Clases - Semana 1

September 24, 2020

1 Introduction to the Course

2 Python Functions

[1]:
```
x = 1
y = 2
x + y
```

[1]:
```
3
```

[2]:
```
x
```

[2]:
```
1
```

[3]:
```
def add_numbers(x, y):
    return x + y
add_numbers(1, 2)
```

[3]:
```
3
```

[4]:
```
def add_numbers(x, y, z = None):
    if (z == None):
        return x + y
    else:
        return x + y + z
print(add_numbers(1, 2))
print(add_numbers(1, 2, 3))
```

```
3
6
```

[5]:
```
def add_numbers(x, y, z = None, flag = False):
    if (flag == True):
        print("Flag is true!")
    if (z == None):
        return x + y
    else:
        return x + y + z
print(add_numbers(1, 2, flag = True))
```

```
Flag is true!
3
```

3 Python Types and Sequences

[6]:
```
type('This is a string')
```

[6]:
```
str
```

[7]:
```
type(None)
```

[7]:
```
NoneType
```

[8]:
```
type(1)
```

[8]:
```
int
```

[9]:
```
type(1.0)
```

[9]:
```
float
```

[10]:
```
type(add_numbers)
```

[10]:
```
function
```

- Tuple: Una tupla es una secuencia de variables. La tupla es inmutable. Esto significa que una tupla tiene elementos ordenados, pero no se pueden modificar una una vez creados. Escribimos tuplas utilizando paréntesis, y podemos mezclar tipos en los contenidos de las tuplas. Aquí hay una tupla que tiene cuatro elementos. Dos son números y dos son cadenas.

[11]:
```
x = (1, 'a', 2, 'b')
type(x)
```

[11]:
```
tuple
```

- Lista: Una lista se declara usando corchetes. Hay un par de maneras diferentes de cambiar el contenido de una lista. Uno es a través de la función "append" que te permite añadir nuevos elementos al final de la lista.

[12]:
```
x = [1, 'a', 2, 'b']
type(x)
```

[12]:
```
list
```

[13]:
```
x.append(3.3)
print(x)
```

```
[1, 'a', 2, 'b', 3.3]
```

[14]:
```
for item in x:
    print(item)
```

```
1
a
2
b
3.3
```

[15]:
```
i = 0
while ( i != len(x)):
    print(x[i])
    i = i + 1
```

```
1
a
2
b
3.3
```

[16]:
```
[1, 2] + [3, 4] #el + concatena valores de la lista
```

[16]:
```
[1, 2, 3, 4]
```

[17]:
```
[1] * 3 # el * repiten los valores de la lista
```

[17]:
```
[1, 1, 1]
```

[18]:
```
1 in [1, 2, 3] #devuelve true o false
```

[18]:
```
True
```

- Slicing: Mientras que la sintaxis de corchetes para acceder a un elemento puede parecer bastante similar a la que has visto en otros lenguajes, En Python, el operador de indexación admite varios valores. El primer parámetro es, la ubicación inicial, Si éste elemento es único entonces se retorna un elemento de la lista. El segundo parámetro indica el final de la rebanada, y es un final exclusivo. Así que si rebanaste con el primer parámetro siendo cero… y el segundo parámetro siendo uno, entonces consigues solamente un item.

[19]:
```
x = 'This is a string'
print(x[0])
print(x[0:1])
print(x[0:2])
```

```
T
T
Th
```

[20]:
```
x[-1] #tambien se pueden usar indices negativos
```

[20]:
```
'g'
```

[21]:
```
x[-4 : -2]
```

[21]:
```
'ri'
```

[22]:
```
x[:3] #el comienzo esta implicito al no poner nada antes del :
```

[22]:
```
'Thi'
```

[23]:
```
x[3:] #el final esta implicito al no poner nada antes del :
```

[23]:
```
's is a string'
```

- Evaluacion de expresiones regulares:

[24]:
```
firstname = 'Christopher'
lastname = 'Brooks'

print(firstname + ' ' + lastname)
print(firstname * 3)
print('Chris' in firstname)
```

```
Christopher Brooks
ChristopherChristopherChristopher
True
```

[25]:
```
firstname = 'Christopher Arthur Hansen Brooks'.split(' ')[0] #con el 0 elijo el␣
 ↪primer elemento
lastname = 'Christopher Arthur Hansen Brooks'.split(' ')[-1]
print(firstname)
print(lastname)
```

```
Christopher
Brooks
```

- Dicts: Los diccionarios se asimilan a las listas y tuplas por el hecho de que contienen una colección de elementos, pero se trata de colecciones etiquetadas que no tienen un orden. Esto significa que por cada valor que insertas en el diccionario, debes también proveer una clave de acceso a dicho valor. En otros lenguajes a esta estructura se la suele llamar "mapa". Y en Python usamos llaves para denotar un diccionario. Aquí tenemos un ejemplo donde vinculamos nombres a direcciones de correo electrónico. Puedes ver que para crear cada elemento del diccionario utilizamos pares de valores separados por "dos puntos". Luego puedes recuperar un valor para una etiqueta dada usando el operador de indexación.

[26]:
```
x = {'Christopher Brooks': 'brooksch@umich.edu', 'Bill Gates': 'billg@microsoft.
 ↪com'}
x['Christopher Brooks']
```

[26]:
```
'brooksch@umich.edu'
```

[27]:
```
x['Kevin Collins-Thompson'] = None #agrego valor al diccionario
x['Kevin Collins-Thompson']
```

[28]:
```
x
```

[28]:
```
{'Christopher Brooks': 'brooksch@umich.edu',
 'Bill Gates': 'billg@microsoft.com',
 'Kevin Collins-Thompson': None}
```

[29]:
```
for name in x:
    print(name)
    print(x[name])
#itero a traves de las claves del diccionario
```

```
Christopher Brooks
brooksch@umich.edu
Bill Gates
billg@microsoft.com
Kevin Collins-Thompson
None
```

[30]:
```
for email in x.values():
    print(email)
#itero a traves de los valores del diccionario
```

```
brooksch@umich.edu
billg@microsoft.com
None
```

[31]:
```
for name, email in x.items():
    print(name)
    print(email)
#itero a traves de los valores y las claves al mismo tiempo
```

```
Christopher Brooks
brooksch@umich.edu
Bill Gates
billg@microsoft.com
Kevin Collins-Thompson
None
```

[32]:
```
x = ('Christopher', 'Brooks', 'brooksch@michigan.edu') #declaro 3 variables en␣
 ↪la tupla, si tuviera 4 tiraria error la siguiente linea
fname, lname, email = x
```

[33]:
```
fname
```

[33]:
```
'Christopher'
```

[34]:
```
lname
```

[34]:
```
'Brooks'
```

[35]:
```
email
```

[35]:
```
'brooksch@michigan.edu'
```

4 Python More on Strings

[36]:
```
print('Chris' + str(2)) #lo tengo que convertir a string, sino tira error
```

```
Chris2
```

[37]:
```
sales_record = {'price': 3.24,
                'num_items': 4,
                'person': 'Chris'}
sales_statement = '{} bought {} item(s) at a price of {} each for a total of␣
 ↪{}' #pongo las llaves para luego aplicar .format
print(sales_statement.format(sales_record['person'],
                             sales_record['num_items'],
                             sales_record['price'],
                             sales_record['num_items'] * sales_record['price']))
```

```
Chris bought 4 item(s) at a price of 3.24 each for a total of 12.96
```

5 Pytho Demostration: Reading and Writing CSV files

Data Files and Summary statistics

[38]:
```
import csv

%precision 2

with open ('mpg.csv') as csvfile:
    mpg = list(csv.DictReader(csvfile))     #necesito que el csv esté en la␣
 ↪misma carpeta donde estoy trabajando para abrirlo asi
mpg[:3]                                     #estoy leyendo el csv como una lista␣
 ↪de diccionarios
```

[38]:
```
[OrderedDict([('', '1'),
              ('manufacturer', 'audi'),
              ('model', 'a4'),
              ('displ', '1.8'),
              ('year', '1999'),
              ('cyl', '4'),
              ('trans', 'auto(l5)'),
              ('drv', 'f'),
              ('cty', '18'),
              ('hwy', '29'),
              ('fl', 'p'),
              ('class', 'compact')]),
 OrderedDict([('', '2'),
              ('manufacturer', 'audi'),
              ('model', 'a4'),
              ('displ', '1.8'),
              ('year', '1999'),
              ('cyl', '4'),
              ('trans', 'manual(m5)'),
              ('drv', 'f'),
              ('cty', '21'),
              ('hwy', '29'),
              ('fl', 'p'),
              ('class', 'compact')]),
 OrderedDict([('', '3'),
              ('manufacturer', 'audi'),
              ('model', 'a4'),
              ('displ', '2'),
              ('year', '2008'),
              ('cyl', '4'),
              ('trans', 'manual(m6)'),
              ('drv', 'f'),
              ('cty', '20'),
              ('hwy', '31'),
              ('fl', 'p'),
              ('class', 'compact')])]
```

[39]:
```
len(mpg) #significa que tengo 234 diccionarios, uno para cada uno de los␣
 ↪vehiculos
```

[39]:
```
234
```

[40]:
```
mpg[0].keys() #veo los nombres de las columnas del csv con el metodo .keys
```

[40]:
```
odict_keys(['', 'manufacturer', 'model', 'displ', 'year', 'cyl', 'trans', 'drv',
'cty', 'hwy', 'fl', 'class'])
```

- Calculo el promedio de consumo para ciudad y autopista:

[41]:
```
sum(float(d['cty']) for d in mpg) / len(mpg) #con float lo convierto en numero␣
 ↪decimal (porque todos los datos son strings), con el for voy recorriendo los␣
 ↪valores de consumo en ciudad en el csv
#saco el promedio de consumo en ciudad, sumo todos los consumos y divido por la␣
 ↪cantidad
```

[41]:
```
16.86
```

[42]:
```
sum(float(d['hwy']) for d in mpg) / len(mpg) #promedio de consumo en autopista␣
 ↪(highyway), tiene sentido que sea mayo
```

[42]:
```
23.44
```

- Ahora vamos a calcular el promedio de consumo en ciudad agrupado por la cantidad de cilindros del auto:

[43]:
```
cylinders = set(d['cyl'] for d in mpg) #saco los valores unicos de cantidad de␣
 ↪cilindros
cylinders
```

[43]:
```
{'4', '5', '6', '8'}
```

[44]:
```
CtyMpgByCyl = [] #creo una lista vacia para almacenar los calculos

for c in cylinders:                      #con este for recorro los tipos de␣
 ↪cilindros (4, 5, 6 y 8)
    summpg = 0                           #para ir sumando los consumos del mismo␣
 ↪tipo
    cyltypecount = 0                     #para ir contando la cantidad del mismo␣
 ↪tipo
    for d in mpg:                        #con este for recorro mpg, si es el tipo␣
 ↪de cilindro que me interesa actualizo suma y contador
        if d['cyl'] == c:
            summpg += float(d['cty'])
            cyltypecount += 1
    CtyMpgByCyl.append((c, summpg / cyltypecount)) #mientras vaya terminando␣
 ↪con los distintos tipos de cilindros va a calcular y agregar a la lista
                                                   #se pone el tipo de cilindro␣
 ↪y el promedio de consumo en ciudad en la lista
CtyMpgByCyl.sort(key = lambda x: x[0])  #ordena la lista por consumo de mayor a␣
 ↪menor
CtyMpgByCyl
```

