Temario

```
def rectabla(E, c):
  R = {}
  def C(i):
    if i == 1:
      R[1] =  0
    elif i == 2:
      R[2] = c(1,2)
    else:
      R[i] = min(R[i-1]+ c(i-1, i), R[i-2] + c(i-2, i))
    return R[i]
  return C(E)
```

```
def itertabla(E, c):
  R = {}
  R[1] = 0
  R[2] = c(1,2)
  for i in range(3, E+1):
    R[i] = min(R[i-1] + c(i-1, i), R[i-2]+ c(i-2, i))
  return R[E]
```

Problema de la moneda

```
def moneda(Q, v):
  def M(q):
    if q == 0:
      return 0
    else:
      return min(M(q-a) + 1 for a in v if q-a >= 0)
  return M(Q)

Q = 24
v = [1,2,5,10,20,50]

moneda(Q,v)
```

```
3
```

```
def moneda_recursivo(Q, v):
  def M(q):
    if q == 0:
      return 0, []
    else:
      va = 2**31
      miniari = []
```

```
      for a in v:
        if q - a >= 0:
          valor, l = M(q-a)
          valor += 1
          if valor < va:
            va = valor
            miniari = l
            miniari.append(a)
      return va, miniari
  return M(Q)

Q = 29
v = [1,2,5,10,20,50]

moneda_recursivo(Q,v)
```

```
(4, [20, 5, 2, 2])
```

```
Q = 29
v = [1,2,5,10,20,50]

def moneda_iterativo(Q, v):
  M = [None] * (Q+1)
  M[0] = 0
  for q in range(1, Q+1):
    M[q] = min(M[q-a] + 1 for a in v if q - a >= 0)
  return M[Q]

moneda_iterativo(Q, v)
```

```
4
```

Distancia de Levenshtein

```
def levenstein(x, y):
  D = {}
  D[0,0] = 0

  for i in range(1, len(x)+1):
    D[i, 0] = D[i-1, 0] +1
  for j in range(1, len(y)+1):
    D[0, j] = D[0, j-1] +1
    for i in range(1, len(x)+1):
      D[i, j] = min(D[i-1, 0] + 1,
                    D[i, j-1] + 1,
                    D[i-1, j-1]+ (x[i-1] != y[j-1]))
  return D[len(x), len(y)]

levenstein("ejemplo", "campos")
```

```
5
```

Longest Common Sequence

```
# Aplicacion: Longest Comon Subsequence
def LCS(x, y):
  lcs = {}
  for i in range(len(x)+1):
    lcs[i, 0] = 0
  for j in range(1, len(y)+1):
    lcs[0, j] = 0
    for i in range(1, len(x)+1):
      if x[i-1] != y[j-1]:
        lcs[i,j] = max(lcs[i-1, j], lcs[i, j-1])
      else:
        lcs[i,j] = lcs[i-1, j-1] +1 #Añado al contador de secuencias
  return lcs[len(x), len(y)]

LCS("comparsa", "causarte")
```

```
4
```

Mochila de Dora la exportadronjas

```
def iterative_knapsack_delante(W, v, w):
  Vcurr = {0:0}
  for vi,wi in zip(v,w):
    Vnext = {}
    for peso,benef in Vcurr.items():
      Vnext[peso] = max(benef, Vnext.get(peso,0))
      if peso+wi <= W:
        Vnext[peso+wi] = max(benef+vi, Vnext.get(peso+wi,0))
    Vcurr = Vnext
  return max(Vcurr.values())

v=[4,5,3,6]
W=8
v=[2,1,1,2]
#iterative_knapsack_delante(W, v, w)
```

```
def mochidora(W, v, w):
  R = {}
  def V(i, c):
    if i== 0 or c == o:
      R[i, c] = 0
    elif c-w[i] >= 0:
      if (i-1, c) not in R:
        V(i-1, c)
      elif (i-1, c - w[i]) not in R:
        V(i-1, c-w[i])
```

```
      else:
        R[i,c] = max(R[i-1, c], R[i-1, c-w[i]] +v[i]) # metemos a la mochila o no

    else:
      if (i-1, c) not in R:
        R[i-1, c] = V(i-1, c)
      R[i, c] = R[i-1, c]
    return R[i,c]
  return V(len(v), W)
```

