Apuntes + ejercicios de Sage - MatDis

June 14, 2021

1 LISTAS

Van entre []

Para posicionarnos en lugar de la lista ponemos: nombrelista[posicion]

Algunas propiedades:

- l.remove(elemento) —> si el elemento está en la lista elimina su primera aparición
- l.count(elemento) —> cuenta las veces que aparece elemento en lista
- l.index(elemento) —> posición que ocupa un elemento en lista
- l.append(elemento) —> añade elemento al final de la lista
- l.insert(k, elemento) —> añade elemento a la lista en la posicion k
- l.extend(lista) —> añade al final de l todos los elementos de lista

Ejemplo de lista:

```
lista = ['OROS', 'COPAS', 'ESPADAS', 'BASTOS']
lista[-2]   #accedo a la posición -2
lista.index('COPAS')   #devuelve la posición que ocupa en la lista
Parent(1)
```

```
'ESPADAS'
1
<sage.structure.parent.Parent object at 0x7fd9e29454a0>
```

Otro ejemplo de lista:

```
z = [1, 2, 3, 4, 5, 6, 7, 8, 9]
z[-5:-1]  #devuelve los valores desde la posición -5 (desde atrás \
   empezamosa contar en el -1)
z[-2:]
```

1.1 Manipulación de listas

- Para ordenarla alfabéticamente utilizamos: ***sort***
- Para moficicar un elemento: ***_nombrelista_[posicion]***
- Para operar con un elemento: ***sum*** y ***prod***
- Para añadir un elemento: ***append***
- Para concatener dos listas: ***extend***
- Para intercalar los elementos de dos listas: ***list(zip(lista1, lista2))***

```
lista.sort(); lista
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
z = list(zip(lista1, lista2)); z
lista[0] = 'hola'; lista   #en la posicion cero de la lista se va a \
   añadir un Hola
sum[1..10]   #suma todos los elementos del uno al diez
lista.extend([1..3]); lista
list(zip([1..3], lista))
```

```
['BASTOS', 'COPAS', 'ESPADAS', 'OROS']
[(1, 4), (2, 5), (3, 6)]
['hola', 'COPAS', 'ESPADAS', 'OROS']
55
['hola', 'COPAS', 'ESPADAS', 'OROS', 1, 2, 3]
[(1, 'hola'), (2, 'COPAS'), (3, 'ESPADAS')]
```

1.2 Construyendo listas iterando sobre listas

```
lista = [2..6]
2 in lista   #preguntamos si está el número dos en la lista
```

```
True
```

1.3 Construcción de listas a partir de otra lista filtrando los elementos mediante una condición lógica

```
dos = [100, 102..999]
tres = [102, 105..999]
seis = [x for x in dos if x in tres]; seis
```

```
[102, 108, 114, 120, 126, 132, 138, 144, 150, 156, 162, 168, 174, 180, 186, 192, 198, 204,
210, 216, 222, 228, 234, 240, 246, 252, 258, 264, 270, 276, 282, 288, 294, 300, 306, 312,
318, 324, 330, 336, 342, 348, 354, 360, 366, 372, 378, 384, 390, 396, 402, 408, 414, 420,
426, 432, 438, 444, 450, 456, 462, 468, 474, 480, 486, 492, 498, 504, 510, 516, 522, 528,
```

534, 540, 546, 552, 558, 564, 570, 576, 582, 588, 594, 600, 606, 612, 618, 624, 630, 636, 642, 648, 654, 660, 666, 672, 678, 684, 690, 696, 702, 708, 714, 720, 726, 732, 738, 744, 750, 756, 762, 768, 774, 780, 786, 792, 798, 804, 810, 816, 822, 828, 834, 840, 846, 852, 858, 864, 870, 876, 882, 888, 894, 900, 906, 912, 918, 924, 930, 936, 942, 948, 954, 960, 966, 972, 978, 984, 990, 996]

2 CONSTRUCCIÓN DE TABLAS DE VERDAD

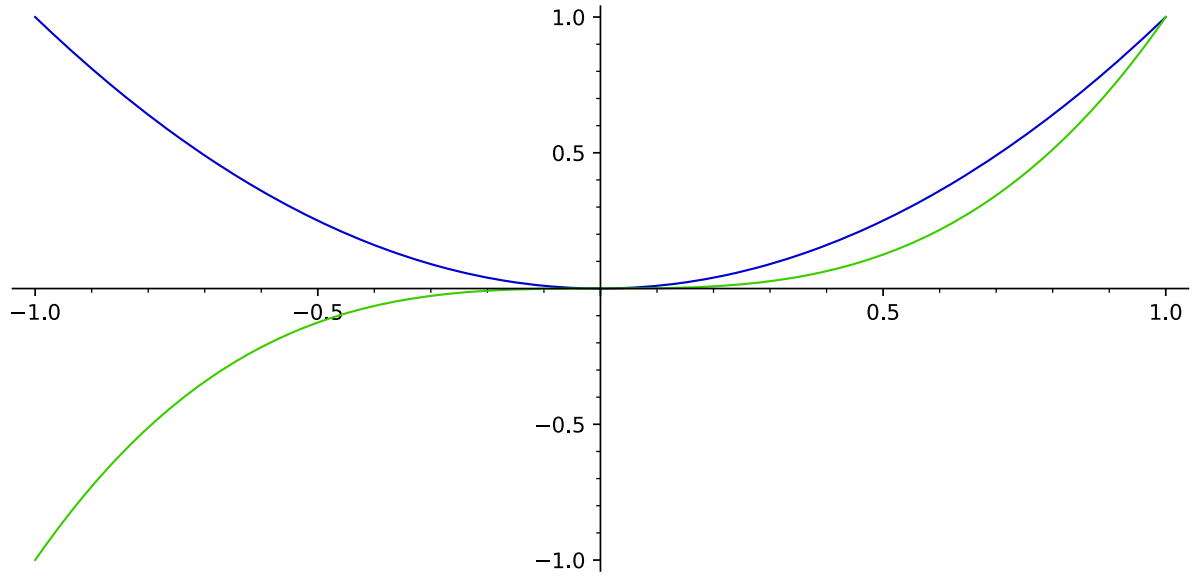
```
print('-----------------------')
print('   p    q       p&q')
print('-----------------------')
for p in [True, False]:
    for q in [True, False]:
        print(p, q, p and q)
```

```
-----------------------
  p  q    p&q
-----------------------
True True True
True False False
False True False
False False False
```