[44]:
```
[('4', 21.01), ('5', 20.50), ('6', 16.22), ('8', 12.57)]
```

- Ahora calculamos el promedio de consumo en autopista para los distintos tipos de vehiculos:

[45]:
```
vehicleclass = set(d['class'] for d in mpg)
vehicleclass
```

[45]:
```
{'2seater', 'compact', 'midsize', 'minivan', 'pickup', 'subcompact', 'suv'}
```

[46]:
```
HwyMpgByClass = []

for t in vehicleclass:
    summpg = 0
    count = 0
    for d in mpg:
        if d['class'] == t:
            summpg += float(d['hwy'])
            count += 1
    HwyMpgByClass.append((t, summpg / count))
HwyMpgByClass.sort(key = lambda x: x[1]) #ordeno los consumos por clase de␣
↪menor a mayor
HwyMpgByClass
```

[46]:
```
[('pickup', 16.88),
 ('suv', 18.13),
 ('minivan', 22.36),
 ('2seater', 24.80),
 ('midsize', 27.29),
 ('subcompact', 28.14),
 ('compact', 28.30)]
```

6 Python Dates and Times

[47]:
```
import datetime as dt
import time as tm
```

[48]:
```
tm.time() #el tiempo esta contado a partir de una fecha determinada, por eso␣
↪hay que convertirlo
```

[48]:
```
1600654537.17
```

[49]:
```
dtnow = dt.datetime.fromtimestamp(tm.time()) #aca convierto el tiempo al valor␣
↪del momento actual
dtnow
#veo año, mes, dia y hora. La hora esta en otro uso con 3 horas mas a la␣
↪nuestra
```

[49]:
```
datetime.datetime(2020, 9, 21, 2, 15, 37, 174218)
```

[50]:
```
dtnow.year, dtnow.month, dtnow.day, dtnow.hour, dtnow.minute, dtnow.second
```

[50]:
```
(2020, 9, 21, 2, 15, 37)
```

[51]:
```
delta = dt.timedelta(days = 100) #creo un delta de tiempo de 100 dias
delta
```

[51]:
```
datetime.timedelta(days=100)
```

[52]:
```
today = dt.date.today()  #fecha de hoy
today
```

[52]:
```
datetime.date(2020, 9, 21)
```

[53]:
```
today - delta #calculo que dia fue hace 100 dias
```

[53]:
```
datetime.date(2020, 6, 13)
```

[54]:
```
today > today - delta
```

[54]:
```
True
```

7 Advanced Python Objects, map()

[55]:
```
class Person:                           #defino la clase, todo lo de abajo␣
↪esta dentro de la misma
    department = 'School of Information'

    def set_name(self, new_name):          #defino un metodo, se escribe␣
↪parecido a una funcion (hay que agregar self)
        self.name = new_name
    def set_location(self, new_location):
        self.location = new_location
```

[56]:
```
person = Person()
person.set_name('Christopher Brooks')
person.set_location('Ann Arbor, MI, USA')
print('{} live in {} and works in the department {}'.format(person.name, person.
↪location, person.department))
```

```
Christopher Brooks live in Ann Arbor, MI, USA and works in the department School
of Information
```

- Imaginemos que tenemos dos listas de números, Quizás los precios de dos tiendas diferentes con exactamente los mismos artículos. Y queremos encontrar el mínimo que tendríamos que pagar Si compráramos el artículo más barato entre las dos tiendas. Para hacer esto, podríamos iterar cada lista, comparando artículos y eligiendo el más barato. Con "map", podemos hacer esta comparación en una sola sentencia.

[57]:
```
store1 = [10.00, 11.00, 12.34, 2.34]
store2 = [9.00, 11.10, 12.34, 2.01]
cheapest= map(min, store1, store2)
cheapest
```

[57]:
```
<map at 0x7f46eb5a2828>
```

[58]:
```
for item in cheapest:
    print(item)
```

```
9.0
11.0
12.34
2.01
```

- Ejercicio: Here is a list of faculty teaching this MOOC. Can you write a function and apply it using map() to get a list of all faculty titles and last names (e.g. ['Dr. Brooks', 'Dr. Collins-Thompson', . . .]) ?

[59]:
```
people = ['Dr. Christopher Brooks', 'Dr. Kevyn Collins-Thompson', 'Dr. VG Vinod␣
↪Vydiswaran', 'Dr. Daniel Romero']
```

```
def split_title_and_name(person): #defino la funcion
    title = person.split()[0]      #obtengo el titulo de la person
    lastname = person.split()[-1] #obtengo el apellido de la persona
    return '{} {}'.format(title, lastname) #retorno titulo y nombre

list(map(split_title_and_name, people)) #en el map pongo la funcion a usar y la␣
 ↪lista de valores
```

[59]:
```
['Dr. Brooks', 'Dr. Collins-Thompson', 'Dr. Vydiswaran', 'Dr. Romero']
```

8 Advanced Python Lambda and List Comprehensions

"Lambda" es la manera que tiene Python para crear funciones anónimas. Son similares a otras funciones, pero no tienen nombre. La intención es que sean simples o de corta duración, y fáciles de escribirlas en una sola línea en vez de tener que preocuparse de crear una función con nombre.

[60]:
```
my_function = lambda a, b, c: a + b #lambda devuelve una sola expresion, en␣
 ↪este caso la suma de a y b. Los parametros son a, b y c
```

[61]:
```
my_function(1, 2, 3) #llamo a la funcion lambda
```

[61]:
```
3
```

- Ejercicio: convert this function into a lambda:

[62]:
```
people = ['Dr. Christopher Brooks', 'Dr. Kevyn Collins-Thompson', 'Dr. VG Vinod␣
 ↪Vydiswaran', 'Dr. Daniel Romero']

def split_title_and_name(person):
    return person.split()[0] + ' ' + person.split()[-1]

#option 1
for person in people:
    print(split_title_and_name(person) == (lambda x: x.split()[0] + ' ' + x.
 ↪split()[-1])(person))

#option 2
list(map(split_title_and_name, people)) == list(map(lambda person: person.
 ↪split()[0] + ' ' + person.split()[-1], people))
```

```
True
True
True
True
```

[62]:
```
True
```

[63]:
```
my_list = []
for number in range(0, 1000):
    if number % 2 == 0:      #veo si la division entera del numero por 2 tiene␣
 ↪resto cero
        my_list.append(number)
my_list
```

[63]:
```
[0,
 2,
 4,
 6,
 8,
 10,
 12,
 14,
 16,
 18,
 20,
 22,
 24,
 26,
 28,
 30,
 32,
 34,
 36,
 38,
 40,
 42,
 44,
 46,
 48,
 50,
 52,
 54,
 56,
 58,
 60,
 62,
 64,
 66,
 68,
 70,
 72,
 74,
 76,
 78,
80,
82,
84,
86,
88,
90,
92,
94,
96,
98,
100,
102,
104,
106,
108,
110,
112,
114,
116,
118,
120,
122,
124,
126,
128,
130,
132,
134,
136,
138,
140,
142,
144,
146,
148,
150,
152,
154,
156,
158,
160,
162,
164,
166,
168,
170,
172,
174,
176,
178,
180,
182,
184,
186,
188,
190,
192,
194,
196,
198,
200,
202,
204,
206,
208,
210,
212,
214,
216,
218,
220,
222,
224,
226,
228,
230,
232,
234,
236,
238,
240,
242,
244,
246,
248,
250,
252,
254,
256,
258,
260,
262,
264,
266,
268,
270,
272,
274,
276,
278,
280,
282,
284,
286,
288,
290,
292,
294,
296,
298,
300,
302,
304,
306,
308,
310,
312,
314,
316,
318,
320,
322,
324,
326,
328,
330,
332,
334,
336,
338,
340,
342,
344,
346,
348,
350,
352,
354,
356,
358,
360,
362,
364,
366,
368,
370,
372,
374,
376,
378,
380,
382,
384,
386,
388,
390,
392,
394,
396,
398,
400,
402,
404,
406,
408,
410,
412,
414,
416,
418,
420,
422,
424,
426,
428,
430,
432,
434,
436,
438,
440,
442,
444,
446,
448,
450,
452,
454,
456,
458,
460,
462,
464,
466,
468,
470,
472,
474,
476,
478,
480,
482,
484,
486,
488,
490,
492,
494,
496,
498,
500,
502,
504,
506,
508,
510,
512,
514,
516,
518,
520,
522,
524,
526,
528,
530,
532,
534,
536,
538,
540,
542,
544,
546,
548,
550,
552,
554,
556,
558,
560,
562,
564,
566,
568,
570,
572,
574,
576,
578,
580,
582,
584,
586,
588,
590,
592,
594,
596,
598,
600,
602,
604,
606,
608,
610,
612,
614,
616,
618,
620,
622,
624,
626,
628,
630,
632,
634,
636,
638,
640,
642,
644,
646,
648,
650,
652,
654,
656,
658,
660,
662,
664,
666,
668,
670,
672,
674,
676,
678,
680,
682,
684,
686,
688,
690,
692,
694,
696,
698,
700,
702,
704,
706,
708,
710,
712,
714,
716,
718,
720,
722,
724,
726,
728,
730,
732,
734,
736,
738,
740,
742,
744,
746,
748,
750,
752,
754,
756,
758,
760,
762,
764,
766,
768,
770,
772,
774,
776,
778,
780,
782,
784,
786,
788,
790,
792,
794,
796,
798,
800,
802,
804,
806,
808,
810,
812,
814,
816,
818,
820,
822,
824,
826,
828,
830,
832,
834,
836,
838,
840,
842,
844,
846,
848,
850,
852,
854,
856,
858,
860,
862,
864,
866,
868,
870,
872,
874,
876,
878,
880,
882,
884,
886,
888,
890,
892,
894,
896,
898,
900,
902,
904,
906,
908,
910,
912,
914,
916,
918,
920,
922,
924,
 926,
 928,
 930,
 932,
 934,
 936,
 938,
 940,
 942,
 944,
 946,
 948,
 950,
 952,
 954,
 956,
 958,
 960,
 962,
 964,
 966,
 968,
 970,
 972,
 974,
 976,
 978,
 980,
 982,
 984,
 986,
 988,
 990,
 992,
 994,
 996,
 998]
```

[64]:
```
my_list = [number for number in range(0, 1000) if number % 2 == 0] #escrito␣
↪como lista de comprension
my_list #primero pongo lo que quiero poner en la lista (number), luego el for y␣
↪luego la condicion
```

