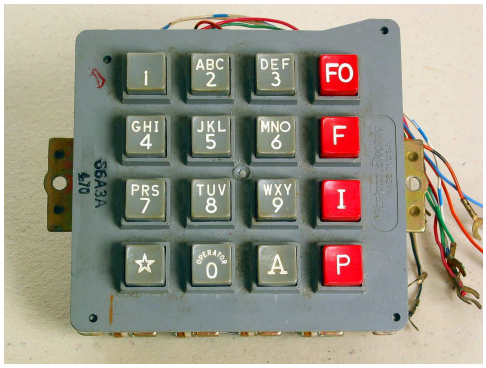


Figure 8.1: Touch-ToneTM(Font: Wikipedia)

Per tal de fer els números de telèfons més fàcils de memoritzar, als proveidors de servei telefònic els agrada trobar paraules (mnemònics) que facin que el número sigui fàcil de recordar. Per exemple, el número 6378687 pot ser recordat fàcilment per un dels seus mnemònics, NERVOUS.

Suposeu que heu estat llogats per una companyia telefònica local amb l'objectiu de dissenyar i implementar òptimament la funció **recursiva llistarMnemonics**, que generarà totes les possibles combinacions de lletres corresponents a un número donat, representat com a un string de dígits. Per exemple, la crida a **llistarMnemonics("723")** ha de generar les següents 27 combinacions possibles de lletres corresponents al prefix 723.

```
PAD PBD PCD RAD RBD RCD SAD SBD SCD
PAE PBE PCE RAE RBE RCE SAE SBE SCE
PAF PBF PCF RAF RBF RCF SAF SBF SCF
```

NOTA: No està permés l'ús d'slices/llesques ni funcions predefinides de Python. Les úniques operacions permeses són len, concatenació i accés a un element d'un string. La resolució del problema mitjançant una funció no recursiva tindrà una puntuació nul·la.

EXERCICI 8.32 Exploració recursiva. Una coneguda eina social vol implementar una anàlisi de la seva xarxa d'amistats. En aquesta xarxa s'ha arribat a la conclusió que cada persona en el món està connectada amb una altra per una distància promig de només 3 connexions i mitja! La distància és el número de connexions d'amics entre dos usuaris. La teva tasca consisteix en calcular la distància mínima entre dos usuaris. Assumeix que tot usuari està connectat amb la resta. Considera una xarxa simple amb les connexions que segueixen,

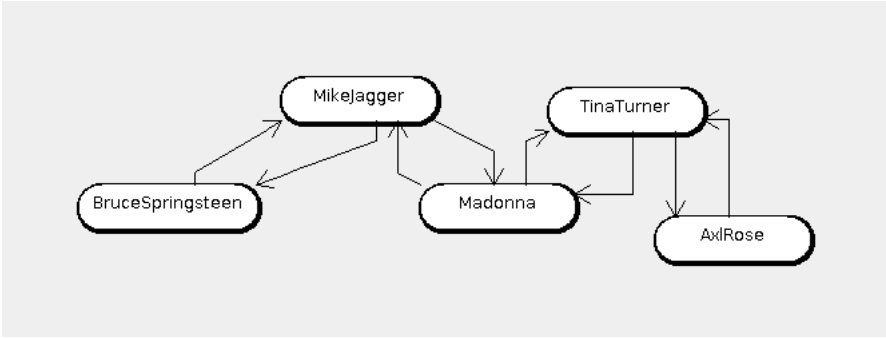


Figure 8.2: Xarxa de Connexions entre usuaris a la xarxa social

```
MikeJagger --> [Madonna,BruceSpringsteen]
BruceSpringsteen --> [MikeJagger]
Madonna --> [MikeJagger, TinaTurner, AxlRose]
TinaTurner --> [Madonna, AxlRose]
AxlRose --> [TinaTurner, Madonna]
```

En aquest cas,

- la distància mínima entre TinaTurner i AxlRose és 1 (Tenen connexió directa)
- la distància mínima entre TinaTurner i BruceSpringsteen és 3 (TinaTurner-Madonna-MikeJagger-BruceSpringsteen)
- la distància mínima entre MikeJagger i AxlRose és 2 (MikeJagger-Madonna-AxlRose)
- la distància mínima entre Madonna i Madonna és 0 (Madonna és Madonna)

Per tal de cercar la distància mínima entre un usuari inici i un usuari final cal utilitzar un tipus d'exploració recursiva anomenada aprofundiment iteratiu (iterative deepening). Inicialment es busca si es pot trobar un camí de l'inici al final amb distància com a molt 0. Si falla, es busca amb distància 1. Es continua recursivament buscant un camí augmentat successivament la distància. Finalment es troba un camí amb distància com a màxim n. La distància mínima entre l'inici i el final és n.

Per exemple, per trobar la distància mínima entre TinaTurner i MikeJagger:

- Hi ha un camí entre TinaTurner i MikeJagger amb distància 0? No
- Hi ha un camí entre TinaTurner i MikeJagger amb distància 1? No
- Hi ha un camí entre TinaTurner i MikeJagger amb distància 2? Sí

Doncs la distància mínima de TinaTurner a MikeJagger és 2. Se't demana la implementació de la funció recursiva **minDistance**, amb el funcionament esperat que es detalla a continuació.**NOTA:** Una solució no recursiva comportarà una puntuació nul·la. Pots utilitzar funcions auxiliars per a resoldre-ho.

```
def minDistance(usuaris,inici,final):
    """
    >>> minDistance({'MikeJagger':['Madonna','BruceSpringsteen'],'BruceSpringsteen':['MikeJagger'],
'Madonna':['MikeJagger', 'TinaTurner', 'AxlRose'],'TinaTurner':['Madonna', 'AxlRose'],'AxlRose':
['TinaTurner', 'Madonna']},'TinaTurner','AxlRose')
    1
    >>> minDistance({'MikeJagger':['Madonna','BruceSpringsteen'],'BruceSpringsteen':['MikeJagger'],
'Madonna':['MikeJagger', 'TinaTurner', 'AxlRose'],'TinaTurner':['Madonna', 'AxlRose'],'AxlRose':
['TinaTurner', 'Madonna']},'TinaTurner','BruceSpringsteen')
    3
    >>> minDistance({'MikeJagger':['Madonna','BruceSpringsteen'],'BruceSpringsteen':['MikeJagger'],
'Madonna':['MikeJagger', 'TinaTurner', 'AxlRose'],'TinaTurner':['Madonna', 'AxlRose'],'AxlRose':
['TinaTurner', 'Madonna']},'MikeJagger','AxlRose')
    2
    >>> minDistance({'MikeJagger':['Madonna','BruceSpringsteen'],'BruceSpringsteen':['MikeJagger'],
'Madonna':['MikeJagger', 'TinaTurner', 'AxlRose'],'TinaTurner':['Madonna', 'AxlRose'],'AxlRose':
['TinaTurner', 'Madonna']},'Madonna','Madonna')
    0
    >>> minDistance({'MikeJagger':['Madonna','BruceSpringsteen'],'BruceSpringsteen':['MikeJagger'],
'Madonna':['MikeJagger', 'TinaTurner', 'AxlRose'],'TinaTurner':['Madonna', 'AxlRose'],'AxlRose':
['TinaTurner', 'Madonna']},'AxlRose','BruceSpringsteen')
    3
    """
```

9 Cost en temps

EXERCICI 9.1 Quin és l'ordre assimptòtic de les següents funcions?