Practicas

```
import numpy as np
import math

def levenshtein(x, y, threshold):
    M = np.ones((len(x) + 1, len(y) + 1))*np.inf
    for i in range(0, len(x) + 1):
        M[i, 0] = i
    for j in range(0, len(y) + 1):
        M[0, j] = j
    m = len(x)/len(y)
    for i in range(1, len(x) + 1):
        colMin = np.inf;
        for j in range(max(math.floor(m*i-threshold), 1), min(math.ceil(m*i+threshold), le
            if x[i - 1] == y[j - 1]:
                M[i, j] = min(M[i - 1, j] + 1, M[i, j - 1] + 1, M[i-1][j-1])
            else:
                M[i, j] = min(M[i - 1, j] + 1, M[i, j - 1] + 1, M[i-1][j-1] + 1)
            if colMin > M[i,j]:
                colMin = M[i,j]
        if colMin > threshold:
            return None
    return M[len(x), len(y)]



def damerau_levenshtein_restringida(x, y, threshold):
    M = np.ones((len(x) + 1, len(y) + 1))*np.inf
    for i in range(0, len(x) + 1):
        M[i, 0] = i
    for j in range(0, len(y) + 1):
        M[0, j] = j
    m = len(x)/len(y)
    for i in range(1, len(x) + 1):
        colMin = np.inf;
        for j in range(max(math.floor(m*i-threshold), 1), min(math.ceil(m*i+threshold), le
            if i > 1 and j > 1 and x[i - 2] == y[j - 1] and x[i - 1] == y[j - 2]:
                if x[i - 1] == y[j - 1]:
                    M[i, j] = min(M[i - 1, j] + 1, M[i, j - 1] + 1, M[i-1][j-1], M[i-2][j-
                else:
                    M[i, j] = min(M[i - 1, j] + 1, M[i, j - 1] + 1, M[i-1][j-1] + 1, M[i-2
```

```
            else:
                if x[i - 1] == y[j - 1]:
                    M[i, j] = min(M[i - 1, j] + 1, M[i, j - 1] + 1, M[i-1][j-1])
                else:
                    M[i, j] = min(M[i - 1, j] + 1, M[i, j - 1] + 1, M[i-1][j-1] + 1)
            if colMin > M[i,j]:
                colMin = M[i,j]
        if colMin > threshold:
            return None
    return M[len(x), len(y)]


def damerau_levenshtein_intermedia(x, y,threshold):
    M = np.ones((len(x) + 1, len(y) + 1))*np.inf
    for i in range(0, len(x) + 1):
        M[i, 0] = i
    for j in range(0, len(y) + 1):
        M[0, j] = j
    m = len(x)/len(y)
    for i in range(1, len(x) + 1):
        colMin = np.inf;
        for j in range(max(math.floor(m*i-threshold), 1), min(math.ceil(m*i+threshold), le
            minInit = 0
            if x[i - 1] == y[j - 1]:
                minInit = min(M[i-1, j] + 1, M[i, j-1] + 1, M[i-1][j-1])
            else:
                minInit = min(M[i-1, j] + 1, M[i, j-1] + 1, M[i-1][j-1] + 1)

            if j > 1 and i > 1 and x[i - 2] == y[j - 1] and x[i - 1] == y[j - 2]:
                M[i,j] = min(minInit, M[i-2][j-2] + 1)
            elif j > 2 and i > 1 and x[i-2] == y[j-1] and x[i-1] == y[j-3]:
                M[i,j] = min(minInit, M[i-2][j-3] + 2)
            elif i > 2 and j > 1 and x[i - 3] == y[j-1] and x[i-1] == y[j-2]:
                M[i,j] = min(minInit, M[i-3][j-2] + 2)
            else:
                M[i,j] = minInit
            if colMin > M[i,j]:
                colMin = M[i,j]
        if colMin > threshold:
            return None
    return M[len(x), len(y)]


x = "google"
y = {"google","kooble","bubble", "gogole","ggole"}

for w in y:
  print(levenshtein(x, w, 2))
```

```
None
2.0
2.0
0.0
2.0
```

```
def nqueens(n):
    sol = [None]*n
    def show_solution(solution):
        output = ["    "+"".join(str((i+1) % 10) for i in  range(n))+"\n"]
        for i in range(n):
            output.append("%3d %s\n" % (i+1,"".join("X" if solution[j]==i else "." for j i
        return "".join(output)

    def is_promising(longSol, queen):
        return all(queen != sol[i] and longSol-i != abs(queen-sol[i]) for i in range(longS

    def backtracking(longSol):
        if longSol == n:
          print(sol)
          return show_solution(sol)
        else:
            for queen in range(n):
                if is_promising(longSol, queen):
                    sol[longSol] = queen
                    r = backtracking(longSol+1)
                    if r is not None:
                        return r
        return None # explicit
    return backtracking(0)

print(nqueens(8))
```

```
[0, 4, 7, 5, 2, 6, 1, 3]
    12345678
  1 X.......
  2 ......X.
  3 ....X...
  4 .......X
  5 .X......
  6 ...X....
  7 .....X..
  8 ..X.....
```

```
def nqueens(n, allSolutions):
    sol = [None]*n
    def show_solution(solution):
        output = ["    "+"".join(str((i+1) % 10) for i in  range(n))+"\n"]
        for i in range(n):
            output.append("%3d %s\n" % (i+1,"".join("X" if solution[j]==i else "-" for j i
        return "".join(output)
    def is_promising(longSol, queen):
        return all(queen != sol[i] and longSol-i != abs(queen-sol[i]) for i in range(longS
    def backtracking(longSol):
        if longSol == n:
            results.append(show_solution(sol))
        if allSolutions or len(results) == 0:
            for queen in range(n):
                if is_promising(longSol, queen):
                    sol[longSol] = queen
                    backtracking(longSol+1)
```

```
        return None # explicit
    results = []
    backtracking(0)
    return results
for a in nqueens(4, 'TODAS'):
  print(a)
```

```
     1234
    1 --X-
    2 X---
    3 ---X
    4 -X--

     1234
    1 -X--
    2 ---X
    3 X---
    4 --X-
```