3 DIBUJAR UNA GRÁFICA

Utilizamos el comando ***plot***

```
plot([x^2, x^3])
```



4 INVERSOS MODULARES

Para que exista el inverso x módulo m es necesario que: mcd(x, m) = 1, es decir, que non tengan divisores en común. Hay varias formas de calcular el inverso, si existe: - $mod(1/x, m) = 1/mod(x, m)$, como clase de restos módulo m - $inverse\_mod(x, m) = 1/x\%m$ (no confundir con $1/x\%m$)), como número entero - $xgcd(x, m)$, mediante el algoritmo de Euclides extendido (Teorema de Bézout)

```
mod(1/3, 5)
inverse_mod(3, 5)
xgcd(3, 5)
xgcd(3, 5)[1]
```

```
2
2
(1, 2, -1)
2
```

5 EXPONENCIACIÓN MODULAR

Para hacer b^e y después calcuar el módulo. Primero, se calcula $b \ mod \ n$ y después se eleva a e, lo que requiere e productos.

A continuación, veremos un algoritmo que reduce drásticamente el número de operaciones para llevar a cabo los cálculos. La idea es que, si $e = e_0 + e_1 2^1 + e_2 2^2 + \cdots + e_k 2^k$ se tiene que $b^{e_0+e_1 2^1+e_2 2^2+\cdots+e_k 2^k} = b^{e_0} \times b^{e_1 2^1} \times b^{e_2 2^2} \times \cdots \times b^{e_k 2^k} = \left(b^{2^0}\right)^{e_0} \times \left(b^{2^1}\right)^{e_1} \times \left(b^{2^2}\right)^{e_2} \times \cdots \times \left(b^{2^k}\right)^{e_k}$

Ahora, hay que observar que: 1. Si $e_i = 0$ el factor $\left(b^{2^i}\right)^{e_i}$ toma el valor 1, no es necesario multiplicarlo. 2. Las bases son, cada una, el cuadrado de la anterior, por lo que la función puede ser así:

```
def exp_mod(b, e, n):
    E = e.digits(2)                 # la lista de cifras de e en base 2
    t = len(E)                      # la cantidad de cifras
    base = mod(b, n)                # las operaciones, a partir de aquí,\
    son de aritmética modular
    num = 1                         # para ir multiplicando los valores \
   que vamos construyendo
    for i in range(t):
        if E[i] == 1:
            num *= base             # multiplicamos por la base
        base *= base                # elevamos la base al cuadrado
    return num
```

```
%time      # para medir el tiempo de ejecución
exp_mod(123421341234, 12342342423423443556756756345234213, 234)
```

```
144
CPU time: 0.01 s, Wall time: 0.01 s
```

```
12345.digits(10)
```

```
ZZ([5, 4, 3, 2, 1], 10)
12345.digits(2)
power_mod(10, 5, 7)              # 10^5 mod 7
```

```
[5, 4, 3, 2, 1]
12345
[1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1]
5
```

6 PERMUTACIONES

Son selecciones donde sí importa el orden.

Podemos hacer permutaciones de: - Permutations(lista) - Permutations(número) - Permutations(conjunto) - Permutacions(cadena)

Cada una de esas modalidades puede ser llamada con un segundo parámetro, un entero positivo. - Permutations(lista, k) - Permutations(número, k) - Permutations(conjunto, k) - Permutacions(cadena, k)

Permutations() crea un objeto iterable al que se le puede aplicar varios métodos, como ***.cardinality()***, ***.list()*** o ***.random_element()***

```
Permutations([1,2,3])
```

```
Standard permutations of 3
```

```
Permutations([1,2,3]).list()
```

```
[[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]
```

```
Permutations([1,2,3]).random_element()
```

```
[3, 2, 1]
```

```
L = Permutations([1,2,3])
```

```
L.cardinality()
```

```
6
```

```
Permutations([1,2,3],2).list()
```

```
[[1, 2], [1, 3], [2, 1], [2, 3], [3, 1], [3, 2]]
```

```
[p for p in L if p[0]==3]
```

```
[[3, 1, 2], [3, 2, 1]]
```

```
a=Permutations('abcde')
a.list()[:5]
```

```
[['a', 'b', 'c', 'd', 'e'], ['a', 'b', 'c', 'e', 'd'], ['a', 'b', 'd', 'c', 'e'], ['a',
'b', 'd', 'e', 'c'], ['a', 'b', 'e', 'c', 'd']]
```

```
[''.join(p) for p in a]
```

```
['abcde', 'abced', 'abdce', 'abdec', 'abecd', 'abedc', 'acbde', 'acbed', 'acdbe', 'acdeb',
'acebd', 'acedb', 'adbce', 'adbec', 'adcbe', 'adceb', 'adebc', 'adecb', 'aebcd', 'aebdc',
```

'aecbd', 'aecdb', 'aedbc', 'aedcb', 'bacde', 'baced', 'badce', 'badec', 'baecd', 'baedc',
'bcade', 'bcaed', 'bcdae', 'bcdea', 'bcead', 'bceda', 'bdace', 'bdaec', 'bdcae', 'bdcea',
'bdeac', 'bdeca', 'beacd', 'beadc', 'becad', 'becda', 'bedac', 'bedca', 'cabde', 'cabed',
'cadbe', 'cadeb', 'caebd', 'caedb', 'cbade', 'cbaed', 'cbdae', 'cbdea', 'cbead', 'cbeda',
'cdabe', 'cdaeb', 'cdbae', 'cdbea', 'cdeab', 'cdeba', 'ceabd', 'ceadb', 'cebad', 'cebda',
'cedab', 'cedba', 'dabce', 'dabec', 'dacbe', 'daceb', 'daebc', 'daecb', 'dbace', 'dbaec',
'dbcae', 'dbcea', 'dbeac', 'dbeca', 'dcabe', 'dcaeb', 'dcbae', 'dcbea', 'dceab', 'dceba',
'deabc', 'deacb', 'debac', 'debca', 'decab', 'decba', 'eabcd', 'eabdc', 'eacbd', 'eacdb',
'eadbc', 'eadcb', 'ebacd', 'ebadc', 'ebcad', 'ebcda', 'ebdac', 'ebdca', 'ecabd', 'ecadb',
'ecbad', 'ecbda', 'ecdab', 'ecdba', 'edabc', 'edacb', 'edbac', 'edbca', 'edcab', 'edcba']

```
Permutations(4) == factorial(4)
```
False

7 COMBINACIONES o SUBSETS

Son selecciones donde no importa el orden.