1. $f(n) = -2n^2 + 103n + 10$
2. $f(n) = \log n + n^3$
3. $f(n) = n + \sin n^2$
4. $f(n) = n(2 + \frac{3n}{4})$
5. $f(n) = \sqrt{23n} + 34$
6. $f(n) = \frac{34n^3 - 6n^2 + 45n - 9}{n^2 + 2}$

EXERCICI 9.2 Siguin $\mathcal{A}$ i $\mathcal{B}$ dos programes diferents per fer el mateix càlcul. Assumiu que el cost assimptòtic temporal en cas pitjor per a cada un d'ells és respectivament $O(n^2)$ i $O(n^3)$. Indiqueu i justifiqueu la correctesa de les següents afirmacions:

1. Per qualsevol mida n de les dades, $\mathcal{A}$ és més ràpid que $\mathcal{B}$.
2. A partir d'una certa mida de les dades n_0, per qualsevol configuració de les dades $\mathcal{A}$ és més ràpid que $\mathcal{B}$.
3. Per resoldre el problema de càlcul en qüestió, sempre preferirem l'algoritme $\mathcal{A}$ abans que el $\mathcal{B}$.

EXERCICI 9.3 Siguin A i B dues matrius quadrades de $n \times n$ amb elements reals. La matriu producte $C = AB$ es defineix com:

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

Assumint que representeu una matriu com una llista de llistes, dissenyeu una funció que calcula el producte de dues matrius quadrades. Quan la tingueu implementada, calculeu el cost assimptòtic en cas pitjor.

Exercici 9.4 Considereu el següent programa:

```
k = 1
for i in l[::2]:
    if i % 5 == 1:
        k += i
        k *= 2
    else:
        k = 0
```

Assumint que:

1. La llista l té longitud n.
2. Una assignació costa t_a segons.
3. Un pas d'iteració costa t_i segons.
4. Una comparació costa t_c segons.

Quina és la funció de cost en cas pitjor d'aquest algoritme? Quin és l'ordre assimptòtic d'aquesta funció?

Exercici 9.5 Considereu el següent algoritme sobre una llista l:

```
s = 0
for i in l[::2]:
    s += i
```

Quin és el cost assimptòtic en el cas pitjor respecte la longitud de la llista?

Exercici 9.6 Considereu el següent algoritme sobre una llista l:

```
s = 0
for i in l[::2]:
    for j in l:
        if i>j:
            s += i*j
        else:
            s += 2
```

Quin és el cost assimptòtic en el cas pitjor respecte la longitud de la llista?

Exercici 9.7 Considereu el següent algoritme sobre una llista l:

```
s = 0
for i in l[:len(l)/4]:
    if i % 2 == 0:
        for k in l:
            s += k/i
```

Quin és el cost assimptòtic en el cas pitjor respecte la longitud de la llista?

EXERCICI 9.8 Considereu el següent algoritme sobre una llista l:

```
s = 0
for i in l:
    for j in l[:i]:
        for k in l[i:j]:
            s += i
```

Quin és el cost assimptòtic en el cas pitjor respecte la longitud de la llista?

EXERCICI 9.9 Considereu la següent funció sobre una llista l:

```
def funcio_a(l):
    s = 0
    for i in l:
        funcio_b(s,l)
```

Quin és el cost assimptòtic en el cas pitjor de funcio_a respecte la longitud de la llista sabent que el cost de funcio_b és $O(n^2)$?

EXERCICI 9.10 Suposant que totes les operacions d'accés a una llista tenen complexitat assimptòtica en cas pitjor de $O(1)$, quina seria la complexitat assimptòtica d'un algoritme per calcular la intersecció de dues llistes l_1 i l_2? Noteu que calcular $l_1 \cap l_2$ vol dir calcular una tercera llista l_3 que conté els elements comuns de l_1 i l_2, i.e. $\forall x : x \in l_3 \iff x \in l_1 \wedge x \in l_2$.

EXERCICI 9.11 Donades les funcions que segueixen, **justifiqueu** per cadascuna si són d'ordre logarítmic $O(logn)$, d'ordre lineal $O(n)$, d'ordre quadràtic $O(n^2)$ o bé d'ordre cúbic $O(n^3)$.

Apartat a)

```
def comprovaa(n):
    x=0
    for i in range(n):
        x+=1
    for j in range(n):
        x+=2
    return x
```

Apartat b)

```
def comprovac(n):
    l=1
    m=0
    p=1
    for r in range(n):
        w=l*r+45
        v=m*m
    for i in range(n):
        for j in range(n):
            a=i*i
            b=j*j
            c=i*j
    z=33
```

Exercici 9.12 Donada una llista de n elements, respon les preguntes següents.

1. El cost assimptòtic en cas pitjor de l'algorisme d'ordenació **Merge Sort** és:
2. El cost assimptòtic en cas pitjor de l'algorisme d'ordenació **Bubble Sort** és:
3. El cost assimptòtic en cas pitjor de l'algorisme d'ordenació per **selecció** és:
4. Justifica amb un exemple el funcionament de l'algorisme d'**ordenació per selecció** i el seu cost.
5. Donada una **queue**, el cost del mètode *put* és ... i el cost del mètode *remove* és ...
6. Donada una **stack**, el cost del mètode *push* és ... i el cost del mètode *pop* és ...

Exercici 9.13 Justifica si són correctes o falses les afirmacions que segueixen. Una resposta sense justificació no serà avaluada.

1. La funció append de Python aplicada a una llista d'elements, té una complexitat en cas pitjor de $O(n)$.
2. L'operació '+' de Python aplicada sobre llistes, té una complexitat en cas pitjor de $O(n)$, sent n el nombre d'elements de la llista més gran a concatenar.
3. La cerca binària d'un element en una llista de n elements té una complexitat en cas pitjor de $O(n)$.
4. La cerca en un arbre binari de cerca de n nodes, d'un element que no es troba en l'arbre, tindria un cost de $O(n)$.

Exercici 9.14 Traça d'un programa i notació big-O. Donada la definició de funcions que segueix,

```
def mystery(n):
    sum = 0;
    for i in range(n,0,-1):
        sum += riddle(i)
    return sum
```

```
def riddle(n):
    sum = 0;
    for i in range(0,n):
        sum += i
    return sum
```

[Apartat a] Què retorna riddle(4)?

[Apartat b] Què retorna mystery(4)?

[Apartat c] Quina és la complexitat en cas pitjor de la funció riddle expressada en termes de notació big-O, sent N el valor de l'argument n?

[Apartat d] Quina és la complexitat en cas pitjor de la funció mistery expressada en termes de notació big-O, sent N el valor de l'argument n?