[64]:
```
[0,
 2,
 4,
 6,
 8,
10,
12,
14,
16,
18,
20,
22,
24,
26,
28,
30,
32,
34,
36,
38,
40,
42,
44,
46,
48,
50,
52,
54,
56,
58,
60,
62,
64,
66,
68,
70,
72,
74,
76,
78,
80,
82,
84,
86,
88,
90,
92,
94,
96,
98,
100,
102,
104,
106,
108,
110,
112,
114,
116,
118,
120,
122,
124,
126,
128,
130,
132,
134,
136,
138,
140,
142,
144,
146,
148,
150,
152,
154,
156,
158,
160,
162,
164,
166,
168,
170,
172,
174,
176,
178,
180,
182,
184,
186,
188,
190,
192,
194,
196,
198,
200,
202,
204,
206,
208,
210,
212,
214,
216,
218,
220,
222,
224,
226,
228,
230,
232,
234,
236,
238,
240,
242,
244,
246,
248,
250,
252,
254,
256,
258,
260,
262,
264,
266,
268,
270,
272,
274,
276,
278,
280,
282,
284,
286,
288,
290,
292,
294,
296,
298,
300,
302,
304,
306,
308,
310,
312,
314,
316,
318,
320,
322,
324,
326,
328,
330,
332,
334,
336,
338,
340,
342,
344,
346,
348,
350,
352,
354,
356,
358,
360,
362,
364,
366,
368,
370,
372,
374,
376,
378,
380,
382,
384,
386,
388,
390,
392,
394,
396,
398,
400,
402,
404,
406,
408,
410,
412,
414,
416,
418,
420,
422,
424,
426,
428,
430,
432,
434,
436,
438,
440,
442,
444,
446,
448,
450,
452,
454,
456,
458,
460,
462,
464,
466,
468,
470,
472,
474,
476,
478,
480,
482,
484,
486,
488,
490,
492,
494,
496,
498,
500,
502,
504,
506,
508,
510,
512,
514,
516,
518,
520,
522,
524,
526,
528,
530,
532,
534,
536,
538,
540,
542,
544,
546,
548,
550,
552,
554,
556,
558,
560,
562,
564,
566,
568,
570,
572,
574,
576,
578,
580,
582,
584,
586,
588,
590,
592,
594,
596,
598,
600,
602,
604,
606,
608,
610,
612,
614,
616,
618,
620,
622,
624,
626,
628,
630,
632,
634,
636,
638,
640,
642,
644,
646,
648,
650,
652,
654,
656,
658,
660,
662,
664,
666,
668,
670,
672,
674,
676,
678,
680,
682,
684,
686,
688,
690,
692,
694,
696,
698,
700,
702,
704,
706,
708,
710,
712,
714,
716,
718,
720,
722,
724,
726,
728,
730,
732,
734,
736,
738,
740,
742,
744,
746,
748,
750,
752,
754,
756,
758,
760,
762,
764,
766,
768,
770,
772,
774,
776,
778,
780,
782,
784,
786,
788,
790,
792,
794,
796,
798,
800,
802,
804,
806,
808,
810,
812,
814,
816,
818,
820,
822,
824,
826,
828,
830,
832,
834,
836,
838,
840,
842,
844,
846,
848,
850,
852,
854,
856,
858,
860,
862,
864,
866,
868,
870,
872,
874,
876,
878,
880,
882,
884,
886,
888,
890,
892,
894,
896,
898,
900,
902,
904,
906,
908,
910,
912,
914,
916,
918,
920,
922,
924,
926,
928,
930,
932,
934,
936,
938,
940,
942,
944,
946,
948,
 950,
 952,
 954,
 956,
 958,
 960,
 962,
 964,
 966,
 968,
 970,
 972,
 974,
 976,
 978,
 980,
 982,
 984,
 986,
 988,
 990,
 992,
 994,
 996,
 998]
```

- Ejercicio: Here, why don't you try converting a function into a list comprehension.

[65]:
```
def times_tables():
    lst = []
    for i in range(10):
        for j in range (10):
            lst.append(i*j)
    return lst

times_tables() == [j*i for i in range(10) for j in range(10)]
```

[65]:
```
True
```

- Ejercicio: Here's a harder question which brings a few things together.

Many organizations have user ids which are constrained in some way. Imagine you work at an internet service provider and the user ids are all two letters followed by two numbers (e.g. aa49). Your task at such an organization might be to hold a record on the billing activity for each possible user.

Write an initialization line as a single list comprehension which creates a list of all possible user ids. Assume the letters are all lower case.

```
[66]: lowercase = 'abcdefghijklmnopqrstuvwxyz'
      digits = '0123456789'

      correct_answer = [a+b+c+d for a in lowercase for b in lowercase for c in digits␣
       ↪for d in digits]

      correct_answer[:50] # Display first 50 ids
```

```
[66]: ['aa00',
       'aa01',
       'aa02',
       'aa03',
       'aa04',
       'aa05',
       'aa06',
       'aa07',
       'aa08',
       'aa09',
       'aa10',
       'aa11',
       'aa12',
       'aa13',
       'aa14',
       'aa15',
       'aa16',
       'aa17',
       'aa18',
       'aa19',
       'aa20',
       'aa21',
       'aa22',
       'aa23',
       'aa24',
       'aa25',
       'aa26',
       'aa27',
       'aa28',
       'aa29',
       'aa30',
       'aa31',
       'aa32',
       'aa33',
       'aa34',
       'aa35',
       'aa36',
       'aa37',
       'aa38',
       'aa39',
       'aa40',
       'aa41',
       'aa42',
       'aa43',
       'aa44',
       'aa45',
       'aa46',
       'aa47',
       'aa48',
       'aa49']
```

9 Fundamentals of Data Manipulation

10 Numerical Python Library (NumPy)

[2]:
```
import numpy as np
import math
```

11 - Array creation

[68]:
```
a = np.array([1, 2, 3]) #creo matriz introduciendo una lista
print(a)
print(a.ndim) #dimension de a (cantidad de filas)
```

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

[69]:
```
b = np.array([[1, 2, 3], [4, 5, 6]]) #creo matriz introduciendo lista de listas
b
```

[69]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

[70]:
```
b.shape #veo las dimensiones de b (filas, columnas)
```

[70]:
```
(2, 3)
```

[71]:
```
a.dtype #veo el tipo de los items de la matriz
```

[71]:
```
dtype('int64')
```

[72]:
```
a.dtype.name #veo solo el contenido poniendo .name
```

[72]:
```
'int64'
```

En las matrices el tipo de dato es igual para todos los valores, es decir si yo pongo un numero decimal en una matriz, todos los numeros van a ser float, aunque sean enteros.

[73]:
```
d = np.zeros((2, 3)) #genero matriz de todos ceros indicandole las dimensiones
print(d)

e = np.ones((2,3))   #genero matriz de todos unos indicandole las dimensiones
print(e)
```

```
[[0. 0. 0.]
 [0. 0. 0.]]
[[1. 1. 1.]
 [1. 1. 1.]]
```

[74]:
```
np.random.rand(2, 3)  #genero matriz con numeros al azar indicandole las␣
 ↪dimensiones
```

[74]:
```
array([[0.2231077 , 0.09340128, 0.25987488],
       [0.79496412, 0.4243058 , 0.72574239]])
```

[75]:
```
f = np.arange(10, 50, 2) #similar al range, el ultimo valor es el salto entre␣
 ↪valor y valor
f
```

[75]:
```
array([10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42,
       44, 46, 48])
```

[76]:
```
np.linspace(0, 2, 15) #similar al arange pero para floats. El ultimo numero es␣
 ↪la cantidad de numeros que quiero generar entros los calores dados␣
 ↪(inclusive ambos)
```

[76]:
```
array([0.        , 0.14285714, 0.28571429, 0.42857143, 0.57142857,
       0.71428571, 0.85714286, 1.        , 1.14285714, 1.28571429,
       1.42857143, 1.57142857, 1.71428571, 1.85714286, 2.        ])
```

12 - Array operations

[77]:
```
a = np.array([10, 20, 30, 40])
b = np.array([1, 2, 3, 4])

c = a - b
print(c)

d = a * b
print(d)
```

```
[ 9 18 27 36]
[ 10  40  90 160]
```

[78]:
```
#Ejemplo para pasar valores en řF a řC
farenheit = np.array([0, -10, -5, -15])
```

```
celsius = (farenheit - 31) * (5/9)
print(celsius)
```

```
[-17.22222222 -22.77777778 -20.          -25.55555556]
```

[79]:
```
celsius > -20 #devuelve un boolean para cada valor
```

[79]:
```
array([ True, False, False, False])
```

[80]:
```
celsius % 2 == 0
```

[80]:
```
array([False, False,  True, False])
```

[81]:
```
A = np.array([[1, 1], [0, 1]])
B = np.array([[2, 0], [3, 4]])
print(A*B) #con el * multiplico cada valor con su respectivo de la otra matriz
print(A@B) #el @ se usa para multiplicar matrices
```

```
[[2 0]
 [0 4]]
[[5 4]
 [3 4]]
```

Si hago una operacion entro dos matrices con diferentes tipos de valores (int y float), python ajusta el tipo de valor del resultado al mas apropiado (float). Esto se llama upcasting.

[82]:
```
print(A.sum()) #funciones sum, max, min y mean
print(A.max())
print(A.min())
print(A.mean())
```

```
3
1
0
0.75
```

[85]:
```
b = np.arange(1, 16, 1).reshape(3, 5) #matriz de 1 a 15 de 3x5. Uso el .reshape␣
 ↪para cambiar dimension
print(b)
```

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]]
```

- Abro una imagen, tiene que estar en la misma carpeta donde estoy trabajando

[87]:
```
from PIL import Image
from IPython.display import display
```

[90]:
```python
im = Image.open('chris.tiff')
display(im)
```

[93]:
```python
array = np.array(im) #como las imagenes se almacenan en filas y columnas, puede␣
 ↪convertirla en una matriz
print(array.shape)
print(array)
```

```
(200, 200)
[[118 117 118 ... 103 107 110]
 [113 113 113 ... 100 103 106]
 [108 108 107 ...  95  98 102]
 ...
 [177 181 182 ... 193 198 192]
 [178 182 183 ... 193 201 189]
 [178 182 184 ... 193 201 187]]
```

- Los valores de la imagen son "uint8", es decir todos positivos entre 0 y 255 (2**8 = 256). Para imagenes en blanco y negro, el negro es 0 y el blanco 255. Ahora vamos a invertir el color de la imagen:

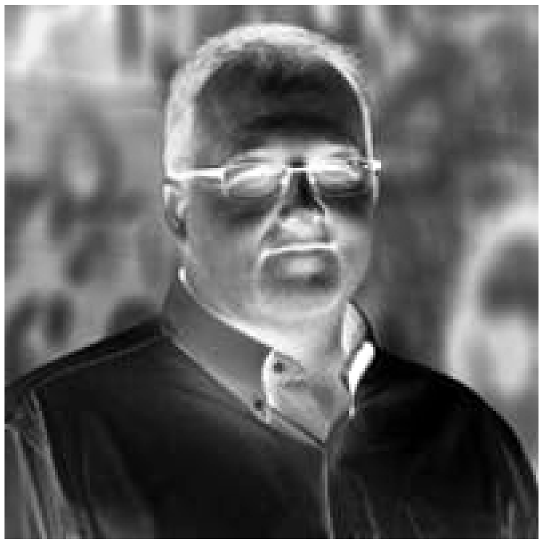
[94]:
```python
mask = np.full(array.shape, 255) #creo una mascara de las mismas dimensiones␣
 ↪con todos sus valores igual a 255
mask
```

[94]:
```
array([[255, 255, 255, ..., 255, 255, 255],
       [255, 255, 255, ..., 255, 255, 255],
       [255, 255, 255, ..., 255, 255, 255],
       ...,
       [255, 255, 255, ..., 255, 255, 255],
       [255, 255, 255, ..., 255, 255, 255],
       [255, 255, 255, ..., 255, 255, 255]])
```

[95]:
```
modified_array = array - mask #hago la resta para obtener el numero inverso
modified_array = modified_array * -1 #multiplico por -1 para que todos los␣
 ↪valores de la matriz sean positivos
modified_array = modified_array.astype(np.uint8) #le pongo el mismo formato que␣
 ↪antes para poder leerla
modified_array
```

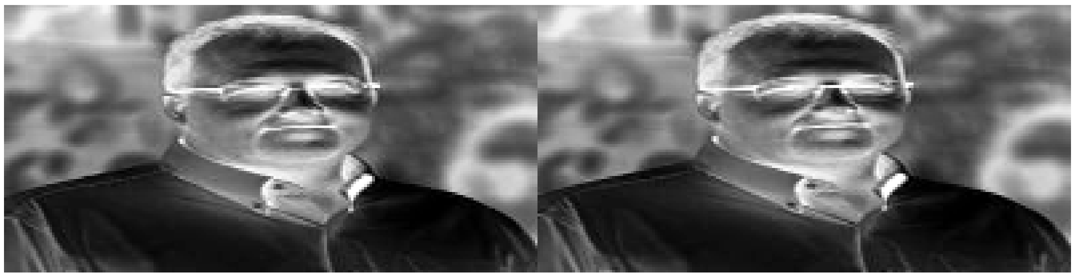
[95]:
```
array([[137, 138, 137, ..., 152, 148, 145],
        [142, 142, 142, ..., 155, 152, 149],
        [147, 147, 148, ..., 160, 157, 153],
        ...,
        [ 78,  74,  73, ...,  62,  57,  63],
        [ 77,  73,  72, ...,  62,  54,  66],
        [ 77,  73,  71, ...,  62,  54,  68]], dtype=uint8)
```

[97]:
```
display(Image.fromarray(modified_array)) #ahora muestro la foto invertida␣
 ↪fromarray
```



[98]:
```
reshaped = np.reshape(modified_array, (100, 400)) #hago un reshape manteniendo␣
 ↪el total de celdas (antes 200x200)
print(reshaped.shape)
display(Image.fromarray(reshaped))
```