Podemos hacer combinaciones de: - Combinations(lista, k) - Combinations(número, k) - Combinations(conjunto, k) - Combinations(cadena, k)

Si se llama sin segundo parámetro, devuelve todas las selecciones de cualquier tamaño posible. Podemos usar ***Subsets()***

Recordar que multinomial: $(k_1, ..., k_s) = \frac{(k_1+...+k_s)!}{k_1! \cdot ... \cdot k_s!}$

```
L = Combinations([1,2,3,4], 2)
L.list()
```
[[1, 2], [1, 3], [1, 4], [2, 3], [2, 4], [3, 4]]

```
Subsets(4).list()
```
[{}, {1}, {2}, {3}, {4}, {1, 2}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, {3, 4}, {1, 2, 3}, {1, 2, 4}, {1, 3, 4}, {2, 3, 4}, {1, 2, 3, 4}]

```
Permutations('MATEMATICA').cardinality() == multinomial(2,3,2,1,1,1)
```
True

```
lista = [x for x in Permutations('02468',3) if x[0] != '0']
['' .join(x) for x in lista]
```
['204', '206', '208', '240', '246', '248', '260', '264', '268', '280', '284', '286', '402', '406', '408', '420', '426', '428', '460', '462', '468', '480', '482', '486', '602', '604', '608', '620', '624', '628', '640', '642', '648', '680', '682', '684', '802', '804', '806', '820', '824', '826', '840', '842', '846', '860', '862', '864']

```
[ ZZ(''.join(x)) for x in lista ]
```
[204, 206, 208, 240, 246, 248, 260, 264, 268, 280, 284, 286, 402, 406, 408, 420, 426, 428, 460, 462, 468, 480, 482, 486, 602, 604, 608, 620, 624, 628, 640, 642, 648, 680, 682, 684, 802, 804, 806, 820, 824, 826, 840, 842, 846, 860, 862, 864]

```
#Todos los números con tres cifras pares
[ZZ(''.join(x)) for x in Permutations('02468'*3,3) if x[0]!='0']
```

[200, 202, 204, 206, 208, 220, 222, 224, 226, 228, 240, 242, 244, 246, 248, 260, 262, 264, 266, 268, 280, 282, 284, 286, 288, 400, 402, 404, 406, 408, 420, 422, 424, 426, 428, 440, 442, 444, 446, 448, 460, 462, 464, 466, 468, 480, 482, 484, 486, 488, 600, 602, 604, 606, 608, 620, 622, 624, 626, 628, 640, 642, 644, 646, 648, 660, 662, 664, 666, 668, 680, 682, 684, 686, 688, 800, 802, 804, 806, 808, 820, 822, 824, 826, 828, 840, 842, 844, 846, 848, 860, 862, 864, 866, 868, 880, 882, 884, 886, 888]

8 Ejercicio mítico de examen

El torneo de tenis: Denotamos por 'AABBA' el resultado de un partido de tenis en el que gana el jugador 'A' por 3 juegos a 2, ganando 'B' los juegos 3 y 4. 1. Construye con Sage todos los posibles resultados en los que 'A' le gana a 'B' por 3 juegos a 2 2. Construye con Sage todos los posibles resultados en los que 'A' le gana a 'B' 3. Si, en vez de 3, para ganar el partido hubiese que ganar en n juegos, con n un número entero positivo, define una función para calcular el número de posibles resultados

Pregunta 1:

Ya que siempre acaba la cadea en 'A', tenemos dos posibilidades: 1. Permutar 'AAABB' y quedarnos con las cadenas que acaban en 'A' 2. Permutar 'AABB' y añadirle por la derecha 'A' a la cadena resultante

```
pregunta_1_opcion_1 = [''.join(x) for x in Permutations('AAABB') if \
    x[-1] == 'A']
pregunta_1_opcion_1
```

['AABBA', 'ABABA', 'ABBAA', 'BAABA', 'BABAA', 'BBAAA']

Pregunta 2:

Todo lo que hay que hacer es concatenar las listas correspondientes a los resultados 3-0, 3-1, 3-2 y de nuevo varias opciones: 1. Permutar 'AAA', 'AAAB' y 'AAABB', filtrar aquellas cadenas que acaban en 'A' y juntas las listas 2. Permutar 'AAA', 'AAAB' y 'AAABB', juntar las correspondientes listas y filtrar aquellas cadenas que acaban en 'A' 3. Permutar 'AA', 'AAB' y 'AABB', juntar las correspondientes lista y añadir a cada una de las cadenas de la lista resultante 'A' por la derecha

```
preguta_2_opcion_1 = [''.join(x) for x in Permutations('AAA') if x\
    [-1]=='A'] + [''.join(x) for x in \
Permutations('AAAB') if x[-1]=='A'] + [''.join(x) for x in \
    Permutations('AAABB') if x[-1]=='A']
preguta_2_opcion_1
```

['AAA', 'AABA', 'ABAA', 'BAAA', 'AABBA', 'ABABA', 'ABBAA', 'BAABA', 'BABAA', 'BBAA

Pregunta 3:

```
def torneo(n):
    resultados = []
```

```
    for k in range(n):   #hacemos todos los posibles resultdos: n a \
   0, n a 1, n a 2 ... n a (n-1)
        resultados += [''.join(x) for x in Permutations('A'*n + 'B'*\
   k) if x[-1] == 'A']
        # cojemos n copias de 'A' y k copias de 'B'
    return resultados
torneo(3)
['AAA', 'AABA', 'ABAA', 'BAAA', 'AABBA', 'ABABA', 'ABBAA', 'BAABA', 'BABAA', 'BBAA
```

9 EJERCICIO 2

Hay una serie de imágenes en las que el camino de la figura 1 podemos representarlo de la siguiente forma: 'HHVVHVHVHV', mientras que el de la figura 4 corresponde a 'HVVVHHVHHV'. Se trata entones de: 1. Representar con Sage todos los posibles caminos de longitud mínimo 2. Contar con Sage el número de caminos que no cruzan la diagonal para arriba (pudiendo tocarla) 3. Definir una función que dados dos números enteros, a y b, nos devuelve todos los caminos para desplazarnos por una retícula como las de arriba entre el punto $(0, 0)$ y el punto (a, b)

```
def cam(a, b):
    return [''.join(p) for p in Permutations('H'*a +  'V'*b)]
cam(3, 3)
['HHHVVV', 'HHVHVV', 'HHVVHV', 'HHVVVH', 'HVHHVV', 'HVHVHV', 'HVHVVH', 'HVVHH
'HVVVHH', 'VHHHVV', 'VHHVHV', 'VHHVVH', 'VHVHHV', 'VHVHVH', 'VHVVHH', 'VVHHH
'VVHVHH', 'VVVHHH']
```

10 RELACIONES DE RECURRENCIA

10.1 Sucesión de Fibonacci

La relación de recurrencia que caracteriza la sucesión de Fibonacci es: $F(n) = F(n-1) + F(n-2)$, con las condiciones iniciales $F(0) = 0$ y $F(1) = 1$.