Exercici 9.15 Donada la llista següent desordenada,

```
[54, 26, 93, 17, 77, 31, 44, 55, 20]
```

Apartat a) Detalla com s'ordenaria descendentment pas per pas aplicant el mecanisme d'ordenació *selection sort*. Quina és la complexitat en cas pitjor del selection sort?

Apartat b) Detalla com s'ordenaria descendentment pas per pas aplicant el mecanisme d'ordenació *bubble sort*. Quina és la complexitat en cas pitjor del bubble sort?

EXERCICI 9.16 Donada les següents funcions recursives, a) Justifiqueu el resultat de cadascun dels seus doctestos b) Pel primer doctest, justifiqueu quantes crides recursives s'efectuaran c) Digueu quin és el cost assimptòtic en cas pitjor de dita funció.

```
def uala(a,i):
    """
    >>> uala([0,1,2,8,13,17,19,32,42],3)
    #doctest1
    >>> uala([-1,1,8,13,34],1)
    #doctest2
    """
    if len(a)==0:
        return False
    else:
        m=len(a)/2
        if a[m]==i:
            return True
        else:
            if i<a[m]:
                return uala(a[:m],i)
            else:
                return uala(a[m+1:],i)

def carai(n):
    """
    >>> carai(1)
    #doctest1
    >>> carai(2)
    #doctest2
    >>> carai(3)
    #doctest3
    """
    if n==1:
        return 1
    else:
        return carai(n-1)+n*n
```

EXERCICI 9.17 Justifica si són correctes o falses les afirmacions que segueixen. Una resposta sense justificació no serà avaluada.

1. La funció append de Python aplicada a una llista d'elements, té una complexitat en cas pitjor de $O(n)$.

2. L'operació '+' de Python aplicada sobre llistes, té una complexitat en cas pitjor de $O(n)$, sent n el nombre d'elements de la llista més gran a concatenar.

3. La cerca binària d'un element en una llista de n elements té una complexitat en cas pitjor de $O(n)$.

4. La cerca en un arbre binari de cerca de n nodes, d'un element que no es troba en l'arbre, tindria un cost de $O(n)$.

EXERCICI 9.18 Donada la llista següent desordenada,

```
[54, 26, 93, 17, 77, 31, 44, 55, 20]
```

Apartat a) Detalla com s'ordenaria descendentment pas per pas aplicant el mecanisme d'ordenació *selection sort*. Quina és la complexitat en cas pitjor del selection sort?

Apartat b) Detalla com s'ordenaria descendentment pas per pas aplicant el mecanisme d'ordenació *bubble sort*. Quina és la complexitat en cas pitjor del bubble sort?

EXERCICI 9.19 Per cadascun dels enunciats següents, escriu si és [Cert/Fals]. En cas que sigui **Fals, justifica la resposta**. (La no justificació invalida la resposta)

1. Els algorismes selection sort, bubble sort i heap sort tenen el mateix cost assimptòtic en cas pitjor.

2. La cerca binària requereix cost linial en cas pitjor.

3. Una pila és una estructura FIFO, i una cua és una estructura LIFO.

4. Sempre hi haurà un cost en temps $O(logn)$ per cercar un node arbitrari en un arbre de cerca binari de tamany n.

5. Una subclasse pot accedir a mètodes privats de la superclasse.

6. Un mètode que triga $O(logn)$ calcularà un resultat més ràpid que un mètode que triga $O(n)$, assumint que n té el mateix valor per cada mètode.

7. Merge sort és un algorisme amb una complexitat en cas pitjor de $O(nlogn)$.

8. Suposem que B és una subclasse de A i C és una subclasse de B. Llavors, tot mètode d'un objecte de classe C pot executar el codi del constructor de A.

9. El block finally en un try..except..else..finally s'executa inclús s'hi no s'ha llençat una excepció i hi ha una instrucció return en el block try..except.

10. Si una classe A té dos mètodes amb signatures $m(self, m = "")$ i $m(self, v = 0)$, diem que el mètode m està redefinit.

EXERCICI 9.20 Per cadascun dels enunciats següents, justifica si és [Cert/Fals]. (La no justificació invalida la resposta)

1. Si b i c són objectes de la classe A, i la classe A no ha realitzat una sobreescriptura/redefinició del mètode equals, llavors b==c evaluarà a True si i només si els valors de cada atribut de b i c són iguals.

2. Suposem les classes A i B, on s'ha definit la classe B com a public class B(A). Suposem que var és un atribut públic de la classe A. Suposem que la variable x és de tipus B. Llavors l'expressió x.var és correcta.

3. Cada mètode recursiu sempre necessita tenir almenys un paràmetre enter que es fa més petit cada vegada que es va cridant recursivament, i un cas base que el chequegi pels valors 1 i 0.

4. Donat un objecte del tipus de dades Stack, podem accedir als seus n elements, utilitzant el mètode top() n vegades.

5. Donada una llista ordenada, la cerca binària requereix cost logarítmic en cas pitjor.

6. Si una classe A té dos mètodes amb signatures $m(self, t = "")$ i $m(self, v = 0)$, diem que el mètode m està sobrecarregat.

EXERCICI 9.21 Classes i mètodes. Donat el següent fragment de codi, i per cadascuna de les afirmacions que segueixen, escriu si l'afirmació és [Certa/Falsa]. **(La no justificació invalida la resposta)**

```
class Python(object):
    x = -1
    y = 0
    def __init__(self, y):
        self.x = y
        self.__z=99
    def calcula(self, y):
        self.x = min(self.x, y)
        return self.x
    def quadra(self):
        self.x = max(self.x, self.y)
        return self.x
    def inventa(self,a):
        return self.x+a
class Java(Python):
    def inventa(self):
        self.y = self.x * self.x
        return self.y
    def quadra(self):
        self.x=0
class Basic(Python):
    def inventa(self):
        self.y=22
        return self.y
class JScript(Java):
    pass
```

```
if __name__=='__main__':
    h1=Basic(2)
    g1=Java(2)
    v=[]
    v.append(h1)
    v.append(g1)
    for a in v:
        print a.inventa()
```

1. El mètode *inventa* de la classe *Java* és un exemple de sobrecàrrega de mètodes.

2. El mètode *init* de la classe *Basic* és un exemple de redefinició de mètodes.

3. Un objecte de la classe *JScript* pot accedir a l'atribut x de la classe *Python*.
4. El resultat de la execució del programa serà 0 0.
5. No es poden crear objectes de la classe *JScript* perquè no hi ha definit el mètode constructor.
6. El mètode min utilitzat en el mètode calcula de la classe Python té una complexitat $O(n)$.
7. Un objecte de la classe *Basic* pot accedir al mètode *inventa* de la classe *Python*.
8. La crida $i.calcula(11)$, sent i un objecte de classe *JavaScript*, generaria un error d'execució.
9. Les crides següents generaran errors d'execució

```
j=Java(1); print j.inventa(22)
```

10. El millor algorisme d'ordenació d'una llista de n objectes és el mètode merge sort perquè té complexitat en cas pitjor $O(logn)$.