```
(100, 400)
```



13 - Indexing, Slicing and Iterating

- Indexing:

[3]:
```
a = np.array([1, 3, 5, 7])
a[2]
```

[3]:
```
5
```

[4]:
```
a = np.array([[1, 2], [3, 4], [5, 6]])
a
```

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

[5]:
```
a [1, 1]
```

[5]:
```
4
```

[7]:
```
np.array([a[0, 0], a[1,1], a[2, 1]]) #puede seleccionar varios valores segun␣
  ↪los indices (filas y columnas)
```

[7]:
```
array([1, 4, 6])
```

[9]:
```
print(a[[0, 1, 2], [0, 1, 1]]) #aca selecciono primero las filas y despues las␣
  ↪columnas
```

```
[1 4 6]
```

- Boolean indexing:

[10]:
```
print(a > 5)
```

```
[[False False]
 [False False]
 [False  True]]
```

[12]:
```
print(a[a > 5]) #puedo imprimir un valor usando como mascara un booleano
```

```
[6]
```

- Slicing

[14]:
```
a = np.arange(6)
print(a[:3])
```

```
[0 1 2]
```

[15]:
```
print(a[2:4])
```

```
[2 3]
```

[16]:
```
a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
a
```

[16]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

[18]:
```
a[:2] #si pongo un solo argumento me devuelve todos los elementos de de las␣
 ↪filas correspondientes (0 y 1)
```

[18]:
```
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

[20]:
```
a[:2, 1:3] #ahora si le estoy indicando filas y columnas
```

[20]:
```
array([[2, 3],
       [6, 7]])
```

- Al hacer slicing estoy viendo a la matriz (no haciendo una copia), cualquier modificacion que haga afectara a la matriz base:

[22]:
```
sub_array = a[:2, 1:3]
print('sub array index [0,0] value before change: ', sub_array[0, 0]) #valor␣
 ↪original
sub_array[0,0] = 50 #cambio el valor a 50
print('sub array index [0,0] value after change: ', sub_array[0, 0]) #valor␣
 ↪cambiado
print('original array index [0, 1] value after change: ', a[0,1]) #valor␣
 ↪cambiado de la matriz base, el index es distinto por el slicing hecho antes
```

```
sub array index [0,0] value before change:  50
sub array index [0,0] value after change:  50
original array index [0, 1] value after change:  50
```

14 Trying Numpy with Datasets

[16]:
```
#importo un csv usando genfromtxt, con el delimiter indico como delimitar (, o ;
↪) y puedo skipear la primer fila
import numpy as np
wines = np.genfromtxt('winequality-red.csv', delimiter = ';', skip_header = 1)
wines
```

[16]:
```
array([[ 7.4  ,  0.7  ,  0.   , ...,  0.56 ,  9.4  ,  5.   ],
       [ 7.8  ,  0.88 ,  0.   , ...,  0.68 ,  9.8  ,  5.   ],
       [ 7.8  ,  0.76 ,  0.04 , ...,  0.65 ,  9.8  ,  5.   ],
       ...,
       [ 6.3  ,  0.51 ,  0.13 , ...,  0.75 , 11.   ,  6.   ],
       [ 5.9  ,  0.645,  0.12 , ...,  0.71 , 10.2  ,  5.   ],
       [ 6.   ,  0.31 ,  0.47 , ...,  0.66 , 11.   ,  6.   ]])
```

[4]:
```
print(wines[:,0]) #obtengo todos los datos de la primera columna
print(wines[:, 0:1]) #idem anterior pero lo muestra distinto
```

```
[7.4 7.8 7.8 ... 6.3 5.9 6. ]
[[7.4]
 [7.8]
 [7.8]
 ...
 [6.3]
 [5.9]
 [6. ]]
```

[5]:
```
wines[:, 0:3]
```

[5]:
```
array([[7.4  , 0.7  , 0.   ],
       [7.8  , 0.88 , 0.   ],
       [7.8  , 0.76 , 0.04 ],
       ...,
       [6.3  , 0.51 , 0.13 ],
       [5.9  , 0.645, 0.12 ],
       [6.   , 0.31 , 0.47 ]])
```

[6]:
```
wines[:, [0, 2, 4]] #si no quiero columnas consecutivas puedo especificar␣
↪cuales quiero con una lista
```

[6]:
```
array([[7.4  , 0.   , 0.076],
       [7.8  , 0.   , 0.098],
       [7.8  , 0.04 , 0.092],
       ...,
       [6.3  , 0.13 , 0.076],
       [5.9  , 0.12 , 0.075],
       [6.   , 0.47 , 0.067]])
```

[7]:
```
wines[:, -1].mean() #saco la media de la ultima columna
```

[7]:
```
5.6360225140712945
```

[48]:
```
#pongo los nombres para poder consultar datos de manera mas facil
graduate_admission = np.genfromtxt('Admission_Predict.csv', dtype = None,␣
 ↪delimiter = ',', skip_header = 1, names = ('Serial Nro', 'GRE Score', 'TOEFL␣
 ↪Score', 'University Rating', 'SOP', 'LOR', 'CGPA', 'Research', 'Chance of␣
 ↪Admit'))
graduate_admission
```

