Departament de Disseny i Programació de Sistemes Electrònics
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Prova final de TECPRO

17/06/2019 — Grau en Enginyeria de Sistemes TIC — 2H

COGNOMS:

NOM: GRUP de LAB:

Exercici 1. 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 2. 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 3. [**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 4. 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 5. 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!
```

Exercici 6. 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,

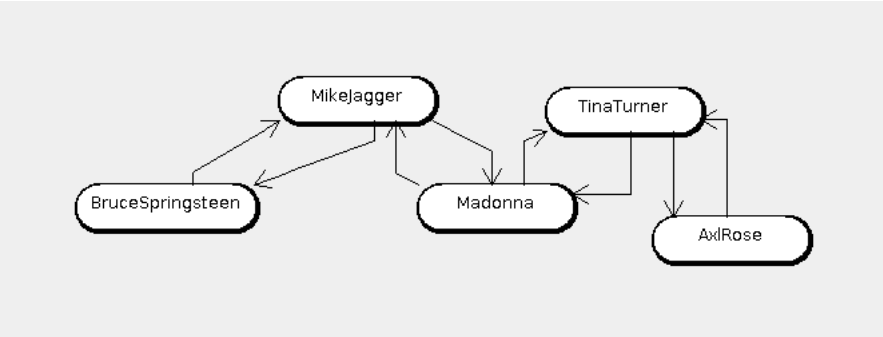


Figure 1: Xarxa de Connexions entre usuaris a la xarxa social

```
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')
```

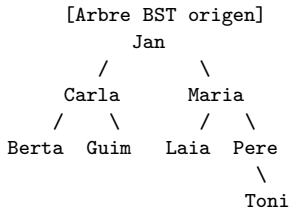
```
    1
    >>> minDistance({'MikeJagger':['Madonna','BruceSpringsteen'],'BruceSpringsteen':['MikeJagger'],
'Madonna':['MikeJagger', 'TinaTurner', 'AxlRose'],'TinaTurner':['Madonna', 'AxlRose'],'AxlRose':
['TinaTurner', 'Madonna']},'TinaTurner','BruceSpringsteen')
    3
    >>> minDistance({'MikeJagger':['Madonna','BruceSpringsteen'],'BruceSpringsteen':['MikeJagger'],
'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
    """
```

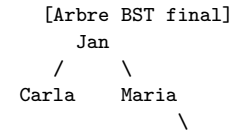
Exercici 7. BST. Se't proporciona la següent implementació de BST i els exemples de funcionament que segueixen.

```
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)