Definimos entonces una función en Sage la calcule:

```
def Fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return Fib(n - 1) + Fib(n - 2)
Fib(4)
3
```

10.2 Torres de Hannoi

Siendo $H(n)$ la cantidad de movimientos necesarios para completar el juego de las torres de Hannoi con n discos, sabemos que se cumple la siguiente relación de recurrencia: $H(n) = 2H(n-1)+1$, que junto con la condición inicial: $H(1) = 1$ define la sucesión H_n

```
def hannoi(n):
    if n == 1:
        return 1
    else:
        return 2*hannoi(n - 1) + 1
```

```
for n in [1..20]:
    print('Con', n, 'discos:', hannoi(n), 'movimientos')
```

```
Con 1 discos: 1 movimientos
Con 2 discos: 3 movimientos
Con 3 discos: 7 movimientos
Con 4 discos: 15 movimientos
Con 5 discos: 31 movimientos
Con 6 discos: 63 movimientos
Con 7 discos: 127 movimientos
Con 8 discos: 255 movimientos
Con 9 discos: 511 movimientos
Con 10 discos: 1023 movimientos
Con 11 discos: 2047 movimientos
Con 12 discos: 4095 movimientos
Con 13 discos: 8191 movimientos
Con 14 discos: 16383 movimientos
Con 15 discos: 32767 movimientos
Con 16 discos: 65535 movimientos
Con 17 discos: 131071 movimientos
Con 18 discos: 262143 movimientos
Con 19 discos: 524287 movimientos
Con 20 discos: 1048575 movimientos
```

10.3 Algoritmo para el cálculo del máximo común divisor usando el algoritmo de Euclides

```
def mcd(a, b):
    if b == 0:
        return a
    else:
        return mcd(b, a%b)      #algoritmo de Euclides
mcd(234, 3456)
```

```
18
```

Recordar cómo se calculan las cifras de un número n en una base cualquiera, b:

1. Dividimos n entre b y el resto es la primera cifra (la menos significativa)
2. Dividimos el cociente anterior de nuevo entre b, y el resto es la segunda cifra por la derecha
3. .....
4. Seguimos hasta que el cociente nos dea 0

Claramente es un proceso recursivo, en el que se va rebajando el número en cada iteración

```
def cif(n, b):
    if n < b:              # si n menor que b
        return str(n)      # el cociente entero n/b será 0
    else:
        return cif(n//b, b) + str(n%b)
        #calculamos las cifras de n//b y le añadimos la menos \
   significativa, que era n modulo b
cif(12341234123, 10)
```

```
'12341234123'
```

Definimos ahora la función NUM que recupera el número en decimal a partir de su representación en una base b, b 26

```
def NUM(s, b):
    guarismos = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    if len(s) == 1:
        return guarismos.find(s)   # posición que ocupa la cifra \
   dentro de guarismos
    else:
        return NUM(s[:-1], b) * b + guarismos.find(s[-1])
        # pensar que, en base b, 'xyzt' = 'xyz0' + 't' = 'xyz'*b + '\
   t'
NUM('4D2', 16)
```

```
1234
```

Recordar cómo se calcularon en clase el número de cadenas de n bits sin ceros consecutivos: 1. Separar las cadenas segundo sean el último bit, '0' o '1' - Si el último bit es '0', necesariamente el penúltimo es '1', porque no pueden ir '00', y en los $n-2$ bits restanres no puede haber '00' - Si el último bit es '1', en los $n-1$ restantes no puede haber '00' 2. Dualmente: - Si partimos de una cadena de $n-2$ bits sin '00', añadiéndole '10' al final, tenemos una cadena de n bits que acaba en '0' - Si partimos de una cadena de $n-1$ bits sin '00', añadiéndole '1' al final, tenemos una cadena de n bits sin '00'

```
def kdas(n):
    if n == 1:
        return ['0', '1'] # la lista con todas las cadenas de 1 bit \
   sin ceros consecutivos
    elif n == 2:
        return ['01', '10', '11'] # la lista con todas las cedenas \
   de 2 bits sin ceros consecutivos
```

```
    else:
        return [c + '10' for c in kdas(n-2)] + [c + '1' for c in \
   kdas(n-1)]
kdas(3)
```

```
['010', '110', '011', '101', '111']
```

11 EXÁMENES

12 Examen Enero 2021

12.0.1 Pregunta 1.a)

Coa cadea 'hhmmm' representamos unha fila na que 3 homes e 2 mulleres agardar para recibir a vacina contra a SARS-Cov-2, sendo un home o primeiro na fila.

Construír con SageMath a lista H3M2 formada polas cadeas que representan todas as maneiras en que 3 homes e 2 mulleres poden facer cola para recibir a vacina, sen que as dúas mulleres vaian seguidas na cola.

```
H3M2 = [''.join(i) for i in Permutations('hhhmm') if not 'mm' in ''.\
   join(i)]; H3M2
```

```
['hhmhm', 'hmhhm', 'hmhmh', 'mhhhm', 'mhhmh', 'mhmhh']
```

12.0.2 Pregunta 1.b)

Definir a función boa, que dada C, unha cadea de calquera lonxitude, nos devolve True se en C o caracter m non aparece en dúas posicións consecutivas

```
def boa(C):
    n = len(C)
    for i in range(n-1):
        if 'm' == C[i] == C[i+1]:  #si aparece en dos posiciones \
   consecutivas
            return False
    return True
```

```
# otra forma más fácil de hacerlo es:
def boa2(C):
    if 'mm' in C:
        return False
    else:
        return True
```