EXERCICI 9.22 Donada la llista següent desordenada,

```
[54, 26, 93, 17, 77, 31, 44, 55, 20]
```

Apartat a) Detalla com s'ordenaria ascendentment pas per pas aplicant el mecanisme d'ordenació *bubble sort*. Quina és la complexitat en cas pitjor del bubble sort?

Apartat b) Detalla com s'ordenaria ascendentment pas per pas aplicant el mecanisme d'ordenació *merge sort*. Quina és la complexitat en cas pitjor del merge sort?

EXERCICI 9.23 Dissenyeu la funció booleana **recursiva** de nom *formatOK(input, labels)*, tal que, donat un input determina que aquest input està ben formatat respecte a una llista d'etiquetes proporcionada (labels), si i només si: 1) l'input és una llista, 2) l'input té com a mínim una longitud de 2, 3) el primer ítem de l'input es troba a la llista d'etiquetes i 4) cadascun dels ítems que queden a la llista és un string o bé una llista ben formatada.

```
def formatOK(input, labels):
    >>> formatOK(['VP', ['V', 'eat']], ['VP', 'V'])
    True
    >>> formatOK(['NP', ['N', 'a', 'or', 'b'], 'c'], ['NP', 'V', 'N'])
    True
    >>> formatOK([1, [2, 'oui', [1, 'no']], 'no'], [1, 2])
    True
    >>> formatOK(['VP', ['V', 'eat']], ['VP'])
    False
    >>> formatOK(['VP', ['V']], ['VP', 'V'])
    False
    >>> formatOK('VP', ['VP', 'V'])
    False
    """
```

EXERCICI 9.24 Donada una llista de mida n, justifica quina serà el complexitat teòrica en cas pitjor, si ordenem la llista utilitzant les estratègies (La no justificació invalida la resposta):

1. selection sort
2. merge sort
3. bubble sort

EXERCICI 9.25 L'algoritme recursiu d'ordenació de llistes anomenat *mergesort* opera de la següent forma:

1. (base) La llista té un sol element, per tant ja està ordenada.
2. (recurrència) Es divideix la llista en dues subllistes. S'ordenen les subllistes recursivament. Es torna la fussió de les dues subllistes.

Es demana que:

1. Escriviu una funció tal que, donades dues llistes de naturals ordenades de forma creixent, retorna una llista de naturals ordenada de forma creixent resultat d'unir les dues llistes anteriors. La funció ha de tenir un cost assimptòtic en cas pitjor de $O(n)$ essent n la mida de la llista fusionada.
2. Escriviu l'algoritme recursiu *mergesort* usant la funció anterior.
3. Calculeu l'equació recurrent de cost en cas pitjor $T(n)$ de l'algoritme *mergesort*.
4. Comproveu que $f(n) = n \log_2 n$ és una solució de l'equació anterior.

EXERCICI 9.26 Aplicant la definició d'ordre d'una funció, demostreu que si $f(x) = 3 \sin x$, llavors $f \in O(n^2)$.

EXERCICI 9.27 Considereu la pràctica del simulador de circuits digitals.

1. Definiu una bona manera de mesurar la grandària d'un circuit.
2. Donada al mesura definida anteriorment, quin és el cost assimptòtic en cas pitjor del simulador?

10 Estructures de dades lineals

EXERCICI 10.1 Donada una pila p amb **len**(p)> 0 dissenyeu una funció que buida la pila.

EXERCICI 10.2 Per convertir un natural expressat en base 10 a un natural expressat en base 2, una forma habitual consisteix en anar dividint per dos i després fixar-se en els residus i el darrer quocient que han quedat. Per exemple, en el cas de $n = 35$:

$$\begin{aligned} 35/2 &\rightarrow 17 \text{ i sobren } 1 \\ 17/2 &\rightarrow 8 \text{ i sobren } 1 \\ 8/2 &\rightarrow 4 \text{ i sobren } 0 \\ 4/2 &\rightarrow 2 \text{ i sobren } 0 \\ 2/2 &\rightarrow 1 \text{ i sobren } 0 \end{aligned}$$

Aleshores, tenim que $35_{(10} = 100011_{(2}$. Fixeu-vos que:

1. el primer dígit binari és el darrer quocient i la resta de dígits són els residus.

2. que els residus s'han d'escriure en ordre invers a com s'obtenen.

Escriviu, usant una pila, una funció tal que, donat un natural, escrigui la seva representació binària.

EXERCICI 10.3 Sigui p una pila d'enters. Dissenyeu una funció que calculi la suma dels seus elements usant únicament les operacions públiques. Ajudeu-vos d'una pila auxiliar.

EXERCICI 10.4 Una expressió ben parentitzada és una cadena en la que els parèntesis, claus i claudàtors es tancan i obren de forma correcta. Per exemple, `'(2+sin(3))*4+k[1]'` és una cadena ben parentitzada, en canvi `'(k[4]+(3+(12))'` no. Es demana que, fent ús d'una pila escriviu una funció tal que, donada una cadena, retorni cert ssi està ben parentitzada.

EXERCICI 10.5 Implementeu la classe Cua usant dos piles com a representació.

Exercici 10.6 Suposant la implementació de l'estructura de dades lineal pila implementada en Python d'acord amb les indicacions treballades a classe, **justifiqueu** quin serà el resultat d'executar les següents instruccions.

```
from stack import Stack
s=Stack()
s.push(22)
s.push(34)
s.push(22)
s.push(3)
while not s.isEmpty():
    z=s.top()
    if z==22:
        s.push(1)
    elif z==1:
        print "Trobat 1"
        s.pop()
        s.pop()
    else:
        print z
        s.pop()
```

Exercici 10.7 L'expressió correcta.

Suposem que cal gestionar la correcta parentització d'una expressió en un sistema d'enviament d'informació. Hi ha tres tipus d'elements que considerem parèntesi: (), [] i {}. Suposeu que no s'envia res més que no siguin aquests parèntesi d'obertura/tancament i que l'ordre de parèntesi és important. Per exemple, {()} està ben parentitzada, però {(}) no.

1) Escriviu una funció tal que donada una expressió d'aquest tipus, retorni **True** o bé **False** si aquesta està ben parentitzada. Comproveu els doctests que figuren a continuació.

Pista: Podeu utilitzar la classe Stack directament i els seus mètodes associats.

2) Justifiqueu quina complexitat en cas pitjor té la funció que heu dissenyat.

```
def benParentitzada(e):
    """
    >>> benParentitzada('([])')
    True
    >>> benParentitzada('(((((((([[[{{}}]]])))))))')
    True
    >>> benParentitzada('{[(])')
    False
    """
```

Exercici 10.8 Suposant la implementació de l'estructura de dades lineal pila (Stack), i les seves operacions push(), top(), pop(), len.