[48]:
```
array([(  1, 337, 118, 4, 4.5, 4.5, 9.65, 1, 0.92),
       (  2, 324, 107, 4, 4. , 4.5, 8.87, 1, 0.76),
       (  3, 316, 104, 3, 3. , 3.5, 8.  , 1, 0.72),
       (  4, 322, 110, 3, 3.5, 2.5, 8.67, 1, 0.8 ),
       (  5, 314, 103, 2, 2. , 3. , 8.21, 0, 0.65),
       (  6, 330, 115, 5, 4.5, 3. , 9.34, 1, 0.9 ),
       (  7, 321, 109, 3, 3. , 4. , 8.2 , 1, 0.75),
       (  8, 308, 101, 2, 3. , 4. , 7.9 , 0, 0.68),
       (  9, 302, 102, 1, 2. , 1.5, 8.  , 0, 0.5 ),
       ( 10, 323, 108, 3, 3.5, 3. , 8.6 , 0, 0.45),
       ( 11, 325, 106, 3, 3.5, 4. , 8.4 , 1, 0.52),
       ( 12, 327, 111, 4, 4. , 4.5, 9.  , 1, 0.84),
       ( 13, 328, 112, 4, 4. , 4.5, 9.1 , 1, 0.78),
       ( 14, 307, 109, 3, 4. , 3. , 8.  , 1, 0.62),
       ( 15, 311, 104, 3, 3.5, 2. , 8.2 , 1, 0.61),
       ( 16, 314, 105, 3, 3.5, 2.5, 8.3 , 0, 0.54),
       ( 17, 317, 107, 3, 4. , 3. , 8.7 , 0, 0.66),
       ( 18, 319, 106, 3, 4. , 3. , 8.  , 1, 0.65),
       ( 19, 318, 110, 3, 4. , 3. , 8.8 , 0, 0.63),
       ( 20, 303, 102, 3, 3.5, 3. , 8.5 , 0, 0.62),
       ( 21, 312, 107, 3, 3. , 2. , 7.9 , 1, 0.64),
       ( 22, 325, 114, 4, 3. , 2. , 8.4 , 0, 0.7 ),
       ( 23, 328, 116, 5, 5. , 5. , 9.5 , 1, 0.94),
       ( 24, 334, 119, 5, 5. , 4.5, 9.7 , 1, 0.95),
       ( 25, 336, 119, 5, 4. , 3.5, 9.8 , 1, 0.97),
       ( 26, 340, 120, 5, 4.5, 4.5, 9.6 , 1, 0.94),
       ( 27, 322, 109, 5, 4.5, 3.5, 8.8 , 0, 0.76),
       ( 28, 298,  98, 2, 1.5, 2.5, 7.5 , 1, 0.44),
       ( 29, 295,  93, 1, 2. , 2. , 7.2 , 0, 0.46),
       ( 30, 310,  99, 2, 1.5, 2. , 7.3 , 0, 0.54),
       ( 31, 300,  97, 2, 3. , 3. , 8.1 , 1, 0.65),
       ( 32, 327, 103, 3, 4. , 4. , 8.3 , 1, 0.74),
       ( 33, 338, 118, 4, 3. , 4.5, 9.4 , 1, 0.91),
       ( 34, 340, 114, 5, 4. , 4. , 9.6 , 1, 0.9 ),
       ( 35, 331, 112, 5, 4. , 5. , 9.8 , 1, 0.94),
       ( 36, 320, 110, 5, 5. , 5. , 9.2 , 1, 0.88),
       ( 37, 299, 106, 2, 4. , 4. , 8.4 , 0, 0.64),
       ( 38, 300, 105, 1, 1. , 2. , 7.8 , 0, 0.58),
       ( 39, 304, 105, 1, 3. , 1.5, 7.5 , 0, 0.52),
       ( 40, 307, 108, 2, 4. , 3.5, 7.7 , 0, 0.48),
       ( 41, 308, 110, 3, 3.5, 3. , 8.  , 1, 0.46),
       ( 42, 316, 105, 2, 2.5, 2.5, 8.2 , 1, 0.49),
       ( 43, 313, 107, 2, 2.5, 2. , 8.5 , 1, 0.53),
       ( 44, 332, 117, 4, 4.5, 4. , 9.1 , 0, 0.87),
       ( 45, 326, 113, 5, 4.5, 4. , 9.4 , 1, 0.91),
       ( 46, 322, 110, 5, 5. , 4. , 9.1 , 1, 0.88),
       ( 47, 329, 114, 5, 4. , 5. , 9.3 , 1, 0.86),
       ( 48, 339, 119, 5, 4.5, 4. , 9.7 , 0, 0.89),
       ( 49, 321, 110, 3, 3.5, 5. , 8.85, 1, 0.82),
       ( 50, 327, 111, 4, 3. , 4. , 8.4 , 1, 0.78),
       ( 51, 313,  98, 3, 2.5, 4.5, 8.3 , 1, 0.76),
       ( 52, 312, 100, 2, 1.5, 3.5, 7.9 , 1, 0.56),
       ( 53, 334, 116, 4, 4. , 3. , 8.  , 1, 0.78),
       ( 54, 324, 112, 4, 4. , 2.5, 8.1 , 1, 0.72),
       ( 55, 322, 110, 3, 3. , 3.5, 8.  , 0, 0.7 ),
       ( 56, 320, 103, 3, 3. , 3. , 7.7 , 0, 0.64),
       ( 57, 316, 102, 3, 2. , 3. , 7.4 , 0, 0.64),
       ( 58, 298,  99, 2, 4. , 2. , 7.6 , 0, 0.46),
       ( 59, 300,  99, 1, 3. , 2. , 6.8 , 1, 0.36),
       ( 60, 311, 104, 2, 2. , 2. , 8.3 , 0, 0.42),
       ( 61, 309, 100, 2, 3. , 3. , 8.1 , 0, 0.48),
       ( 62, 307, 101, 3, 4. , 3. , 8.2 , 0, 0.47),
       ( 63, 304, 105, 2, 3. , 3. , 8.2 , 1, 0.54),
       ( 64, 315, 107, 2, 4. , 3. , 8.5 , 1, 0.56),
       ( 65, 325, 111, 3, 3. , 3.5, 8.7 , 0, 0.52),
       ( 66, 325, 112, 4, 3.5, 3.5, 8.92, 0, 0.55),
       ( 67, 327, 114, 3, 3. , 3. , 9.02, 0, 0.61),
       ( 68, 316, 107, 2, 3.5, 3.5, 8.64, 1, 0.57),
       ( 69, 318, 109, 3, 3.5, 4. , 9.22, 1, 0.68),
       ( 70, 328, 115, 4, 4.5, 4. , 9.16, 1, 0.78),
       ( 71, 332, 118, 5, 5. , 5. , 9.64, 1, 0.94),
       ( 72, 336, 112, 5, 5. , 5. , 9.76, 1, 0.96),
       ( 73, 321, 111, 5, 5. , 5. , 9.45, 1, 0.93),
       ( 74, 314, 108, 4, 4.5, 4. , 9.04, 1, 0.84),
       ( 75, 314, 106, 3, 3. , 5. , 8.9 , 0, 0.74),
       ( 76, 329, 114, 2, 2. , 4. , 8.56, 1, 0.72),
       ( 77, 327, 112, 3, 3. , 3. , 8.72, 1, 0.74),
       ( 78, 301,  99, 2, 3. , 2. , 8.22, 0, 0.64),
       ( 79, 296,  95, 2, 3. , 2. , 7.54, 1, 0.44),
       ( 80, 294,  93, 1, 1.5, 2. , 7.36, 0, 0.46),
       ( 81, 312, 105, 3, 2. , 3. , 8.02, 1, 0.5 ),
       ( 82, 340, 120, 4, 5. , 5. , 9.5 , 1, 0.96),
       ( 83, 320, 110, 5, 5. , 4.5, 9.22, 1, 0.92),
       ( 84, 322, 115, 5, 4. , 4.5, 9.36, 1, 0.92),
       ( 85, 340, 115, 5, 4.5, 4.5, 9.45, 1, 0.94),
       ( 86, 319, 103, 4, 4.5, 3.5, 8.66, 0, 0.76),
       ( 87, 315, 106, 3, 4.5, 3.5, 8.42, 0, 0.72),
       ( 88, 317, 107, 2, 3.5, 3. , 8.28, 0, 0.66),
       ( 89, 314, 108, 3, 4.5, 3.5, 8.14, 0, 0.64),
       ( 90, 316, 109, 4, 4.5, 3.5, 8.76, 1, 0.74),
       ( 91, 318, 106, 2, 4. , 4. , 7.92, 1, 0.64),
       ( 92, 299,  97, 3, 5. , 3.5, 7.66, 0, 0.38),
       ( 93, 298,  98, 2, 4. , 3. , 8.03, 0, 0.34),
       ( 94, 301,  97, 2, 3. , 3. , 7.88, 1, 0.44),
       ( 95, 303,  99, 3, 2. , 2.5, 7.66, 0, 0.36),
       ( 96, 304, 100, 4, 1.5, 2.5, 7.84, 0, 0.42),
       ( 97, 306, 100, 2, 3. , 3. , 8.  , 0, 0.48),
       ( 98, 331, 120, 3, 4. , 4. , 8.96, 1, 0.86),
       ( 99, 332, 119, 4, 5. , 4.5, 9.24, 1, 0.9 ),
       (100, 323, 113, 3, 4. , 4. , 8.88, 1, 0.79),
       (101, 322, 107, 3, 3.5, 3.5, 8.46, 1, 0.71),
       (102, 312, 105, 2, 2.5, 3. , 8.12, 0, 0.64),
       (103, 314, 106, 2, 4. , 3.5, 8.25, 0, 0.62),
       (104, 317, 104, 2, 4.5, 4. , 8.47, 0, 0.57),
       (105, 326, 112, 3, 3.5, 3. , 9.05, 1, 0.74),
       (106, 316, 110, 3, 4. , 4.5, 8.78, 1, 0.69),
       (107, 329, 111, 4, 4.5, 4.5, 9.18, 1, 0.87),
       (108, 338, 117, 4, 3.5, 4.5, 9.46, 1, 0.91),
       (109, 331, 116, 5, 5. , 5. , 9.38, 1, 0.93),
       (110, 304, 103, 5, 5. , 4. , 8.64, 0, 0.68),
       (111, 305, 108, 5, 3. , 3. , 8.48, 0, 0.61),
       (112, 321, 109, 4, 4. , 4. , 8.68, 1, 0.69),
       (113, 301, 107, 3, 3.5, 3.5, 8.34, 1, 0.62),
       (114, 320, 110, 2, 4. , 3.5, 8.56, 0, 0.72),
       (115, 311, 105, 3, 3.5, 3. , 8.45, 1, 0.59),
       (116, 310, 106, 4, 4.5, 4.5, 9.04, 1, 0.66),
       (117, 299, 102, 3, 4. , 3.5, 8.62, 0, 0.56),
       (118, 290, 104, 4, 2. , 2.5, 7.46, 0, 0.45),
       (119, 296,  99, 2, 3. , 3.5, 7.28, 0, 0.47),
       (120, 327, 104, 5, 3. , 3.5, 8.84, 1, 0.71),
       (121, 335, 117, 5, 5. , 5. , 9.56, 1, 0.94),
       (122, 334, 119, 5, 4.5, 4.5, 9.48, 1, 0.94),
       (123, 310, 106, 4, 1.5, 2.5, 8.36, 0, 0.57),
       (124, 308, 108, 3, 3.5, 3.5, 8.22, 0, 0.61),
       (125, 301, 106, 4, 2.5, 3. , 8.47, 0, 0.57),
       (126, 300, 100, 3, 2. , 3. , 8.66, 1, 0.64),
       (127, 323, 113, 3, 4. , 3. , 9.32, 1, 0.85),
       (128, 319, 112, 3, 2.5, 2. , 8.71, 1, 0.78),
       (129, 326, 112, 3, 3.5, 3. , 9.1 , 1, 0.84),
       (130, 333, 118, 5, 5. , 5. , 9.35, 1, 0.92),
       (131, 339, 114, 5, 4. , 4.5, 9.76, 1, 0.96),
       (132, 303, 105, 5, 5. , 4.5, 8.65, 0, 0.77),
       (133, 309, 105, 5, 3.5, 3.5, 8.56, 0, 0.71),
       (134, 323, 112, 5, 4. , 4.5, 8.78, 0, 0.79),
       (135, 333, 113, 5, 4. , 4. , 9.28, 1, 0.89),
       (136, 314, 109, 4, 3.5, 4. , 8.77, 1, 0.82),
       (137, 312, 103, 3, 5. , 4. , 8.45, 0, 0.76),
       (138, 316, 100, 2, 1.5, 3. , 8.16, 1, 0.71),
       (139, 326, 116, 2, 4.5, 3. , 9.08, 1, 0.8 ),
       (140, 318, 109, 1, 3.5, 3.5, 9.12, 0, 0.78),
       (141, 329, 110, 2, 4. , 3. , 9.15, 1, 0.84),
       (142, 332, 118, 2, 4.5, 3.5, 9.36, 1, 0.9 ),
       (143, 331, 115, 5, 4. , 3.5, 9.44, 1, 0.92),
       (144, 340, 120, 4, 4.5, 4. , 9.92, 1, 0.97),
       (145, 325, 112, 2, 3. , 3.5, 8.96, 1, 0.8 ),
       (146, 320, 113, 2, 2. , 2.5, 8.64, 1, 0.81),
       (147, 315, 105, 3, 2. , 2.5, 8.48, 0, 0.75),
       (148, 326, 114, 3, 3. , 3. , 9.11, 1, 0.83),
       (149, 339, 116, 4, 4. , 3.5, 9.8 , 1, 0.96),
       (150, 311, 106, 2, 3.5, 3. , 8.26, 1, 0.79),
       (151, 334, 114, 4, 4. , 4. , 9.43, 1, 0.93),
       (152, 332, 116, 5, 5. , 5. , 9.28, 1, 0.94),
       (153, 321, 112, 5, 5. , 5. , 9.06, 1, 0.86),
       (154, 324, 105, 3, 3. , 4. , 8.75, 0, 0.79),
       (155, 326, 108, 3, 3. , 3.5, 8.89, 0, 0.8 ),
       (156, 312, 109, 3, 3. , 3. , 8.69, 0, 0.77),
       (157, 315, 105, 3, 2. , 2.5, 8.34, 0, 0.7 ),
       (158, 309, 104, 2, 2. , 2.5, 8.26, 0, 0.65),
       (159, 306, 106, 2, 2. , 2.5, 8.14, 0, 0.61),
       (160, 297, 100, 1, 1.5, 2. , 7.9 , 0, 0.52),
       (161, 315, 103, 1, 1.5, 2. , 7.86, 0, 0.57),
       (162, 298,  99, 1, 1.5, 3. , 7.46, 0, 0.53),
       (163, 318, 109, 3, 3. , 3. , 8.5 , 0, 0.67),
       (164, 317, 105, 3, 3.5, 3. , 8.56, 0, 0.68),
       (165, 329, 111, 4, 4.5, 4. , 9.01, 1, 0.81),
       (166, 322, 110, 5, 4.5, 4. , 8.97, 0, 0.78),
       (167, 302, 102, 3, 3.5, 5. , 8.33, 0, 0.65),
       (168, 313, 102, 3, 2. , 3. , 8.27, 0, 0.64),
       (169, 293,  97, 2, 2. , 4. , 7.8 , 1, 0.64),
       (170, 311,  99, 2, 2.5, 3. , 7.98, 0, 0.65),
       (171, 312, 101, 2, 2.5, 3.5, 8.04, 1, 0.68),
       (172, 334, 117, 5, 4. , 4.5, 9.07, 1, 0.89),
       (173, 322, 110, 4, 4. , 5. , 9.13, 1, 0.86),
       (174, 323, 113, 4, 4. , 4.5, 9.23, 1, 0.89),
       (175, 321, 111, 4, 4. , 4. , 8.97, 1, 0.87),
       (176, 320, 111, 4, 4.5, 3.5, 8.87, 1, 0.85),
       (177, 329, 119, 4, 4.5, 4.5, 9.16, 1, 0.9 ),
       (178, 319, 110, 3, 3.5, 3.5, 9.04, 0, 0.82),
       (179, 309, 108, 3, 2.5, 3. , 8.12, 0, 0.72),
       (180, 307, 102, 3, 3. , 3. , 8.27, 0, 0.73),
       (181, 300, 104, 3, 3.5, 3. , 8.16, 0, 0.71),
       (182, 305, 107, 2, 2.5, 2.5, 8.42, 0, 0.71),
       (183, 299, 100, 2, 3. , 3.5, 7.88, 0, 0.68),
       (184, 314, 110, 3, 4. , 4. , 8.8 , 0, 0.75),
       (185, 316, 106, 2, 2.5, 4. , 8.32, 0, 0.72),
       (186, 327, 113, 4, 4.5, 4.5, 9.11, 1, 0.89),
       (187, 317, 107, 3, 3.5, 3. , 8.68, 1, 0.84),
       (188, 335, 118, 5, 4.5, 3.5, 9.44, 1, 0.93),
       (189, 331, 115, 5, 4.5, 3.5, 9.36, 1, 0.93),
       (190, 324, 112, 5, 5. , 5. , 9.08, 1, 0.88),
       (191, 324, 111, 5, 4.5, 4. , 9.16, 1, 0.9 ),
       (192, 323, 110, 5, 4. , 5. , 8.98, 1, 0.87),
       (193, 322, 114, 5, 4.5, 4. , 8.94, 1, 0.86),
       (194, 336, 118, 5, 4.5, 5. , 9.53, 1, 0.94),
       (195, 316, 109, 3, 3.5, 3. , 8.76, 0, 0.77),
       (196, 307, 107, 2, 3. , 3.5, 8.52, 1, 0.78),
       (197, 306, 105, 2, 3. , 2.5, 8.26, 0, 0.73),
       (198, 310, 106, 2, 3.5, 2.5, 8.33, 0, 0.73),
       (199, 311, 104, 3, 4.5, 4.5, 8.43, 0, 0.7 ),
       (200, 313, 107, 3, 4. , 4.5, 8.69, 0, 0.72),
       (201, 317, 103, 3, 2.5, 3. , 8.54, 1, 0.73),
       (202, 315, 110, 2, 3.5, 3. , 8.46, 1, 0.72),