12.0.3 Pregunta 1.c)

Definir a función **colas** que para un par de números enteiros non negativos, (a, b) nos devolve a lista coas cadeas que representan todas as maneiras en que a homes e b mulleres poden facer cola para vacinarse, sen que haxa dúas mulleres consecutivas.

```
def colas(a, b):
    l = [''.join(i) for i in Permutations('h'*a + 'm'*b) if not 'mm'\
    in ''.join(i)]
    # 'h'*a + 'm'*b con esto asigno los valores a y b a la cantidad \
   de hombre y mujeres que hay
    return l
```

```
colas(3, 1)
```

```
['hhhm', 'hhmh', 'hmhh', 'mhhh']
```

12.0.4 Pregunta 3.

Definir a función pan mediante un algoritmo recursivo que para cada número enteiro non negativo, nn, nos devolve a lista coas cadeas que representan todas as formas distintas de facer colas de nn persoas de xeito que nunca vaian mulleres consecutivas.

```
def pan(n):
    if n == 0:
        return ['']
    elif n == 1:
        return ['h', 'm']
    else:
        return ['h'+p for p in pan(n-1)] + ['mh'+p for p in pan(n-2)\
   ]
```

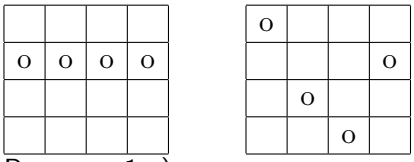
```
pan(4)
```

```
['hhhh', 'hhhm', 'hhmh', 'hmhh', 'hmhm', 'mhhh', 'mhhm', 'mhmh']
```

13 EXAMEN ENERO 2020

13.0.1 Taboleiros

Para representarmos a posición de 4 moedas sobre unha cuadrícula 4×4, sabendo que non hai dúas na mesma columna, podemos utilizar unha lista de 4 elementos que nos indica a fila que ocupa en cada unha das 4 columnas. Así, por exemplo, mentres que [2,2,2,2] nos di que todas as moedas están sobre unha mesma fila (a segunda), [1,3,4,2] representa unha colocación na que as moedas están todas en filas diferentes (ver as figuras debaixo).



Pregunta 1.a)

Construír con SageMath todas as listas que representen posibles colocacións de 4 moedas sobre un taboleiro de 4×4 casiñas en columnas e filas distintas, de xeito que non haxa dúas siutuadas en cadros contiguos (que se toquen en algún vértice).

```
# Para termos unha moeda en cada fila, podemos coller calquera \
    permutación de [1,2,3,4]
P4 = Permutations(4).list();P4
```

```
[[1, 2, 3, 4], [1, 2, 4, 3], [1, 3, 2, 4], [1, 3, 4, 2], [1, 4, 2, 3], [1, 4, 3, 2], [2,
1, 3, 4], [2, 1, 4, 3], [2, 3, 1, 4], [2, 3, 4, 1], [2, 4, 1, 3], [2, 4, 3, 1], [3, 1, 2,
4], [3, 1, 4, 2], [3, 2, 1, 4], [3, 2, 4, 1], [3, 4, 1, 2], [3, 4, 2, 1], [4, 1, 2, 3],
[4, 1, 3, 2], [4, 2, 1, 3], [4, 2, 3, 1], [4, 3, 1, 2], [4, 3, 2, 1]]
```

Pregunta 1.b)

Definir a función **reis** que collido calquera número natural $n > 3$ nos devolve todas as posibles maneiras de colocar n moedas sobre unha retícula $n \times n$, de xeito que todas estean en filas e columnas diferentes e non haxa dúas en cuadros contiguos.

```
def reis(n):
    lista = []
    contar = 0
    for x in Permutations(n):
        contar = 0
        for i in [0..n-2]:
            if abs(x[i] - x[i+1])!=1:
                contar += 1
        if contar == (n-1):
            lista.append(x)
    return lista

reis(5)
```

```
[[1, 3, 5, 2, 4], [1, 4, 2, 5, 3], [2, 4, 1, 3, 5], [2, 4, 1, 5, 3], [2, 5, 3, 1, 4], [3,
1, 4, 2, 5], [3, 1, 5, 2, 4], [3, 5, 1, 4, 2], [3, 5, 2, 4, 1], [4, 1, 3, 5, 2], [4, 2, 5,
1, 3], [4, 2, 5, 3, 1], [5, 2, 4, 1, 3], [5, 3, 1, 4, 2]]
```

Pregunta 1.c)

De cantas maneiras se poden situar 8 reis sobre un taboleiro de xadrez, de xeito que non se ameacen entre si e que non haxa dous na mesma fila nin na mesma columna.

```
len(reis(8))
```

```
5242
```

13.0.2 Exponenciación binaria

Lembrarás que o algoritmo de *exponenciación rápida*, ou *exponenciación binaria* para calcular $b^e \mod m$ consiste en escribir $e = \sum_{i=0}^{k} e_i 2^i$ e facer

$$b^{\left(\sum_{i=0}^{k} e_i 2^i\right)} = \prod_{i=0}^{k} \left(b^{2^i}\right)^{e_i} \mod m$$

e observar que $b^{2^{i+1}} = \left(b^{2^i}\right)^2$

Pregunta 2

Definir a función exp_bin para implementar este algoritmo.

```
def exp_bin(b, e, m):
    x = e.digits(2)
    num = 1
    j = 0
    for i in x:
        num *= mod((b^(2^j))^i, m)
        j += 1
    return num

exp_bin(2, 120, 54)
```

46

```
#También se puede hacer:
def exp_bin(b,e,n):
    E = e.digits(2)                    # a lista de cifras de e en base\
    2
    t = len(E)                         # a cantidade de cifras
    base = mod(b,n)                    # as operacións, a partir de aqu\
   í, son en aritmética modular
    num = 1                            # para irmos multiplicando os \
   valores que imos construíndo
    for i in range(t):
        if E[i] == 1:                  # se a correspondente cifra é 1
            num *= base                # multiplicamos pola base
        base *= base                   # elevando a base ao cadrado
    return num
```

Pregunta 3.a)

Definir a función **revolver** mediante un **algoritmo recursivo** que aplicada a unha cadea nos devolva outra cadea formada polos mesmos caracteres da cadea inicial pero en orden inverso. Por exemplo, **revolver**('abc') = 'cba'.

```
def revolver(s):
    n = len(s)
    if n == 0:
        return s
```

```
    else:
        return s[-1] + revolver(s[:-1])

revolver('123456789')
```
'987654321'

```
#Otra opción es hacer:
# outra opción
def resolver2(s):
    n = len(s)
    if n == 0:
        return s
    else:
        return revolver(s[1:]) + s[0]
resolver2('123456789')
```
'987654321'