Apartat a) Se us demana que implementeu únicament el mètode *reverse(s)*, tals que, donada una pila *s*, permeti girar l'ordre dels elements de *s*. Suposeu que aquest nou mètode s'afegirà a la classe Stack(). Podeu utilitzar una pila addicional, i els mètodes de la classe Stack. Òbviament,

l'ús de funcions predefinides de Python comportarà una puntuació nul·la. Comproveu-ne a continuació el seu ús.

```
if __name__=='__main__':
    s=Stack()
    for i in range(10):
        s.push(i)
    t=s.reverse()
    print t # Escriura 9 8 7 6 5 4 3 2 1 0
    s=t.reverse()
    print s # Escriura 0 1 2 3 4 5 6 7 8 9
```

Apartat b) Ara passeu a recursiu aquest mètode i anomeneu-lo *reverseRec(s)*.

EXERCICI 10.9 Donada una expressió correcta, se us demana que comproveu si conté parèntesis duplicats o no. Un conjunt de parèntesis són duplicats si la mateixa subexpressió està rodejada de múltiples parèntesis. A continuació es mostren expressions amb parèntesis duplicats.

```
#Exemple 1
((a+b)+((c+d))) #La subexpressió ‘‘c+d’’ està envoltada de 2 parells de parèntesis.

#Exemple 2
(((a+(b)))+(c+d)) #La subexpressió ‘‘a+(b)’’ està envoltada per dos parells de parèntesis.

#Exemple 3
(((a+(b))+c+d)) #L'expressió completa està envoltada per dos parells de parèntesis.

Les expressions que segueixen a continuació no contenen parèntesis duplicats.

#Exemple 4
((a+b)+(c+d)) #Cap subexpressió està rodejada de parèntesis duplicats.

#Exemple 5
((a+(b))+(c+d)) #Cap subexpressió està rodejada de parèntesis duplicats.
```

Se us demana que implementeu la funció booleana *findDuplicate(expression)*, tal que, donada una expressió correcta retorni True si conté parèntesis duplicats.

Nota: podeu utilitzar la implementació de l'estructura de dades lineal pila (Stack), i les seves operacions push(), top(), pop(), len (no cal que lliureu la implementació de la classe Stack, podeu utilitzar-la directament).

EXERCICI 10.10 **[Apartat a]**Digues quin és el resultat d'execució del programa que segueix i quina és la finalitat del mètode npi.

```
class Stack:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        return self.items.pop()

    def peek(self):
        return self.items[len(self.items)-1]

    def size(self):
        return len(self.items)
```

```
class Mistery:
    def __init__(self, capacity):
        self.capacity = capacity
        self.s = Stack()

    def npi(self, exp):
        for i in exp:
            if i.isdigit():
                self.s.push(i)
            else:
                val1 = self.s.pop()
                val2 = self.s.pop()
                self.s.push(str(eval(val2 + i + val1)))
        return int(self.s.pop())

w = "231*+9-"
obj = Mistery(len(w))
print "Result: %d"%(obj.npi(w))
```

[**Apartat b**] Quina és la complexitat en cas pitjor de la funció **npi**? Utilitza la notació O, i justifica breument el seu cost.

EXERCICI 10.11 L'empresa que gestiona el navegador ChroItic està buscant programadors que els ajudin a implementar una nova característica en la seva nova versió de navegador, que permeti tracejar els tags de pàgines .html potencials per comprovar que no hi haurà errades en la visualització de les pàgines.

Una pàgina .html està formada per un seguit de tags d'obertura i tancament, com els que segueixen. El principi i final d'un document es marquen amb els tags $\langle html \rangle$ i $\langle /html \rangle$. Fixeu-vos que els tags d'obertura tenen el format $\langle nomtag \rangle$ i els de tancament tenen el format $\langle /nomtag \rangle$. Adoneu-vos que es poden obrir diversos tags a la vegada, però que els tags de tancament han de seguir un ordre.

```
<html> <title>Prova </title> <b> <i>TecproExam </i> </b> </html>
```

En aquest cas, es crea una pàgina hmtl amb un títol i el contingut TecproExam en cursiva i negreta. En el següent exemple s'ha omès el títol de la pàgina, però la seqüència d'obertura i tancament de tags és correcta.

```
<html> <b> <i>TecproExam </i> </b> </html>
```

Exemples de pàgines incorrectes, serien per exemple,

```
<title>Prova <html> </title> <b> <i>TecproExam </i> </b> </html>
#Errònia: el tancament del tag title està en ordre incorrecte

<html>Prova <b>TecproExam </b> #Errònia: no s'ha tancat el tag html
```

Per simplificar, aplicarem els següents supòsits:

1. El separador entre tags serà l'espai en blanc
2. Les paraules entre els tags no contenen espais en blanc
3. Els tags estan ben construïts. És a dir no tindrem tags amb el format $<< nomTag >$ per exemple.

[Apartat a] Dissenyeu òptimament la funció booleana **tagsOK**, tal que donada una seqüència de strings, retorni True si el seu conjunt de tags conforma correctament una pàgina web.

NOTA: Podeu utilitzar directament la classe Stack, amb els seus mètodes associats.

```
def tagsOK(s):
    """
    >>> tagsOK('<html> <title>Prova </title> <b> <i>TecproExam </i> </b> </html>')
    True
    >>> tagsOK('<html> <title>Prova </title> <b> <i>TecproExam </b> </i> </html>')
    False
```

```
>>> tagsOK('<html> <title>Prova <b> <i>TecproExam </b> </i> </html>')
False
>>> tagsOK('<html> <title>Prova </title> <b> <i>TecproExam </i> </b> ')
False
"""
```

[Apartat b] Quina és la complexitat en cas pitjor de la funció **tagsOK**? Utilitza la notació O, i justifica breument el seu cost.

11 Arbres

Exercici 11.1 Donades les següents expressions, dibuixeu el seu arbre sintàctic i escriviu les expressions en notació postfixa i prefixa.

1. $3 \times (2 + 4 + 6)$
2. $(3 \times 2) + (4 \times 5)$
3. $2 \times (2 \times (2 \times x) + 4) + 6$

Exercici 11.2 Dissenyeu una funció tal que, donat un node d'un arbre, retorni la llista dels néts del node.

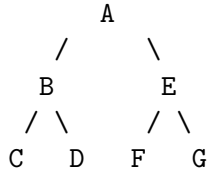
Exercici 11.3 Esteneu el conjunt d'operacions bàsic dels arbres vist a teoria amb una operació per accedir al node pare. En el cas de l'arrel, aquesta operació ha de tornar **None**. Esteneu ara la implementació que teniu a les transparències per tal d'encabir-hi aquesta nova operació.

Exercici 11.4 El *Least common ancestor*, LCA, de dos nodes a i b d'un arbre és el primer node ascendent que tenen en comú a i b. Dissenyeu una funció tal que, donats dos nodes d'un arbre, calcula el seu LCA. Podeu considerar que l'arbre té operació d'accés al pare.

