Capítulo 2: Laços e Listas

Adaptado e traduzido para o português por **Alisson Brito**, Laboratório de Engenharia de Sistemas e Robótica (LASER), da Universidade Federal da Paraíba (UFPB)

Material original: **Hans Petter Langtangen**, Simula Research Laboratory and University of Oslo, Dept. of Informatics

Data: **Junho, 2020**

Fazendo uma tabela de graus Celsius e Fahrenheit

```
-20  -4.0
-15   5.0
-10  14.0
 -5  23.0
  0  32.0
  5  41.0
 10  50.0
 15  59.0
 20  68.0
 25  77.0
 30  86.0
 35  95.0
 40 104.0
```

Como criar uma tabela dessa forma com programação?

Criando uma tabela: solução de iniciante

Nós sabemos como calcular para um valor:

In [ ]:

```
C = -20
F = 9.0/5*C + 32
print (C, F)
```

Basta repetir para os demais valores:

In [ ]:

```
C = -20;  F = 9.0/5*C + 32;  print (C, F)
C = -15;  F = 9.0/5*C + 32;  print (C, F)
# ...
C =  35;  F = 9.0/5*C + 32;  print (C, F)
C =  40;  F = 9.0/5*C + 32;  print (C, F)
```

- Trabalhoso e passivo de erro de digitação
  - Quanto o trabalho de programar se torna repetitivo, provavelmente há uma solução para automatizar o processo
  - O computador é extremamente eficiente para executar operações repetitivas
  - Para isso nós usamos os *laços (loops)*

O laço while

Executa uma sequência de instruções enquanto uma condição for verdadeira (true)

```
while condition:
            <statement 1>
            <statement 2>
            ...
        <first statement after loop>
```

- Todas instruções devem ser indentadas (atenção!!)
- O laço termina quando a indentação é finalizada

Laço while para criar a tabela

In [ ]:

```
print ("------------------") # table heading
C = -20                     # start value for C
dC = 5                      # increment of C in loop
while C <= 40:              # loop heading with condition
    F = (9.0/5)*C + 32      # 1st statement inside loop
    print (C, F)              # 2nd statement inside loop
    C = C + dC              # last statement inside loop
print ("------------------") # end of table line
```

Expressões booleanas retornam True ou False

Examplos: $C = 40$, $C \neq 40$, $C \geq 40$, $C > 40$, $C < 40$.

```
C == 40  # observe que == é expressão booleana, e C = 40 é uma atribuição!
        C != 40
        C >= 40
        C >  40
        C <  40
```

Podemos testar as expressões booleanas em Python:

In [ ]:

```
C = 41
C != 40
```

In [ ]:

```
C < 40
```

In [ ]:

```
C == 41
```

Combinando expressões booleanas

Várias expressões booleandas podem ser combinadas com "and" e "or":

```
while condition1 and condition2:
            ...

        while condition1 or condition2:
            ...
```

Regra 1: `C1 and C2` é `True` se ambos `C1` e `C2` são `True`

Regra 2: `C1 or C2` é `True` se `C1` ou `C2` for `True`

In [ ]:

```
x = 0;  y = 1.2
x >= 0 and y < 1
```

In [ ]:

```
x >= 0 or y < 1
```

In [ ]:

```
x > 0 or y > 1
```

In [ ]:

```
x > 0 or not y > 1
```

In [ ]:

```
-1 < x <= 0   #  -1 < x e x <= 0
```

In [ ]:

```
not (x > 0 or y > 0)
```

Listas em Python são objetos que armazenam uma sequência de outros objetos

Até o momento, uma variável armazena um único valor.

Listas armazenam uma sequência de vários valores.

Os valores de nossa tabela poderiam ser armazenados assim:

In [ ]:

```
C1 = -20
C2 = -15
C3 = -10
# ...
C13 = 40
```

Lembre-se: tarefa chata e repetitiva!

Há então uma forma melhor de fazer:

In [ ]:

```
C = [-20, -15, -10, -5, 0, 5, 10, 15, 20, 25, 30, 35, 40]
```

Agora temos uma única variável, `C` que armazena todos os valores

Operações com listas: iniciando e indexando

Iniciando com cochetes e vírgula entre os objetos:

In [ ]:

```
L1 = [-91, 'um texto', 7.2, 0]
```

Elementos são accessados via um índice: `L1[3]` (índice=3).

Os índices das listas iniciam com zero: `0` , `1` , `2` , ... `len(L1)-1` .