Pregunta 3.b)

Definir a función **cat** mediante un **algoritmo recursivo** que aplicada a un número enteiro non negativo *n* nos devolva todas as cadea formadas por *n* ceros e *n* uns de xeito que, contando dende a esquerda, nunca o número de ceros superé ó número de uns.

```
def cat(n):
    if n == 0:
        return ['']
    else:
        K = []
        for i in [1..n]:
            K += ['1'+x+'0'+y for x in cat(i-1) for y in cat(n-i)]
    return K
```

14 EXAMEN JULIO 2020

14.1 Pregunta 1.a)

Representa con SageMath a lista **M100_579** dos números enteiros positivos menores ou iguales que 100 que non son divisibles por ningún dos números $5, 7$ e 9.

```
M100_579=[x for x in range(101) if (x%5!=0 and x%7!=0 and x%9!=0)]
M100_579
```
[1, 2, 3, 4, 6, 8, 11, 12, 13, 16, 17, 19, 22, 23, 24, 26, 29, 31, 32, 33, 34, 37, 38, 39,
41, 43, 44, 46, 47, 48, 51, 52, 53, 57, 58, 59, 61, 62, 64, 66, 67, 68, 69, 71, 73, 74,
76, 78, 79, 82, 83, 86, 87, 88, 89, 92, 93, 94, 96, 97]

```
#Otra forma de hacerlo:
M100_579 = [n for n in range(101) if gcd(n,5*7) == 1 and n%9 != 0]
```

```
M100_579
```
[1, 2, 3, 4, 6, 8, 11, 12, 13, 16, 17, 19, 22, 23, 24, 26, 29, 31, 32, 33, 34, 37, 38, 39, 41, 43, 44, 46, 47, 48, 51, 52, 53, 57, 58, 59, 61, 62, 64, 66, 67, 68, 69, 71, 73, 74, 76, 78, 79, 82, 83, 86, 87, 88, 89, 92, 93, 94, 96, 97]

14.2 Pregunta 1.b)

Determina a cantidade **C** de números enteiros positivos menores ou iguales a 100 que non son divisibles por ningún dos números $5, 7$ e 9.

```
len(M100_579)
```
60

14.3 Pregunta 1.c)

Define a función **Dn_579** que collido un númeror *n* nos devolve a lista dos primeiros *n* números enteiros positivos que non son divisibles por ningún dos números $5, 7$ e 9 e calcula o valor de **Dn_579** para algúns números *n* pequenos.

```
def Dn_579(n):
    numeros=[]
    j=0
    while len(numeros)!=n:
        if (j%5!=0 and j%7!=0 and j%9!=0):
            numeros.append(j)
        j+=1
    return numeros

Dn_579(10)
```
[1, 2, 3, 4, 6, 8, 11, 12, 13, 16]

```
#Otra forma de hacerlo:
def Dn_579_v2(n):
    L = []
    x = 1
    for i in range(n):
        while gcd(x,5*7)!= 1 or x%9 == 0:
            x += 1
        L.append(x)
        x += 1
    return L
Dn_579_v2(10)
```
[1, 2, 3, 4, 6, 8, 11, 12, 13, 16]

14.4 Pregunta 1.d)

Utiliza a función **Dn_579** da pregunta **1.a)**, a cantidade **C** da pregunta **1.b)** para comprobar que a lista **M100_579** que obtiveches na pregunta **1.a)** é correcta.

(*Se non conseguiches algún dos datos anteriores, indica como farías para realizar esta comprobación se tiveses todos os datos indicados*)

```
Dn_579(60) == M100_579
```
```
True
```

14.5 Pregunta 2.a)

Define a función **test** que collido un número *k* e unha lista *L* de números enteiros positivos nos devolve **True** se *k* non é divisible por ningún dos números de *L*.

```
def test_v1(k,L):
    tst = True
    for i in L:
        tst = tst and k%i != 0
    return tst
test_v1(5, [2])
test_v1(7,[2,3,4,5,6])
```
```
True
True
```

```
#Otra opción:
def test_v2(k,L):
    for p in L:
        if k%p == 0:
            return False
    return True
test_v2(5, [2])
test_v2(7,[2,3,4,5,6])
```
```
True
True
```

14.6 Pregunta 2.b)

Define a función **Dn_L** que collido un número *n* e unha lista *L* de números enteiros positivos nos devolve os *n* primeiros números enteiros positivos que non son divisibles por ningún dos números de *L*.

```
def Dn_L(n,L):
    sitio = []
    con = 0
    k = 1
    while con < n:
```

```
        if test(k,L):
            sitio.append(k)
            con += 1
        k += 1
    return sitio
Dn_L(20,[5,7,9])
```

[1, 2, 3, 4, 6, 8, 11, 12, 13, 16, 17, 19, 22, 23, 24, 26, 29, 31, 32, 33]

14.7 Pregunta 3.a)

Define mediante un **algoritmo recursivo** a función **M** que aplicada a dous números enteiros non negativos, *a* e *b* devolve o seu máximo común divisor.

```
def M(a,b):
    if b == 0:
        return a
    else:
        return M(b,a%b)
M(34,48) == gcd(34,48)
```

True

14.8 Pregunta 3.b)

Utiliza a propiedade que di que para calquera par de números enteiros non negativos, $(a)e(b), [textmcd(a, b)cdotm$ $a \cdot b,]onde(\text{mcd}(a, b))$ é o máximo común divisor de *a* e *b*, e mcm(*a*, *b*) é o mínimo común múltiplo de *a* e *b*, para definir mediante un **algoritmo recursivo** a función **m** que aplicada a dous números enteiros non negativos, *a* e *b* devolve o seu **mínimo común múltiplo**.

```
def m(a,b):
    if M(a,b) == 1:
        return a*b
    else:
        return M(a,b)*m(a/M(a,b),b/M(a,b))
m(6,15)
```

30

15 EXAMEN XULLO 2015

Pregunta uno

Unha lista dise que é palindrómica se ten lonxitude maior ca 0 e os seus elementos lidos de esquerda a dereita e de dereita a esquerda coinciden.

Exemplos: $[1, 0, 0, 1]$, $[2]$, $[a, b, 5, b, a]$, $[10, 5, 10]$, ...

1. Constrúe con Sage todas as listas palindrómicas de lonxitude menor ou igual ca 4 cuxos elementos son números díxitos (nos exemplos anteriores, nin os elementos da terceira lista nin os da cuarta son díxitos).