Exercici 11.5 Donada la definició de la classe arbre binari treballada a classe de teoria, es demana:

Apartat a) implementeu un nou mètode recursiu, *recorregutInordre*, que escrigui per pantalla els nodes de l'arbre en inordre. Un recorregut en inordre d'un arbre consisteix en, per cada node, realitzar un recorregut en inordre de l'arbre esquerre, visitar el node, i després realitzar un recorregut en inordre de l'arbre dret.

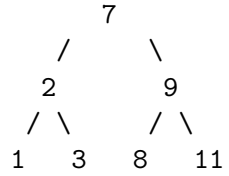
Per exemple, donat l'arbre,



el resultat del recorregut en inordre hauria de ser CBDAFEG

Apartat b) implementeu un nou mètode recursiu, *comptaNodesParells*, tal que, donat un arbre binari de nombres, compti quants dels nodes són parells.

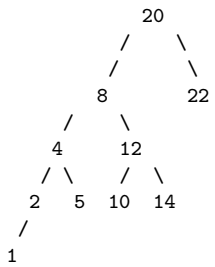
Per exemple, donat l'arbre,



El mètode hauria de retornar el valor 2.

EXERCICI 11.6 Donada la classe arbreBST, corresponent a un Binary Search Tree, amb els mètodes implementats a classe de teoria,

Apartat a) Se us demana que implementeu el mètode recursiu *minValue*, tal que, donat un arbre BST no balancejat, retorni el número més petit emmagatzemat a l'arbre. Per exemple, en l'arbre següent, la resposta hauria de ser 1.



Apartat b) Escriu el mètode recursiu *printArbrePreordre*, tal que, donat un arbre com l'anterior escrigui els nodes seguint un recorregut de l'arbre en preordre.

Per l'arbre de la figura, el seu recorregut en preordre hauria de ser:

20 8 4 2 1 5 12 10 14 22

Apartat c) Escriu quina és la complexitat en cas pitjor del mètode *minValue* i quina és la complexitat en cas pitjor del mètode *printArbrePreordre*. Utilitza la notació O, i justifica breument el seu cost.

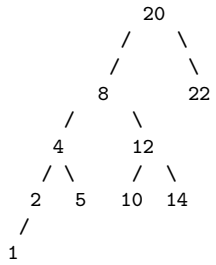
EXERCICI 11.7 Donada la classe arbreBST, corresponent a un Binary Search Tree, amb els mètodes implementats com segueixen,

```
class arbreBST(object):
    def __init__(self):
        self.v=None
        self.left=None
        self.right=None

    def insereix(self,k):
        if self.v==None:
            self.v=k
        else:
            if self.v==k:
                raise Exception("Repetit")
            elif self.v<k:
                if self.right is None:
                    self.right=arbreBST()
                self.right.insereix(k)
            else:
                if self.left is None:
                    self.left=arbreBST()
                self.left.insereix(k)
    def maxValue(self):
        #TO DO
    def printArbrePostordre(self):
        #TO DO
```

```
if __name__=='__main__':
    arbre=arbreBST()
    arbre.insereix(20)
    arbre.insereix(8)
    arbre.insereix(12)
    arbre.insereix(4)
    arbre.insereix(10)
    arbre.insereix(14)
    arbre.insereix(22)
    arbre.insereix(2)
    arbre.insereix(5)
    arbre.insereix(1)
    print arbre.maxValue()
    arbre.printArbrePostordre()
```

Apartat a) Se us demana que implementeu el mètode recursiu *maxValue*, tal que, donat un arbre BST no balancejat, retorni el número més gran emmagatzemat a l'arbre. Per exemple, en l'arbre següent, la resposta hauria de ser 22.



Apartat b) Escriu el mètode recursiu *printArbrePostOrdre*, tal que, donat un arbre com l'anterior escrigui els nodes seguint un recorregut de l'arbre en postordre.

Per l'arbre de la figura, el seu recorregut en postOrdre hauria de ser:

1 2 5 4 10 14 12 8 22 20

Apartat c) Escriu quina és la complexitat en cas pitjor del mètode *maxValue* i quina és la complexitat en cas pitjor del mètode *printArbrePostordre*. Utilitza la notació O, i justifica breument el seu cost.

EXERCICI 11.8 Un arbre binari es pot representar (emmagatzemar) en una llista plana seguint la següent estratègia:

1. El node arrel s'emmagatzema a la posició 0 de la llista.

2. Donat un node que ocupa la posició i de la llista, el seu fill esquerre ocupa la posició $2i+1$ i el seu fill dret la posició $2i+2$.

Seguint aquesta estratègia, implementeu una classe ArbreBin que usa aquesta representació.

EXERCICI 11.9 Considereu el conjunt d'enters $C = \{23, 4, 2, 12, 7, 45, 5, 21, 63, 82, 34\}$ i dibuixeu:

1. El pitjor arbre binari de cerca pel que fa al cost en temps que conté els elements de C.

2. El millor arbre binari de cerca pel que fa al cost en temps que conté els elements de C.

Exercici 11.10 Afegiu a la definició d'arbre binari de cerca que segueix, els mètodes recursius que se us detallen a continuació,

```
class arbreBinari(object):
    def __init__(self):
        self.v=None
        self.left=None
        self.right=None

    def maxim(self):
        """
        Mètode recursiu que retorna el valor més gran de l'arbre
        """
        #TO DO

    def interval(self,inici,final):
        """
        Mètode recursiu que mostra els elements en preordre, que es trobin en l'interval (inici,final)
        """
        #TO DO
```

Exercici 11.11 Qüestions generals.

Apartat a) Defineix breument els conceptes que segueixen.

1. Classe abstracta.
2. Mètode constructor.
3. Redefinició de mètodes.
4. Sobrecàrrega de mètodes.

Apartat b) Respon clarament les següents qüestions.

1. Diferències entre agregació i composició.
2. Diferències entre atributs públics i atributs privats.

Apartat c). Analitza la classe següent i respon les preguntes.

```
class arbreBinari(object):
    def __init__(self,v,left=None,right=None):
        self.v=v
        self.left=left
        self.right=right

    def misteri(self):
        if self is None: return 0
        else:
            if self.left is None and self.right is None:
```

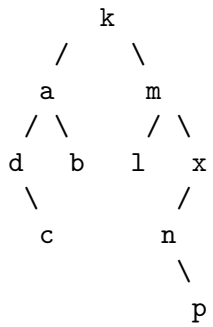
```
                return self.v
            else:
                return self.left.misteri()+self.right.misteri()+self.v

    def nose(self, t):
        if t == self.v:
            return True
        elif t < self.v:
            if self.left is not None:
                node = self.left.nose(t)
        else:
            if self.right is not None:
                node = self.right.nose(t)
        return True
```

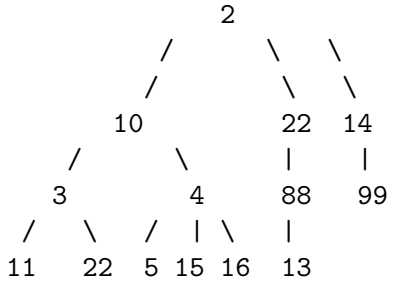
1. L'objectiu del mètode misteri és
2. Donat un arbre binari de n elements, el cost assimptòtic en cas pitjor del mètode misteri és
3. L'objectiu del mètode nose és
4. Donat un arbre binari de n elements, el cost assimptòtic en cas pitjor del mètode nose és

Exercici 11.12 Donat l'arbre binari que segueix, escriu-ne

1. el recorregut en pre-ordre,
2. el recorregut en post-ordre i
3. el recorregut en in-ordre.