       (203, 340, 120, 5, 4.5, 4.5, 9.91, 1, 0.97),
       (204, 334, 120, 5, 4. , 5. , 9.87, 1, 0.97),
       (205, 298, 105, 3, 3.5, 4. , 8.54, 0, 0.69),
       (206, 295,  99, 2, 2.5, 3. , 7.65, 0, 0.57),
       (207, 315,  99, 2, 3.5, 3. , 7.89, 0, 0.63),
       (208, 310, 102, 3, 3.5, 4. , 8.02, 1, 0.66),
       (209, 305, 106, 2, 3. , 3. , 8.16, 0, 0.64),
       (210, 301, 104, 3, 3.5, 4. , 8.12, 1, 0.68),
       (211, 325, 108, 4, 4.5, 4. , 9.06, 1, 0.79),
       (212, 328, 110, 4, 5. , 4. , 9.14, 1, 0.82),
       (213, 338, 120, 4, 5. , 5. , 9.66, 1, 0.95),
       (214, 333, 119, 5, 5. , 4.5, 9.78, 1, 0.96),
       (215, 331, 117, 4, 4.5, 5. , 9.42, 1, 0.94),
       (216, 330, 116, 5, 5. , 4.5, 9.36, 1, 0.93),
       (217, 322, 112, 4, 4.5, 4.5, 9.26, 1, 0.91),
       (218, 321, 109, 4, 4. , 4. , 9.13, 1, 0.85),
       (219, 324, 110, 4, 3. , 3.5, 8.97, 1, 0.84),
       (220, 312, 104, 3, 3.5, 3.5, 8.42, 0, 0.74),
       (221, 313, 103, 3, 4. , 4. , 8.75, 0, 0.76),
       (222, 316, 110, 3, 3.5, 4. , 8.56, 0, 0.75),
       (223, 324, 113, 4, 4.5, 4. , 8.79, 0, 0.76),
       (224, 308, 109, 2, 3. , 4. , 8.45, 0, 0.71),
       (225, 305, 105, 2, 3. , 2. , 8.23, 0, 0.67),
       (226, 296,  99, 2, 2.5, 2.5, 8.03, 0, 0.61),
       (227, 306, 110, 2, 3.5, 4. , 8.45, 0, 0.63),
       (228, 312, 110, 2, 3.5, 3. , 8.53, 0, 0.64),
       (229, 318, 112, 3, 4. , 3.5, 8.67, 0, 0.71),
       (230, 324, 111, 4, 3. , 3. , 9.01, 1, 0.82),
       (231, 313, 104, 3, 4. , 4.5, 8.65, 0, 0.73),
       (232, 319, 106, 3, 3.5, 2.5, 8.33, 1, 0.74),
       (233, 312, 107, 2, 2.5, 3.5, 8.27, 0, 0.69),
       (234, 304, 100, 2, 2.5, 3.5, 8.07, 0, 0.64),
       (235, 330, 113, 5, 5. , 4. , 9.31, 1, 0.91),
       (236, 326, 111, 5, 4.5, 4. , 9.23, 1, 0.88),
       (237, 325, 112, 4, 4. , 4.5, 9.17, 1, 0.85),
       (238, 329, 114, 5, 4.5, 5. , 9.19, 1, 0.86),
       (239, 310, 104, 3, 2. , 3.5, 8.37, 0, 0.7 ),
       (240, 299, 100, 1, 1.5, 2. , 7.89, 0, 0.59),
       (241, 296, 101, 1, 2.5, 3. , 7.68, 0, 0.6 ),
       (242, 317, 103, 2, 2.5, 2. , 8.15, 0, 0.65),
       (243, 324, 115, 3, 3.5, 3. , 8.76, 1, 0.7 ),
       (244, 325, 114, 3, 3.5, 3. , 9.04, 1, 0.76),
       (245, 314, 107, 2, 2.5, 4. , 8.56, 0, 0.63),
       (246, 328, 110, 4, 4. , 2.5, 9.02, 1, 0.81),
       (247, 316, 105, 3, 3. , 3.5, 8.73, 0, 0.72),
       (248, 311, 104, 2, 2.5, 3.5, 8.48, 0, 0.71),
       (249, 324, 110, 3, 3.5, 4. , 8.87, 1, 0.8 ),
       (250, 321, 111, 3, 3.5, 4. , 8.83, 1, 0.77),
       (251, 320, 104, 3, 3. , 2.5, 8.57, 1, 0.74),
       (252, 316,  99, 2, 2.5, 3. , 9.  , 0, 0.7 ),
       (253, 318, 100, 2, 2.5, 3.5, 8.54, 1, 0.71),
       (254, 335, 115, 4, 4.5, 4.5, 9.68, 1, 0.93),
       (255, 321, 114, 4, 4. , 5. , 9.12, 0, 0.85),
       (256, 307, 110, 4, 4. , 4.5, 8.37, 0, 0.79),
       (257, 309,  99, 3, 4. , 4. , 8.56, 0, 0.76),
       (258, 324, 100, 3, 4. , 5. , 8.64, 1, 0.78),
       (259, 326, 102, 4, 5. , 5. , 8.76, 1, 0.77),
       (260, 331, 119, 4, 5. , 4.5, 9.34, 1, 0.9 ),
       (261, 327, 108, 5, 5. , 3.5, 9.13, 1, 0.87),
       (262, 312, 104, 3, 3.5, 4. , 8.09, 0, 0.71),
       (263, 308, 103, 2, 2.5, 4. , 8.36, 1, 0.7 ),
       (264, 324, 111, 3, 2.5, 1.5, 8.79, 1, 0.7 ),
       (265, 325, 110, 2, 3. , 2.5, 8.76, 1, 0.75),
       (266, 313, 102, 3, 2.5, 2.5, 8.68, 0, 0.71),
       (267, 312, 105, 2, 2. , 2.5, 8.45, 0, 0.72),
       (268, 314, 107, 3, 3. , 3.5, 8.17, 1, 0.73),
       (269, 327, 113, 4, 4.5, 5. , 9.14, 0, 0.83),
       (270, 308, 108, 4, 4.5, 5. , 8.34, 0, 0.77),
       (271, 306, 105, 2, 2.5, 3. , 8.22, 1, 0.72),
       (272, 299,  96, 2, 1.5, 2. , 7.86, 0, 0.54),
       (273, 294,  95, 1, 1.5, 1.5, 7.64, 0, 0.49),
       (274, 312,  99, 1, 1. , 1.5, 8.01, 1, 0.52),
       (275, 315, 100, 1, 2. , 2.5, 7.95, 0, 0.58),
       (276, 322, 110, 3, 3.5, 3. , 8.96, 1, 0.78),
       (277, 329, 113, 5, 5. , 4.5, 9.45, 1, 0.89),
       (278, 320, 101, 2, 2.5, 3. , 8.62, 0, 0.7 ),
       (279, 308, 103, 2, 3. , 3.5, 8.49, 0, 0.66),
       (280, 304, 102, 2, 3. , 4. , 8.73, 0, 0.67),
       (281, 311, 102, 3, 4.5, 4. , 8.64, 1, 0.68),
       (282, 317, 110, 3, 4. , 4.5, 9.11, 1, 0.8 ),
       (283, 312, 106, 3, 4. , 3.5, 8.79, 1, 0.81),
       (284, 321, 111, 3, 2.5, 3. , 8.9 , 1, 0.8 ),
       (285, 340, 112, 4, 5. , 4.5, 9.66, 1, 0.94),
       (286, 331, 116, 5, 4. , 4. , 9.26, 1, 0.93),
       (287, 336, 118, 5, 4.5, 4. , 9.19, 1, 0.92),
       (288, 324, 114, 5, 5. , 4.5, 9.08, 1, 0.89),
       (289, 314, 104, 4, 5. , 5. , 9.02, 0, 0.82),
       (290, 313, 109, 3, 4. , 3.5, 9.  , 0, 0.79),
       (291, 307, 105, 2, 2.5, 3. , 7.65, 0, 0.58),
       (292, 300, 102, 2, 1.5, 2. , 7.87, 0, 0.56),
       (293, 302,  99, 2, 1. , 2. , 7.97, 0, 0.56),
       (294, 312,  98, 1, 3.5, 3. , 8.18, 1, 0.64),
       (295, 316, 101, 2, 2.5, 2. , 8.32, 1, 0.61),
       (296, 317, 100, 2, 3. , 2.5, 8.57, 0, 0.68),
       (297, 310, 107, 3, 3.5, 3.5, 8.67, 0, 0.76),
       (298, 320, 120, 3, 4. , 4.5, 9.11, 0, 0.86),
       (299, 330, 114, 3, 4.5, 4.5, 9.24, 1, 0.9 ),
       (300, 305, 112, 3, 3. , 3.5, 8.65, 0, 0.71),
       (301, 309, 106, 2, 2.5, 2.5, 8.  , 0, 0.62),
       (302, 319, 108, 2, 2.5, 3. , 8.76, 0, 0.66),
       (303, 322, 105, 2, 3. , 3. , 8.45, 1, 0.65),
       (304, 323, 107, 3, 3.5, 3.5, 8.55, 1, 0.73),
       (305, 313, 106, 2, 2.5, 2. , 8.43, 0, 0.62),
       (306, 321, 109, 3, 3.5, 3.5, 8.8 , 1, 0.74),
       (307, 323, 110, 3, 4. , 3.5, 9.1 , 1, 0.79),
       (308, 325, 112, 4, 4. , 4. , 9.  , 1, 0.8 ),
       (309, 312, 108, 3, 3.5, 3. , 8.53, 0, 0.69),
       (310, 308, 110, 4, 3.5, 3. , 8.6 , 0, 0.7 ),
       (311, 320, 104, 3, 3. , 3.5, 8.74, 1, 0.76),
       (312, 328, 108, 4, 4.5, 4. , 9.18, 1, 0.84),
       (313, 311, 107, 4, 4.5, 4.5, 9.  , 1, 0.78),
       (314, 301, 100, 3, 3.5, 3. , 8.04, 0, 0.67),
       (315, 305, 105, 2, 3. , 4. , 8.13, 0, 0.66),
       (316, 308, 104, 2, 2.5, 3. , 8.07, 0, 0.65),
       (317, 298, 101, 2, 1.5, 2. , 7.86, 0, 0.54),
       (318, 300,  99, 1, 1. , 2.5, 8.01, 0, 0.58),
       (319, 324, 111, 3, 2.5, 2. , 8.8 , 1, 0.79),
       (320, 327, 113, 4, 3.5, 3. , 8.69, 1, 0.8 ),
       (321, 317, 106, 3, 4. , 3.5, 8.5 , 1, 0.75),
       (322, 323, 104, 3, 4. , 4. , 8.44, 1, 0.73),
       (323, 314, 107, 2, 2.5, 4. , 8.27, 0, 0.72),
       (324, 305, 102, 2, 2. , 2.5, 8.18, 0, 0.62),
       (325, 315, 104, 3, 3. , 2.5, 8.33, 0, 0.67),
       (326, 326, 116, 3, 3.5, 4. , 9.14, 1, 0.81),
       (327, 299, 100, 3, 2. , 2. , 8.02, 0, 0.63),
       (328, 295, 101, 2, 2.5, 2. , 7.86, 0, 0.69),
       (329, 324, 112, 4, 4. , 3.5, 8.77, 1, 0.8 ),
       (330, 297,  96, 2, 2.5, 1.5, 7.89, 0, 0.43),
       (331, 327, 113, 3, 3.5, 3. , 8.66, 1, 0.8 ),
       (332, 311, 105, 2, 3. , 2. , 8.12, 1, 0.73),
       (333, 308, 106, 3, 3.5, 2.5, 8.21, 1, 0.75),
       (334, 319, 108, 3, 3. , 3.5, 8.54, 1, 0.71),
       (335, 312, 107, 4, 4.5, 4. , 8.65, 1, 0.73),
       (336, 325, 111, 4, 4. , 4.5, 9.11, 1, 0.83),
       (337, 319, 110, 3, 3. , 2.5, 8.79, 0, 0.72),
       (338, 332, 118, 5, 5. , 5. , 9.47, 1, 0.94),
       (339, 323, 108, 5, 4. , 4. , 8.74, 1, 0.81),
       (340, 324, 107, 5, 3.5, 4. , 8.66, 1, 0.81),
       (341, 312, 107, 3, 3. , 3. , 8.46, 1, 0.75),
       (342, 326, 110, 3, 3.5, 3.5, 8.76, 1, 0.79),
       (343, 308, 106, 3, 3. , 3. , 8.24, 0, 0.58),
       (344, 305, 103, 2, 2.5, 3.5, 8.13, 0, 0.59),
       (345, 295,  96, 2, 1.5, 2. , 7.34, 0, 0.47),
       (346, 316,  98, 1, 1.5, 2. , 7.43, 0, 0.49),
       (347, 304,  97, 2, 1.5, 2. , 7.64, 0, 0.47),
       (348, 299,  94, 1, 1. , 1. , 7.34, 0, 0.42),
       (349, 302,  99, 1, 2. , 2. , 7.25, 0, 0.57),
       (350, 313, 101, 3, 2.5, 3. , 8.04, 0, 0.62),
       (351, 318, 107, 3, 3. , 3.5, 8.27, 1, 0.74),
       (352, 325, 110, 4, 3.5, 4. , 8.67, 1, 0.73),
       (353, 303, 100, 2, 3. , 3.5, 8.06, 1, 0.64),
       (354, 300, 102, 3, 3.5, 2.5, 8.17, 0, 0.63),
       (355, 297,  98, 2, 2.5, 3. , 7.67, 0, 0.59),
       (356, 317, 106, 2, 2. , 3.5, 8.12, 0, 0.73),
       (357, 327, 109, 3, 3.5, 4. , 8.77, 1, 0.79),
       (358, 301, 104, 2, 3.5, 3.5, 7.89, 1, 0.68),
       (359, 314, 105, 2, 2.5, 2. , 7.64, 0, 0.7 ),
       (360, 321, 107, 2, 2. , 1.5, 8.44, 0, 0.81),
       (361, 322, 110, 3, 4. , 5. , 8.64, 1, 0.85),
       (362, 334, 116, 4, 4. , 3.5, 9.54, 1, 0.93),
       (363, 338, 115, 5, 4.5, 5. , 9.23, 1, 0.91),
       (364, 306, 103, 2, 2.5, 3. , 8.36, 0, 0.69),
       (365, 313, 102, 3, 3.5, 4. , 8.9 , 1, 0.77),
       (366, 330, 114, 4, 4.5, 3. , 9.17, 1, 0.86),
       (367, 320, 104, 3, 3.5, 4.5, 8.34, 1, 0.74),
       (368, 311,  98, 1, 1. , 2.5, 7.46, 0, 0.57),
       (369, 298,  92, 1, 2. , 2. , 7.88, 0, 0.51),
       (370, 301,  98, 1, 2. , 3. , 8.03, 1, 0.67),
       (371, 310, 103, 2, 2.5, 2.5, 8.24, 0, 0.72),
       (372, 324, 110, 3, 3.5, 3. , 9.22, 1, 0.89),
       (373, 336, 119, 4, 4.5, 4. , 9.62, 1, 0.95),
       (374, 321, 109, 3, 3. , 3. , 8.54, 1, 0.79),
       (375, 315, 105, 2, 2. , 2.5, 7.65, 0, 0.39),
       (376, 304, 101, 2, 2. , 2.5, 7.66, 0, 0.38),
       (377, 297,  96, 2, 2.5, 2. , 7.43, 0, 0.34),
       (378, 290, 100, 1, 1.5, 2. , 7.56, 0, 0.47),
       (379, 303,  98, 1, 2. , 2.5, 7.65, 0, 0.56),
       (380, 311,  99, 1, 2.5, 3. , 8.43, 1, 0.71),
       (381, 322, 104, 3, 3.5, 4. , 8.84, 1, 0.78),
       (382, 319, 105, 3, 3. , 3.5, 8.67, 1, 0.73),
       (383, 324, 110, 4, 4.5, 4. , 9.15, 1, 0.82),
       (384, 300, 100, 3, 3. , 3.5, 8.26, 0, 0.62),
       (385, 340, 113, 4, 5. , 5. , 9.74, 1, 0.96),
       (386, 335, 117, 5, 5. , 5. , 9.82, 1, 0.96),
       (387, 302, 101, 2, 2.5, 3.5, 7.96, 0, 0.46),
       (388, 307, 105, 2, 2. , 3.5, 8.1 , 0, 0.53),
       (389, 296,  97, 2, 1.5, 2. , 7.8 , 0, 0.49),
       (390, 320, 108, 3, 3.5, 4. , 8.44, 1, 0.76),
       (391, 314, 102, 2, 2. , 2.5, 8.24, 0, 0.64),
       (392, 318, 106, 3, 2. , 3. , 8.65, 0, 0.71),
       (393, 326, 112, 4, 4. , 3.5, 9.12, 1, 0.84),
       (394, 317, 104, 2, 3. , 3. , 8.76, 0, 0.77),
       (395, 329, 111, 4, 4.5, 4. , 9.23, 1, 0.89),
       (396, 324, 110, 3, 3.5, 3.5, 9.04, 1, 0.82),
       (397, 325, 107, 3, 3. , 3.5, 9.11, 1, 0.84),
       (398, 330, 116, 4, 5. , 4.5, 9.45, 1, 0.91),
       (399, 312, 103, 3, 3.5, 4. , 8.78, 0, 0.67),
       (400, 333, 117, 4, 5. , 4. , 9.66, 1, 0.95)],
      dtype=[('Serial_Nro', '<i8'), ('GRE_Score', '<i8'), ('TOEFL_Score',
'<i8'), ('University_Rating', '<i8'), ('SOP', '<f8'), ('LOR', '<f8'), ('CGPA',
'<f8'), ('Research', '<i8'), ('Chance_of_Admit', '<f8')])
```