2. Cantas hai?

```
#APARTADO 1
# Como tengo que construir todas las listas, pero absolutamente \
   todas, sin excepción de ninguna, lo que hay que hacer es que en \
   un rango de 10 dígitos (desde el cero hasta el nueve) abarcar \
   todas las posibilidades y luego juntarlas. Como máximo, solo habr\
   á cuatro dígitos, por lo tanto sería:
l1 = [[i] for i in range(10)]
l2 = [[i, i] for i in range(10)]
l3 = [[i, j, i] for i in range(10) for j in range(10)]
l4 = [[i, j, j, i] for i in range(10) for j in range(10)]
l = l1 + l2 + l3 + l4
```

```
#APARTADO 2
# Por lo tanto, concluimos que hay 220 listas
len(l)
```

```
220
```

Pregunta 2 (Escolle unha das dúas opcións seguintes)
opción A

1. Define a función **p3p** que colle unha lista L de 3 elementos e devolve a lista palindrómica **p3p**(L) de lonxitude 6 que resulta de engadirlle a L pola dereita os seus elementos en orden contraria: **p3p**([1,2,3]) = [1,2,3,3,2,1]

2. Define a función **palin** que colle unha lista L de n elementos e devolve a lista palindrómica **palin**(L) de lonxitude $2 \times n$ que resulta de engadirlle a L pola dereita os seus elementos en orden inversa.

opción B

1. Define a función **palindro** que dado un número enteiro positivo n nos fabrica todas as listas palindrómicas de lonxitude n formadas con números díxitos.

```
#OPCIÓN A
#APARTADO 1
def p3p(L):
    l = [L[0], L[1], L[2], L[2], L[1], L[0]]
    return l
```

```
p3p([1, 2, 3, 4, 5, 6])
```

```
[1, 2, 3, 3, 2, 1]
```

```
#APARTADO 2
def palin(L):
    lon = len(L)
    l = [L[lon-1-i] for i in range(lon)]
    return L + l

palin([1, 2, 3, 4, 5, 6])
```

[1, 2, 3, 4, 5, 6, 6, 5, 4, 3, 2, 1]

```
#Existe un comando en sage que podemos empregar para facer a inversa\
    dunha lista, polo que teríamos outra posibilidade para facelo:
def palin_2(L):
    m = L[:]
    m.reverse()
    return L + m
```

```
palin_2([1, 2, 3, 4, 5, 6])
```

[1, 2, 3, 4, 5, 6, 6, 5, 4, 3, 2, 1]

```
#OPCIÓN B
def palindro(n):
    listas = Permutations(range(10)*n, n).list()#es la lista formada\
    por TODAS las lista dde longitud n
    return [x for x in listas if vale(x)]
```

Pregunta 3

1. Define a función **nu** que colle unha lista de elementos números díxitos e devolve o número enteiro cuxas cifras, de esquerda a dereita, son os elementos da lista na mesma orde, tendo en conta que os ceros á esquerda nos números non se escriben, salvo que o número sexa 0: **nu**([1,2,3]) = 123, **nu**([0,0,2])= 2, **nu**([0,0,0]) = 0.
2. Define a función **sumcap** que dado un número *n* nos devolve a suma de todos os números capicúa menores ou iguales ca *n*. (Un número dise que é capicúa se as súas cifras lidas de esquerda a dereita e de dereita a esquerda coinciden. Exemplos: $123321, 4, 808, ...$)

```
def nu(L):
    n = len(L)
    numero = 0
    for i in range(n):
        numero += L[i]*10^(n-1-i) #lo que estamos haciendo es que a \
   la variable numero se le va a sumar contantemente dependiendo de \
   la longitud de la lista el valor que está en la posición i \
   multiplicado por las decenas, centenas...
    return numero
```

```
nu([1, 2, 3, 4])
```

1234

```
def sumcap(n):
    lon = n.ndigits() #con este comando vemos a número de díxitos \
   que forman n
    l = []   #creamos unha lista
    for i in [1..lon]:
        l += palindro(i)  #imos engadindo todas las listas \
   palindromicas de lonxitude i
        l =  [x for x in lis_pal if x[0]<>0] #eliminamos aquelas \
   listas que comezan por 0
        capicuas = [nu(x) for x in l if nu(x) <= n] #filtramos \
   aquellas listas que corresponden a núemro
    return sum(capicuas)
```

15.0.1 Pregunta 4

Define de modo recursivo a función **power2** que para cada par de números enteiros non negativos, (a, n), nos devolva a^{2^n}.

Tendo en conta que

$$a^{2^n} = a^{2^{n-1}\times 2} = \left(a^{2^{n-1}}\right)^2$$

teremos que

$$a_n = (a_{n-1})^2$$

```
def power2(a, n):  #cada PAR de enteiros non negativos
    if n == 0:
        return a
    else:
        return power2(a, n-1)^2
```

```
power2(3, 3)
```

```
6561
```

16 EXÁMENES DE OTROS AÑOS

Constrúe con Sage todas as listas de lonxitude 7 formadas con números de unha cifra (listas de 7 díxitos) coa condición de que o producto dos seus elementos sexa 35. Determina cantas hai. O mesmo de antes, pero con producto igual a 30.

```
X=[p for p in Permutations('1111157')]
len(X)
```

```
42
```

```
L=[]
for p in Permutations('1111235'):
    L.append(''.join(p))
for p in Permutations('1111165'):
    L.append(''.join(p))

len(L)
```
252

Define unha función que tome u n par de números (b,c) e devolva todas as listas de 7 díxitos cuxo producto sexa (5b̂)*(7ĉ)

```
def listas75(b,c):
    if (b+c)>7:
        return []
    L=[]

    for p in Permutations(b*'5'+c*'7'+(7-b-c)*'1'):
        L.append(''.join(p))
    return L
listas75(1, 1)
len(listas75(1,1))
```
['5711111', '5171111', '5117111', '5111711', '5111171', '5111117', '7511111', '7151111', '7115111', '7111511', '7111151', '7111115', '1571111', '1517111', '1511711', '1511171', '1511117', '1751111', '1715111', '1711511', '1711151', '1711115', '1157111', '1151711', '1151171', '1151117', '1175111', '1171511', '1171151', '1171115', '1115711', '1115171', '1115117', '1117511', '1117151', '1117115', '1111571', '1111517', '1111751', '1111715', '1111157', '1111175']
42