Exercici 11.13 Donada la classe *Arbre*, amb els mètodes implementats a classe de teoria, en què cada node pot tenir més de 2 fills, se us demana que implementeu el mètode **recursiu** *quantesFullesS*, tal que, donat un arbre, retorni el nombre de fulles amb valors senars que hi ha a l'arbre. Per exemple, en l'arbre següent, la resposta hauria de ser 5.



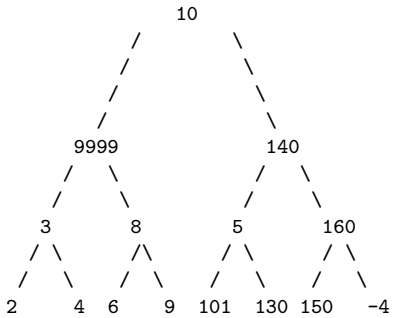
EXERCICI 11.14 La universitat de Júpiter assigna un identificador per a cada estudiant, consistent en un enter aleatori únic. Cada identificador s'emmagatzema en un arbre binari de cerca (BST), segons la definició que segueix.

```
class BST(object):
    def __init__(self):
        self.v=None
        self.left=None
        self.right=None
```

Quan s'admet un estudiant, la universitat tria un enter al·leatori, entre el valor 0 i el valor 9000, i comprova si ja es troba en el BST. Si no hi és, s'assigna l'identificador a l'estudiant. Si hi és, es repeteix el procés amb un nou número al·leatori.

Malauradament, el sindicat de hackers frikis ha modificat el BST i n'ha canviat un o més valors. Tement el pitjor, la universitat t'ha llogat com a estudiant TIC per tal de comprovar el BST. Com a primer pas, decideixes verificar que el BST està correctament estructurat. A tal efecte cal que dissenyeu i implementeu únicament el mètode **recursiu** *desubicat*, tal que ha de comprovar que el BST emmagatzema valors correctes i que compleixen la propietat dels BST. En cas que no ho sigui, ha de retornar el primer valor erroni trobat.

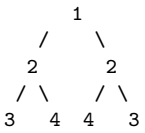
Per exemple, en el BST següent hauria de retornar el valor 9999 (el 5 i el -4 també estan desubicats).



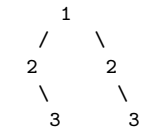
EXERCICI 11.15 Forest Gump t'ha llogat per chequejar si els seus arbres binaris són mirall/simètrics. 2 arbres/subarbres són mirall/simètric,

- si tenem el mateix valor per l'arrel, i
- el subarbre esquerre de l'arbre esquerre i el subarbre dret de l'arbre dret són mirall, i
- el subarbre dret de l'arbre esquerre i el subarbre esquerre de l'arbre dret són mirall.

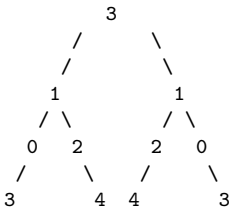
El següent arbre BT és simètric



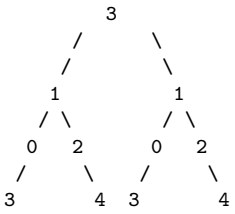
El següent arbre BT no és simètric



El següent arbre BT és simètric



El següent arbre BT no és simètric



La classe de partida i el joc de proves proporcionats són els que segueixen. Chequegeu-los amb cura per entendre el seu funcionament.

```
class BT(object):
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

def isMirror(tree):
    #TO DO
```

```
if __name__=='__main__':
    root = BT(1)
    root.left = BT(2)
    root.right = BT(2)
    root.left.left = BT(3)
    root.left.right = BT(4)
    root.right.left = BT(4)
    root.right.right = BT(3)
    print "1" if isMirror(root) == True else "0" #escriurà 1
    root = BT(1)
    root.left = BT(2)
    root.right = BT(2)
    root.left.right = BT(3)
    root.right.right = BT(3)
    print "1" if isMirror(root) == True else "0" #escriurà 0
```

[Apartat a] A tal efecte cal que dissenyeu i implementeu únicament la funció **recursiva** *isMirror*, tal que ha de comprovar si el BT és simètric/mirall o no ho és.

[Apartat b] Trieu i **justifiqueu** quina és la complexitat d'aquesta funció respecte al número de nodes de l'arbre (n).

1. Linial $O(n)$
2. Logarítmica $O(logn)$ o bé $O(nlogn)$
3. Quadràtica $O(n^2)$
4. Exponencial $O(2^n)$

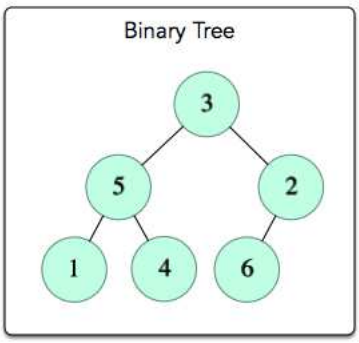
Exercici 11.16 Donat un arbre binari, se us demana realitzar la funció recursiva $checkBinarySearchTree$, que comprovi si es tracta d'un arbre binari de cerca.

```
class BT(object):
  def __init__(self, data):
```

```
        self.data = data
        self.left = None
        self.right = None

def check_binary_search_tree_(tree):
#TO DO
```

Per exemple, donat el següent arbre binari de cerca,



la funció hauria de retornar False

Exercici 11.17 BST. [**Apartat a**] Donat un arbre binari de cerca que emmagatzema valors enters, i dos dos valors enters k1 i k2 (on $k1 < k2$), se us demana que implementeu òptimament el mètode recursiu **mostraRang**, tal que ha de mostrar per pantalla les claus del l'arbre entre el rang k1 i k2, és a dir, mostra tots els valors de l'arbre entre k1 i k2 (ambdòs valors inclosos). Cal mostrar la informació en ordre creixent. Per exemple, donat l'arbre de la Figura 2, si k1=10 i k2=22, el mètode ha d'escriure 12, 20, 22.

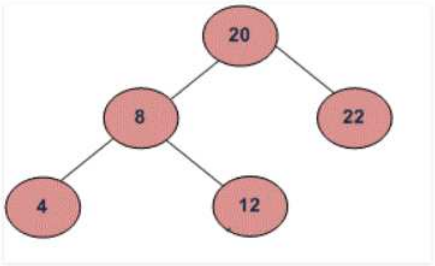


Figure 11.1: A Binary Search Tree

Suposeu la següent definició d'arbre binari de cerca i els exemples de funcionament que segueixen.

```
class BST(object):
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
def mostraRang(root, k1, k2):
        #TO DO
```

```
if __name__==’__main__’:
  k1 = 10 ; k2 = 25 ;
  arbre = BST(20)
  arbre.left = BST(8)
  arbre.right = BST(22)
  arbre.left.left = BST(4)
  arbre.left.right = BST(12)
  mostraRang(arbre, k1, k2)
```

[**Apartat b**] Quina és la complexitat en cas pitjor del mètode **mostraRang** implementat? Utilitza la notació O, i justifica breument el seu cost.