Ciclo Hamiltoniano

```
from random import sample

def hamiltoniano(G):
  def backtracking(path):
    if len(path) == len(G.V):
      if (path[-1] == path[0]) in G.E:
        return path + [path[0]]
      else:
        for v in G.succs(path[-1]):
          if v not in path:
            found = backtracking(path + [v])
            if found:
              return found
      return None
    [random_vertex] = sample(G.V, 1)
    return backtracking([random_vertex])
```

Reinas

```
class NQueensSolver1:
  def __init__(self, n):
    self.n = n

  def is_complete(self, s):
    return len(s) == self.n

  def is_promising(self, s, row):
    return all(row != s[i] and len(s)-i != abs(row-s[i]) for i in range(len(s)))

  def backtracking(self, s):
    if self.is_complete(s):
```

```
      return s

    for row in range(self.n):
      if self.is_promising(s, row):
        found = self.backtracking(s+[row])
        if found != None:
          return found
    return None

  def solve(self):
    return self.backtracking([])

NQueensSolver1(8).solve()
```

```
[0, 4, 7, 5, 2, 6, 1, 3]
```

```
[0, 4, 7, 5, 2, 6, 1, 3]
```

backtracking + mochila de Dora la exportadronjas

```
def subset(w, W):
  x = [0] * len(w)
  def backtracking(i):
    if i == len(w)+1: #is_complete
      if sum(w[j] * x[j] for j in range(1, len(w)+1) == W):
        return x
    else:
      for x[i] in [0, 1]: # Exploring new branch
        if sum([w[j] * x[j] for j in range(1, i + 1)]) <= W: #is_promising
          found = backtracking(i+1)
          if found != None:
            return found
    return None
  return backtracking(1)

subset(w=[4,5,3,6] ,W=8)
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-31-a86c21f14083> in <module>()
     14   return backtracking(1)
```

```
  12345
1 X....
2 ...X.
3 .X...
4 ....X
5 ..X..
```

SEARCH STACK OVERFLOW