In [ ]:

```
mylist = [4, 6, -3.5]
print (mylist[0])
```

In [ ]:

```
print (mylist[1])
```

In [ ]:

```
print (mylist[2])
```

In [ ]:

```
len(mylist)  # length of list
```

Operações com Listas: append, extend, insert, delete

In [ ]:

```
C = [-10, -5, 0, 5, 10, 15, 20, 25, 30]
C.append(35)   # adiciona elemento 35 no final
C
```

In [ ]:

```
C = C + [40, 45]     # extende a lista C com mais dois elementos no final
C
```

In [ ]:

```
C.insert(0, -15)     # insere -15 no índice 0
C
```

In [ ]:

```
print (C)
del C[2]             # remove o 3o elemento
print(C)
```

In [ ]:

```
del C[2]             # remove o elmento índice 2 (3o elemento)
C
```

In [ ]:

```
len(C)               # comprimento da lista
```

Operações com Listas: busca, índices negativos

In [ ]:

```
C.index(20)   # índice do 1o elemento com valor 20
```

In [ ]:

```
35 in C       # o elemento 10 está presente em C?
```

In [ ]:

```
C[-1]         # último elemento da lista
```

In [ ]:

```
C[-2]         # o penúltimo elemento da lista
```

In [ ]:

```
somelist = ['book.tex', 'book.log', 'book.pdf']
texfile, logfile, pdf = somelist  # atribui elementos diretamente para variáveis
texfile
```

In [ ]:

```
logfile
```

In [ ]:

```
pdf
```

Laço For para percorrer uma Lista

Use um laço *for* quando precisar acessar todos elementos de uma lista:

In [ ]:

```
degrees = [0, 10, 20, 40, 100]
for C in degrees:
    print ('Celsius degrees:', C)
    F = 9/5.*C + 32
    print ('Fahrenheit:', F)
print ('The degrees list has', len(degrees), 'elements')
```

Não esqueça a indentação!

In [ ]:

```
degrees = [0, 10, 20, 40, 100]
for C in degrees:
    print (C)
print ('The degrees list has', len(degrees), 'elements')
```

Criando a tabela com um laço For

In [ ]:

```
Cdegrees = [-20, -15, -10, -5, 0, 5, 10, 15,
            20, 25, 30, 35, 40]
for C in Cdegrees:
    F = (9.0/5)*C + 32
    print (C, F)
```

Obs: `print C, F` nos dá uma saída feia. Use a sintaxe printf para melhorar a saída em 2 colunas:

In [ ]:

```
print ('%5d %5.1f'%(C, F))
```

Resultado:

```
-20  -4.0
-15   5.0
-10  14.0
 -5  23.0
  0  32.0
  ......
 35  95.0
 40 104.0
```

Um laço for pode sempre ser traduzido para um laço while

Mas o contrário não é verdadeiro

Versão da tabela com laço while

In [ ]:

```
Cdegrees = [-20, -15, -10, -5, 0, 5, 10,
            15, 20, 25, 30, 35, 40]
index = 0
while index < len(Cdegrees):
    C = Cdegrees[index]
    F = (9.0/5)*C + 32
    print ('%5d %5.1f' % (C, F))
    index +=1
```

Implementando um somatório com laços

$$S = \sum_{i=1}^{N} i^2$$

In [ ]:

```
N = 14

S = 0
for i in range(1, N+1):
    S += i**2
print (S)
```

Ou (menos comum):

In [ ]:

```
S = 0
i = 1
while i <= N:
    S += i**2
    i += 1
print S
```

Em problemas científicos, somatórios aparecem muito! Então não esqueçam!

Armazendo as colunas da tabela como listas

Vamos colocar os valores em Fahrenheit também em lista:

In [ ]:

```
Cdegrees = [-20, -15, -10, -5, 0, 5, 10,
            15, 20, 25, 30, 35, 40]
Fdegrees = []            # inicia com uma lista vazia
for C in Cdegrees:
    F = (9.0/5)*C + 32
    Fdegrees.append(F)   # insere novo elemento em Fdegrees
print (Fdegrees)
```

Laço For com índices da lista

Laços For geralmente varrem os elementos da lista:

```
for element in somelist:
            # processa a variável "element"
```

Alternativamente, podemos acessar os elementos pelos seus índices:

```
for i in range(0, len(somelist), 1):
            element = somelist[i]
            # processa somelist[i] diretamente
```

`range(start, stop, inc)` gera uma lista de inteiros `start` , `start+inc` , `start+2*inc` , assim em diante, *mas não inclui*, `stop` . `range(stop)` é a forma curta para `range(0, stop, 1)` .