[18]:
```
graduate_admission.shape
```

[18]:
```
(400,)
```

[33]:
```
graduate_admission['CGPA'][0:5] #para todos los CGPA muestro las primeras 5␣
 ↪filas
```

[33]:
```
array([1.544 , 1.4192, 1.28  , 1.3872, 1.3136])
```

[49]:
```
#paso el valor de CGPA de una escala de 1 a 10 a escala de 1 a 4
graduate_admission['CGPA'] = graduate_admission['CGPA']/ 10 * 4
graduate_admission['CGPA'][0:20]
```

[49]:
```
array([3.86 , 3.548, 3.2  , 3.468, 3.284, 3.736, 3.28 , 3.16 , 3.2  ,
       3.44 , 3.36 , 3.6  , 3.64 , 3.2  , 3.28 , 3.32 , 3.48 , 3.2  ,
        3.52 , 3.4  ])
```

[30]:
```
len(graduate_admission[graduate_admission['Research'] == 1]) #uso una mascara␣
↪para seleccionar los que tienen 1 en research y determino la cantidad con len
```

[30]:
```
219
```

- Para los estudiantes con una chance de admision alta (>0.8) y baja (<0.4) calculamos la media del GRE score:

[37]:
```
#tengo que usar _ para los espacios en los nombres. Primero hago una mask con␣
↪los estudiantes que me importan segun chance
#de admision. Luego le digo que calcule la  media del GRE score para esa mask.
print(graduate_admission[graduate_admission['Chance_of_Admit'] > 0.
↪8]['GRE_Score'].mean())
print(graduate_admission[graduate_admission['Chance_of_Admit'] < 0.
↪4]['GRE_Score'].mean())
```

```
328.7350427350427
302.2857142857143
```