EXERCICI 11.18 Donada la classe arbreBST, corresponent a un Binary Search Tree i balancejat, amb els mètodes implementats com segueixen,

```
class arbreBST(object):
    def __init__(self):
        self.v=None
        self.left=None
        self.right=None

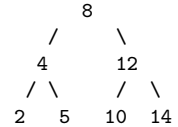
    def insereix(self,k):
        if self.v==None:
            self.v=k
        else:
            if self.v==k:
                raise Exception("Repetit")
            elif self.v<k:
                if self.right is None:
                    self.right=arbreBST()
                self.right.insereix(k)
            else:
                if self.left is None:
                    self.left=arbreBST()
                self.left.insereix(k)
    def minValue(self):
        #TO DO
    def printCommon(self,other):
        #TO DO
```

Apartat a) Se us demana que implementeu el mètode recursiu *nodesKfulles*, tal que, donat un arbre BST, escrigui tots els nodes que tenen k fulles. Per exemple, donat l'arbre que segueix, si la k=1, hauria de retornar el node 2. Si k=2, hauria de retornar els nodes 20, 8, 4 i 12

Apartat a.1) Sent n el nombre d'elements guardats a l'arbre, tria i justifica quina és l'ordre de complexitat en cas pitjor del mètode *minValue*.

1. $O(n)$
2. $O(logn)$
3. $O(n^2)$

Apartat b) Escriu el mètode recursiu *printCommon*, tal que, donat 2 arbres BST no balancejats, escrigui els nodes que tenen en comú els 2 arbres.



Pels arbres de la figura, el mètode hauria d'escriure per pantalla,

Apartat b.1) Sent n el nombre d'elements guardats a l'arbre a i m el nombre d'elements guardats a l'abre b, tria i justifica quina és l'ordre de complexitat en cas pitjor del mètode *printCommon(a,b)*.

1. $O(n * m)$
2. $O(n + m)$
3. $O((n)^2)$

Exercici 11.19 BST. Se't proporciona la següent implementació de BST i els exemples de funcionament que segueixen.

```
class BST(object):
    def __init__(self):
        self.v=None
        self.left=None
        self.right=None

    def insereix(self,k):
        if self.v==None:
            self.v=k
        else:
            if self.v==k:
                raise Exception("Repetit")
            elif self.v<k:
                if self.right is None:
                    self.right=arbreBinari()
                self.right.insereix(k)
            else:
                if self.left is None:
                    self.left=arbreBinari()
                self.left.insereix(k)

    def __str__(self):
        return str(self.v)

    def printArbrePreordre(self):
        if self is None: return
        if self.v is None: return
        print self,
        if self.left is None: pass
        else:
            self.left.printArbrePreordre()
        if self.right is None: pass
        else:
            self.right.printArbrePreordre()

    def es_fulla(self):
        return self.right is None and self.left is None

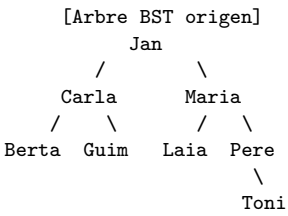
    def BorraFulles(self): #TO DO

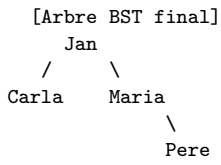
    def ComptaEsquerra(self):  #TO DO
```

```
if __name__=='__main__':
    arbre=BST()
    arbre.insereix("Jan")
    arbre.insereix("Carla")
    arbre.insereix("Berta")
    arbre.insereix("Guim")
    arbre.insereix("Maria")
    arbre.insereix("Pere")
    arbre.insereix("Toni")
    arbre.insereix("Laia")
    arbre.printArbrePreordre()
    print arbre.ComptaEsquerra()
    print
    arbre.BorraFulles()
    arbre.printArbrePreordre()
    print arbre.ComptaEsquerra()
    #Resultats execució
    #Jan Carla Berta Guim Maria Laia Pere Toni 3

    #Jan Carla Maria Pere 1
```

Apartat a) Implementa el mètode recursiu **BorraFulles**, tal que, donat un arbre BST, el deixi sense fulles, com a l'exemple següent.





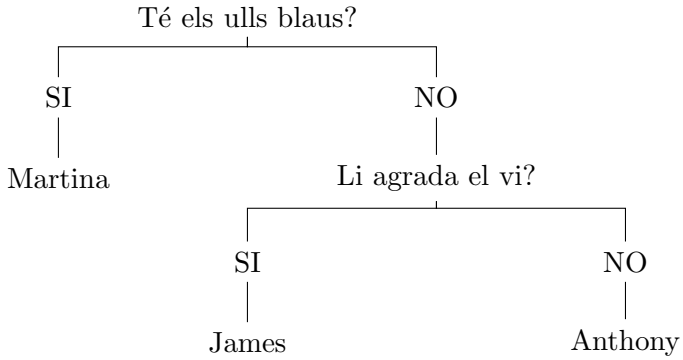
Apartat b) Implementa el mètode recursiu **ComptaEsquerra**, tal que, donat un BST retorni el número de fills esquerres en l'arbre. Un "fill esquerre" és un node que apareix com a arrel d'un subarbre esquerre d'un altre node. Per exemple, l'arbre BST origen té 3 fills esquerre (nodes Carla,Berta, Laia) i l'arbre BST final té 1 fill esquerre (node Carla)

EXERCICI 11.20 Amplieu la classe Arbre amb dos nous mètodes. Un que permeti grabar l'arbre en un fitxer de text i l'altre que permeti llegir-lo. Assumeix que el valor dels nodes són enters.

Per emmagatzemar-lo en el fitxer usa un recorregut postordre.

EXERCICI 11.21 Un arbre de decissió és un arbre que permet classificar una entitat a base de respondre preguntes, normalment de resposta binària. En aquests arbres, un node a correspon a una pregunta i els nodes fills d'a indiquen la resposta afirmativa o negativa a la pregunta. Els nodes fulla descriuen el resultat de la classificació.

Per exemple, el següent arbre permet classificar en un conjunt de tres persones:



Aquests arbres són típic en jocs que "encerten" les idees de les persones a base de fer preguntes. En aquest exercici es proposa que feu un joc d'aquests que permeti endevinar al jugador en quina persona de classe està pensant. A tal efecte, dissenyeu un arbre de decissió que permeti classificar totes les persones de la vostra classe. Creeu una classe específica que representi aquest arbre fent ús la la classe Arbre que ja teniu. Després creeu un programa que va preguntant i navegant per l'arbre fins arribar a una fulla, és a dir fins a obtenir la resposta.