In [ ]:

```
range(3)          # = range(0, 3, 1)
```

In [ ]:

```
range(2, 8, 3)
```

Como podemos mudar um elemento em uma lista?

Para adicionar 2 a todos elementos de uma lista:

In [ ]:

```
v = [-1, 1, 10]
for e in v:
    e = e + 2
```

In [ ]:

```
print (v)
```

Qual é o problema?

Dentro do loop, `e` é uma variável inteira (`int`), e não uma lista. Dessa forma, `v` permanece inalterada.

Modificar elemento de uma lista necessita de acesso pelo índice

In [ ]:

```
v[1] = 4    # atribui 4 ao 2o element (índice 1) em v
v
```

In [ ]:

```
for i in range(len(v)):
    v[i] = v[i] + 2
```

In [ ]:

```
v
```

Criação compacta de listas

Examplo: computar duas listas em um laço for.

In [ ]:

```
n = 16
Cdegrees = []; Fdegrees = []  # listas vazias

for i in range(n):
    Cdegrees.append(-5 + i*0.5)
    Fdegrees.append((9.0/5)*Cdegrees[i] + 32)
print (Cdegrees)
print (Fdegrees)
```

Python possui uma forma ainda mais compacta, chamada *list comprehension*, para gerar lista com laço "for":

In [ ]:

```
Cdegrees = [-5 + i*0.5 for i in range(n)]
Fdegrees = [(9.0/5)*C + 32 for C in Cdegrees]
print (Cdegrees)
print (Fdegrees)
```

Forma geral:

```
somelist = [expression for element in somelist]
```

onde `expression` diz respeito à `element`

Percorrendo múltiplas listas simultaneamente com "zip"

Como percorrer 2 listas?

Solução 1: laço com índices

In [ ]:

```
for i in range(len(Cdegrees)):
    print (Cdegrees[i], Fdegrees[i])
```

Solução 2: usar a construção `zip` (é bem "Pythonic"):

In [ ]:

```
for C, F in zip(Cdegrees, Fdegrees):
    print (C, F)
```

Exemplo com três listas:

In [ ]:

```
l1 = [3, 6, 1];  l2 = [1.5, 1, 0];  l3 = [9.1, 3, 2]
for e1, e2, e3 in zip(l1, l2, l3):
    print (e1, e2, e3)
```

Listas encadeadas: listas de listas

- Uma lista pode conter *qualquer* objeto, que pode ser outra lista
- Ao invés de termos 2 listas separadas (uma para cada coluna), podemos unir as duas em uma nova lista:

In [99]:

```
Cdegrees = range(-20, 41, 5)
Fdegrees = [(9.0/5)*C + 32 for C in Cdegrees]

table1 = [Cdegrees, Fdegrees]  # lista com duas listas (veja como uma tabela)

print (table1[0])     # the Cdegrees list
print (table1[1])    # the Fdegrees list
print (table1[0][2]) # the 3rd element in Cdegrees
print (table1[1][3]) # o 4o elemento em Cdegrees
```

```
range(-20, 41, 5)
[-4.0, 5.0, 14.0, 23.0, 32.0, 41.0, 50.0, 59.0, 68.0, 77.0, 86.0, 9
5.0, 104.0]
-10
23.0
```

Tabela de colunas vs. tabela de linhas

- A tabela anterior `table1 = [Cdegrees,Fdegrees]` é uma tabela com duas linhas (tabela de colunas)
- Vamos agora fazer uma tabela de linhas, com duas colunas `[C,F]` :

In [100]:

```
table2 = []
for C, F in zip(Cdegrees, Fdegrees):
    row = [C, F]
    table2.append(row)

# mais compacta com 'list comprehension':
table2 = [[C, F] for C, F in zip(Cdegrees, Fdegrees)]
table2
```

Out[100]:

```
[[-20, -4.0],
 [-15, 5.0],
 [-10, 14.0],
 [-5, 23.0],
 [0, 32.0],
 [5, 41.0],
 [10, 50.0],
 [15, 59.0],
 [20, 68.0],
 [25, 77.0],
 [30, 86.0],
 [35, 95.0],
 [40, 104.0]]
```

Percorrendo elementos em uma lista encadeada:

```
for C, F in table2:
            # opera com C e F para cada linha em table2

        # or
        for row in table2:
            C, F = row
            ...
```