Para dous números k e n, define unha función que nos devolva todas as listas de lonxitude k formadas con díxitos cuxo producto é n, sabendo que os únicos primos que dividen a n son 5 e 7.

```
def listillas75(k,n):
    if 7^k<n:
        return []
    w=n
    cinco=0
    siete=0
    while w>1:
        if w%5==0:
            cinco+=1
            w=w/5
        if w%7==0:
            siete+=1
            w=w/7
    L=[]
    for p in Permutations(cinco*'5'+siete*'7'+(k-cinco-siete)*'1',k)\
```

```
    :
        L.append(''.join(p))
    return L
listillas75(3,175)
```

```
['557', '575', '755']
```

Define outra función como a de antes pero se os únicos primos que dividen a n son 3, 5 e 7.

```
def listillas753(k,n):
    if 7^k<n:
        return []
    w=n
    tres=0
    cinco=0
    siete=0
    while w>1:
        if w%3==0:
            tres+=1
            w=w/3
        if w%5==0:
            cinco+=1
            w=w/5
        if w%7==0:
            siete+=1
            w=w/7
    L=[]
    for i in [0..tres//2]:
        for j in [0..tres]:
            if (2*i+j==tres):
                for p in Permutations(j*'3'+cinco*'5'+siete*'7'+i*'9'\
  '+(k-cinco-siete-i-j)*'1',k):
                    L.append(''.join(p))
    return L
len(listillas753(5,945))
```

```
140
```

Define de modo recursivo a función bin que para cada par de números enteiros non negativos, (n,k), nos devolva 0, se n

```
def bin(n,k):
    if n<k:
        return 0
    elif n==k: return 1
    else:
        return (n/(n-k))*bin(n-1,k)
bin(14, 7)
bin(14,7)==binomial(14,7)
```

```
3432
```

True

Constrúe con Sage todas as listas palindrómicas de lonxitude menor ou igual ca 4 cuxos elementos son números díxitos (nos exemplos anteriores, nin os elementos da terceira lista nin os da cuarta son díxitos). Cantas hai?

```
L=[]
for i in [0..9]:
    L.append(i)
    L.append([i,i])
    for j in [0..9]:
        L.append([i,j,i])
        L.append([i,j,j,i])
len(L)
```

220

Define a función pal_6 que colle unha lista L de 3 elementos e devolve a lista palindrómica pal_6(L) de lonxitude 6 que resulta de engadirlle a L pola dereita os seus elementos en orden contraria: pal_6([1,2,3]) = [1,2,3,3,2,1]

```
def pal_6(L):
    pal6=[L[0],L[1],L[2],L[2],L[1],L[0]]
    return pal6
pal_6([1,2,9])
```

[1, 2, 9, 9, 2, 1]

Define a función pal_2n que colle unha lista L de nn elementos e devolve a lista palindrómica pal_2n(L) de lonxitude $2 \times n2 \times n$ que resulta de engadirlle a L pola dereita os seus elementos en orden inversa.

```
def pal_2n(L):
    pal2n=[]
    w=len(L)
    for i in [0..len(L)-1]:
        pal2n.append(L[i])
    for i in [0..len(L)-1]:
        pal2n.append(L[(len(L)-1)-i])
    return pal2n
pal_2n([1,2,3,4,7,3,4])
```

[1, 2, 3, 4, 7, 3, 4, 4, 3, 7, 4, 3, 2, 1]

Define a función pal_x que dado un número enteiro positivo nn nos fabrica todas as listas palindrómicas de lonxitude nn formadas con números díxitos.

```
def pal_x(n):
    L=[]
    contador=0
    for i in Permutations('0123456789'*n,n):
```

```
        contador=0
        for j in range(n//2):
            if i[j]==i[(n-1)-j]:
                contador+=1
        if contador==(n//2):
            L.append(''.join(i))
    return L

len(pal_x(3))
```
100

Define a función de_list_a_num que colle unha lista de elementos números díxitos e devolve o número enteiro cuxas cifras, de esquerda a dereita, son os elementos da lista na mesma orde, tendo en conta que os ceros á esquerda nos números non se escriben, salvo que o número sexa 0: de_list_a_num([1,2,3]) = 123, de_list_a_num([0,0,2])= 2, de_list_a_num([0,0,0]) = 0.

```
def de_list_a_num(L):
    num=0
    for i in range(len(L)):
        num+=L[i]*(10^((len(L)-1)-i))
    return num
de_list_a_num([1, 0, 0])
```
100

Define a función sum_d_cap que dado un número nn nos devolve a suma de todos os números capicúa menores ou iguales ca nn. (Un número dise que é capicúa se as súas cifras lidas de esquerda a dereita e de dereita a esquerda coinciden. Exemplos: 123321, 4, 808, \dots123321,4,808,...)

```
def cifras(n):
    cifras=0
    while n>=1:
        n=n/10
        cifras+=1
    return cifras

def sum_d_cap(n):
    suma=0
    for i in [0..n]:
        x=cifras(i)
        w=i
        L=[]
        for z in [0..x-1]:
            L.append(w//(10^((x-1)-z)))
            w=w-(L[z]*10^((x-1)-z))
        q=len(L)//2
        contador=0
        for k in [0..q-1]:
            if L[k]==L[(len(L)-1)-k]:
```

```
                contador+=1
        if contador==q:
            suma+=i
    return suma
```

```
sum_d_cap(1000)
```

```
50040
```

FIBONACCI

```
#Fibonacci
def F(n):
    if n==0:
        return 0
    elif n==1:
        return 1
    else:
        return F(n-1)+F(n-2)
```

```
F(7)
```

```
13
```

Número de cadenas con tres unos consecutivos

```
def C(n):
    if n==1:
        return 0
    elif n==2:
        return 0
    elif n==3:
        return 1
    else:
        return C(n-1)+C(n-2)+C(n-3)+2^(n-3)
```

```
C(6)
```

```
20
```

Cadenas de longitud n que empiezan por 11

```
def cadenas11(n):
    L=['11'+''.join(i) for i in Permutations('01'*(n-2),n-2)]
    return L
cadenas11(4)
```

```
['1100', '1101', '1110', '1111']
```

Cadenas de n bits sin dos o mas ceros consecutivos

```
def cadenas00(n):
    total=[''.join(w) for w in Permutations('01'*n,n)]
```

```
    filtro=[c for c in total if not '00' in c]
    return filtro
cadenas00(4)
```

```
['0101', '0110', '0111', '1010', '1011', '1101', '1110', '1111']
```