[50]:
```
print(graduate_admission[graduate_admission['Chance_of_Admit'] > 0.8]['CGPA'].
↪mean())
print(graduate_admission[graduate_admission['Chance_of_Admit'] < 0.4]['CGPA'].
↪mean())
```

```
3.7106666666666666
3.0222857142857142
```

15 Manipulating Text with Regular Expression

[1]:
```
import re
```

[2]:
```
text = 'This is a good day' #creo un texto

if re.search('good', text): #con la funcion research obtengo un boolean si la␣
↪palabra indicada esta en el texto
    print('Wonderfull')
else:
    print(':(')
```

```
Wonderfull
```

[3]:
```
if re.match('This', text): #con el match obtengo un boolean si la palabra␣
 ↪indicada es la primera del texto
    print('Está')
else:
    print('No está')
```

```
Está
```

[4]:
```
text = 'Amy works diligently. Amy gets goods grades. Our student Amy is␣
 ↪succedful'

re.split('Amy', text) #con split divido por la palabra deseada (en este caso␣
 ↪Amy) y me devuelve una lista
```

[4]:
```
['',
 ' works diligently. ',
 ' gets goods grades. Our student ',
 ' is succedful']
```

[5]:
```
re.findall('Amy', text) #con findall obtengo todas las veces que menciono la␣
 ↪palabra ingresada
```

[5]:
```
['Amy', 'Amy', 'Amy']
```

[6]:
```
#Se utiliza ^ para indicar el comienzo del string que deseo matchear. Para el␣
 ↪final se utiliza $.
#Si se pone ^ al principio de un string significa que lo que quiero matchear␣
 ↪empieza con string.
#Analogamente para el final con $.
```

[7]:
```
re.search('^Amy', text) #devuelve un objeto, este objeto es siempre True.␣
 ↪Tambien indica la palabra matcheada y la ubicacion
```

[7]:
```
<re.Match object; span=(0, 3), match='Amy'>
```

16 Patterns and Character Classes

[8]:
```
#Supongamos las notas de un estudiante en todos los cursos que hizo
grades = 'ACAAAABCBCBAA'

re.findall('B', grades) #cuantas B se saco en los cursos
```

[8]:
```
['B', 'B', 'B']
```

[9]:
```
re.findall('[AB]', grades) #cuantas notas A o B saco el estudiante
```

[9]:
```
['A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'A', 'A']
```

[10]:
```
re.findall('[A][B-C]', grades) #todas las veces que recibio un A y luego un B o␣
 ↪C
```

[10]:
```
['AC', 'AB']
```

[11]:
```
re.findall('AB|AC', grades) #idem anterior pero usando OR (|)
```

[11]:
```
['AC', 'AB']
```

[12]:
```
re.findall('[^A]', grades) #el  ^ es el operador para negar (tiene que estar␣
 ↪dentro de los corchetes), es decir voy a obtener todas las notas menos las A
```

[12]:
```
['C', 'B', 'C', 'B', 'C', 'B']
```

[13]:
```
re.findall('^[^A]', grades) #el resultado es vacio porque el estoy pidiendo␣
 ↪cualquier valor al principio del string que no sea A
#como el string empieza con A, no hay match
```

[13]:
```
[]
```

17 Quantifiers

Los quantifiers sirven para matchear un patron una cierta cantidad de veces. Se escribe de esta manera 'e{m,n}', siendo e lo que quiero matchear, m el valor minimo de veces y n el valor maximo de veces. IMPORTANTE: no debe haber espacio entre los valores de m, n y la coma dentro de las llaves.

[14]:
```
re.findall('A{2,10}', grades) #cuantas veces el alumno saco dos A o mas␣
 ↪consecutivas
#busca combinaciones de dos A hasta diez A.
```

[14]:
```
['AAAA', 'AA']
```

[15]:
```
re.findall('A{1,1}A{1,1}', grades) #las veces que el alumno saco dos A␣
 ↪consecutivas
#busca cuando hay dos A consecutivas
```

[15]:
```
['AA', 'AA', 'AA']
```

[16]:
```
re.findall('A{2}', grades) #si pongo un valor solo dentro de las llaves lo toma␣
 ↪como minimo y maximo
# si directamente no pongo las llaves lo toma como {1,1} por defecto
```

[16]:
```
['AA', 'AA', 'AA']
```

[17]:
```
#con esto podemos encontrar una tendencia negativa en las notas del estudiante:
re.findall('A{1,10}B{1,10}C{1,10}', grades)
```

[17]:
```
['AAAABC']
```

[18]:
```
#Lo anterior fue medio trucho porque el maximo fue asignado de manera␣
 ↪arbitraria y grande. Para eso existen:
#* para matchear 0 o mas veces
#? para matchear 1 o mas veces
#+ para matchear 1 o mas veces
```

[19]:
```
with open('ferpa.txt','r') as file: #abrimos un dataset
    wiki=file.read()                #lo leemos a traves de una variable llamada␣
 ↪wiki
wiki                                #lo mostramos
```

[19]:
```
'Overview[edit]\nFERPA gives parents access to their child\'s education records,
an opportunity to seek to have the records amended, and some control over the
disclosure of information from the records. With several exceptions, schools
must have a student\'s consent prior to the disclosure of education records
after that student is 18 years old. The law applies only to educational agencies
and institutions that receive funds under a program administered by the U.S.
Department of Education.\n\nOther regulations under this act, effective starting
January 3, 2012, allow for greater disclosures of personal and directory student
identifying information and regulate student IDs and e-mail addresses.[2] For
example, schools may provide external companies with a student\'s personally
identifiable information without the student\'s consent.[2]\n\nExamples of
situations affected by FERPA include school employees divulging information to
anyone other than the student about the student\'s grades or behavior, and
school work posted on a bulletin board with a grade. Generally, schools must
have written permission from the parent or eligible student in order to release
any information from a student\'s education record.\n\nThis privacy policy also
governs how state agencies transmit testing data to federal agencies, such as
the Education Data Exchange Network.\n\nThis U.S. federal law also gave students
18 years of age or older, or students of any age if enrolled in any post-
secondary educational institution, the right of privacy regarding grades,
enrollment, and even billing information unless the school has specific
permission from the student to share that specific type of information.\n\nFERPA
also permits a school to disclose personally identifiable information from
education records of an "eligible student" (a student age 18 or older or
enrolled in a postsecondary institution at any age) to his or her parents if the
student is a "dependent student" as that term is defined in Section 152 of the
Internal Revenue Code. Generally, if either parent has claimed the student as a
dependent on the parent\'s most recent income tax statement, the school may non-
consensually disclose the student\'s education records to both
parents.[3]\n\nThe law allowed students who apply to an educational institution
such as graduate school permission to view recommendations submitted by others
as part of the application. However, on standard application forms, students are
given the option to waive this right.\n\nFERPA specifically excludes employees
of an educational institution if they are not students.\n\nThe act is also
referred to as the Buckley Amendment, for one of its proponents, Senator James
L. Buckley of New York.\n\nAccess to public records[edit]\nThe citing of FERPA
to conceal public records that are not "educational" in nature has been widely
criticized, including by the act\'s primary Senate sponsor.[4] For example, in
the Owasso Independent School District v. Falvo case, an important part of the
debate was determining the relationship between peer-grading and "education
records" as defined in FERPA. In the Court of Appeals, it was ruled that
students placing grades on the work of other students made such work into an
```

```
"education record." Thus, peer-grading was determined as a violation of FERPA
privacy policies because students had access to other students\' academic
performance without full consent.[5] However, when the case went to the Supreme
Court, it was officially ruled that peer-grading was not a violation of FERPA.
This is because a grade written on a student\'s work does not become an
"education record" until the teacher writes the final grade into a grade
book.[6]\n\nStudent medical records[edit]\nLegal experts have debated the issue
of whether student medical records (for example records of therapy sessions with
a therapist at an on-campus counseling center) might be released to the school
administration under certain triggering events, such as when a student sued his
college or university.[7][8]\n\nUsually, student medical treatment records will
remain under the protection of FERPA, not the Health Insurance Portability and
Accountability Act (HIPAA). This is due to the "FERPA Exception" written within
HIPAA.[9]'
```

[20]:
```
re.findall('[a-zA-Z]{1,100}\[edit\]', wiki) #busco una lista de todos los␣
 ↪encabezados, diga lo que diga de la A a la Z (tanto
#en minusculas como mayusculas) y agrego '[edit\]' porque todos los encabezados␣
 ↪terminan asi. Importante el \ del medio
```

[20]:
```
['Overview[edit]', 'records[edit]', 'records[edit]']
```

[21]:
```
re.findall('[\w]{1,100}\[edit\]', wiki) #con el \w le indico que quiero letras,␣
 ↪numeros, simbolos, lo que sea.
```

[21]:
```
['Overview[edit]', 'records[edit]', 'records[edit]']
```

[22]:
```
re.findall('[\w]*\[edit\]', wiki) #* para matchear 0 o mas veces
```

[22]:
```
['Overview[edit]', 'records[edit]', 'records[edit]']
```

[23]:
```
re.findall('[\w ]*\[edit\]', wiki) #agregando espacio puedo obtener el titulo␣
 ↪entero
```

[23]:
```
['Overview[edit]',
 'Access to public records[edit]',
 'Student medical records[edit]']
```

[24]:
```
#entonces para obtener una lista de los titulos del texto hago un for:
for title in re.findall('[\w ]*\[edit\]', wiki):
    print(re.split('[\[]', title)[0]) #tambien funciona con '[]\[]''
```

```
Overview
Access to public records
Student medical records
```

18 Groups

[25]:
```
#los grupos se hacen con parentesis, puede hacer lo mismo que antes con groups:
re.findall('([\w ]*)(\[edit\])', wiki)
```

[25]:
```
[('Overview', '[edit]'),
 ('Access to public records', '[edit]'),
 ('Student medical records', '[edit]')]
```

[26]:
```
#si deseo una lista con cada uno de los valores matcheados puedo usar finditer:
for item in re.finditer('([\w ]*)(\[edit\])', wiki):
    print(item.groups())
```

```
('Overview', '[edit]')
('Access to public records', '[edit]')
('Student medical records', '[edit]')
```

[27]:
```
#si deseo solo una parte del grupo puedo hacer lo siguiente:
for item in re.finditer('([\w ]*)(\[edit\])', wiki):
    print(item.group(1)) #group(0) significa todo el match
```

```
Overview
Access to public records
Student medical records
```

[28]:
```
#tambien se pueden poner nombre a los grupos con la sintaxis: (?P<name>) y␣
 ↪luego usar .groupdict(). Ejemplo a continuacion:
for item in re.finditer('(?P<title>[\w ]*)(?P<edit_link>\[edit\])', wiki):
    print(item.groupdict()['title'])
```

```
Overview
Access to public records
Student medical records
```

[29]:
```
print(item.groupdict())
```

```
{'title': 'Student medical records', 'edit_link': '[edit]'}
```

19 Look-ahead and look-behind

[43]:
```
#en este caso el patron se aplica al texto antes y despues a lo que queremos␣
 ↪separar. Por ejemplo en los titulos queremos
#separar lo que viene antes del [edit], pero no nos interesa el [edit] por si␣
 ↪mismo. Se utiliza ?= para hacer un look ahead,
#es decir le indico que quiero la palabra que mirando para adelante encuentro e␣
 ↪[edit].
```