Ilustração de tabela de linhas

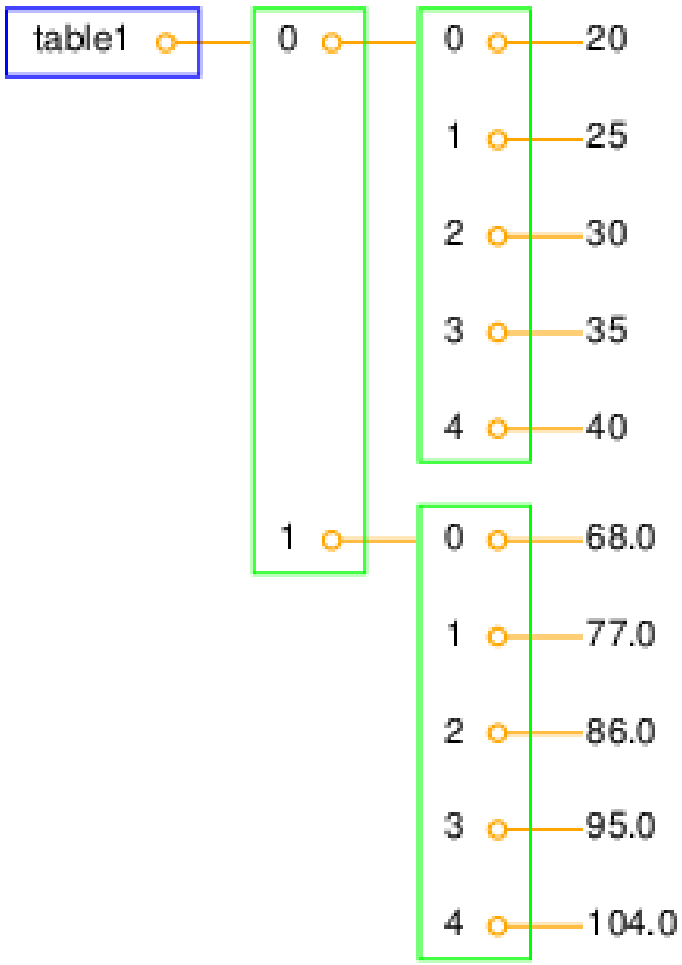
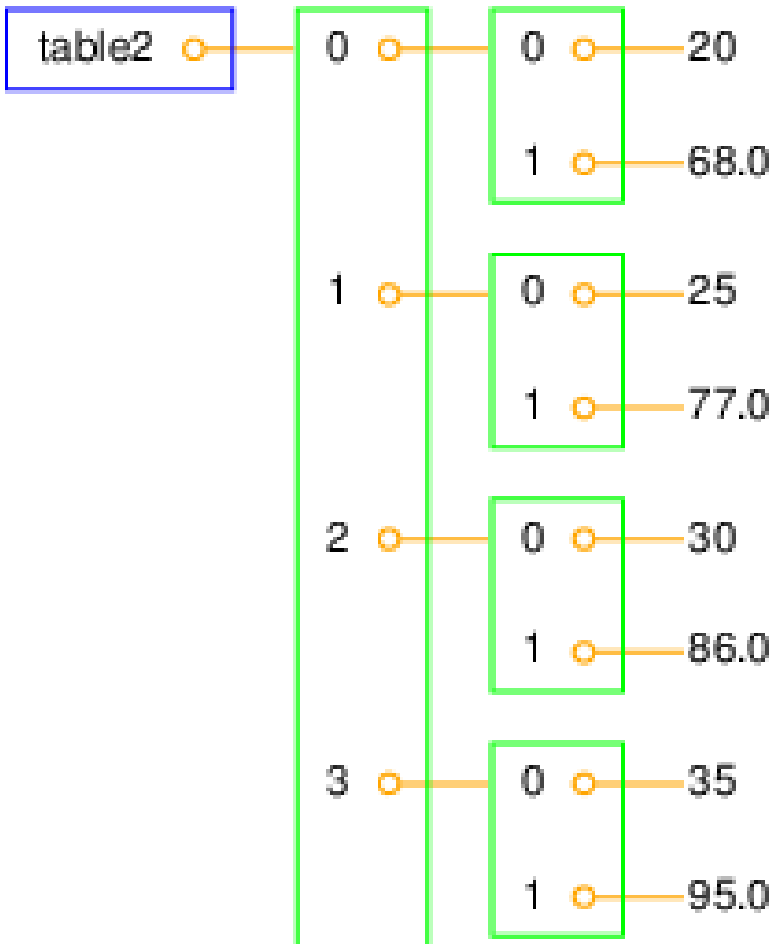


Ilustração de tabela de colunas



In [101]:

```
A = [2, 3.5, 8, 10, 12]
A[2:]   # do índice 2 até o final da lista
```

Out[101]:

```
[8, 10, 12]
```

In [102]:

```
A[1:3]  # do índice 1 até (mas não incluído) o índice 3
```

Out[102]:

```
[3.5, 8]
```

In [103]:

```
A[:3]   # do início até o índice 3 (mas não incluído)
```

Out[103]:

```
[2, 3.5, 8]
```

In [104]:

```
A[1:-1] # do índice 1 até o penúltimo elemento
```

Out[104]:

```
[3.5, 8, 10]
```

In [105]:

```
A[:]    # a lista completa
```

Out[105]:

```
[2, 3.5, 8, 10, 12]
```

Obs: sublistas são cópias da lista original!

O que este trecho de código faz?

In [114]:

```
for C, F in table2[Cdegrees.index(-15):Cdegrees.index(40)]:
    print (C, F)
```

```
-15 5.0
-10 14.0
-5 23.0
0 32.0
5 41.0
10 50.0
15 59.0
20 68.0
25 77.0
30 86.0
35 95.0
```

* É um laço sobre a lista `table2`

- Ele percorre do índice `Cdegrees.index(-15)` até o índice `Cdegrees.index(40)` , ou seja dos índices correspondentes aos valores -15 e 40 (não incluindo o próprio 40).

Interação em listas encadeadas

Listas com várias dimensões: `somelist[i1][i2][i3]...`

Loops em sublistas usando índices:

```
for i1 in range(len(somelist)):
            for i2 in range(len(somelist[i1])):
                for i3 in range(len(somelist[i1][i2])):
                    for i4 in range(len(somelist[i1][i2][i3])):
                        value = somelist[i1][i2][i3][i4]
                        # work with value
```

Loops em sublistas:

```
    for sublist1 in somelist:
                for sublist2 in sublist1:
                    for sublist3 in sublist2:
                        for sublist4 in sublist3:
                            value = sublist4
                            # work with value
```

Iteração sobre uma específica lista encadeada

In [119]:

```
L = [[9, 7], [-1, 5, 6]]
for row in L:
    for column in row:
        print (column)
```

```
9
7
-1
5
6
```

Simule este programa na mão!

Questão

Como podemos indexar o elemento de valor 5?

Tuplas são listas constantes

Tuplas são listas constantes não podem ser modificadas:

In [121]:

```
t = (2, 4, 6, 'temp.pdf')    # define a tupla (cria-se com parêntesis)
t =  2, 4, 6, 'temp.pdf'     # pode remover os parentesis
t[1] = -1
```

```
---------------------------------------------------------------------
-------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-121-8d4ecdb74235> in <module>
      1 t = (2, 4, 6, 'temp.pdf')    # define a tupla (cria-se com p
arêntesis)
      2 t =  2, 4, 6, 'temp.pdf'     # pode remover os parentesis
----> 3 t[1] = -1

TypeError: 'tuple' object does not support item assignment
```

In [122]:

```
t.append(0)
```

```
---------------------------------------------------------------------
-------
AttributeError                            Traceback (most recent cal
l last)
<ipython-input-122-027f59be7fb0> in <module>
----> 1 t.append(0)

AttributeError: 'tuple' object has no attribute 'append'
```

In [123]:

```
del t[1]
```

```
---------------------------------------------------------------------
-------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-123-77cea8bc7ee1> in <module>
----> 1 del t[1]

TypeError: 'tuple' object doesn't support item deletion
```

Podem-se fazer outras operações com tuplas, semelhantes às listas:

In [124]:

```
t = t + (-1.0, -2.0)          # somar duas tuplas
t
```

Out[124]:

```
(2, 4, 6, 'temp.pdf', -1.0, -2.0)
```

In [125]:

```
t[1]                          # indexar
```

Out[125]:

```
4
```

In [126]:

```
t[2:]                         # subtupla/fatias
```

Out[126]:

```
(6, 'temp.pdf', -1.0, -2.0)
```

In [127]:

```
6 in t                        # checar se elemento faz parte da tupla
```

Out[127]:

```
True
```

Por que usar tuplas, se listas possuem mais funcionalidades?

- Tuplas são constantes e então protegidas de mudanças acidentais
- Tuplas são mais rápidas do que listas
- Tuplas são amplamente usdas em Python (então você precisa conhecer!)
- Tuplas (listas não!) podem ser usadas como chaves para dicionários (vamos estudar dicionários mais a frente)

```
while condition:
            <block of statements>

        for element in somelist:
            <block of statements>
```

Operações com Listas

Construction	Meaning
`a = []`	initialize an empty list
`a = [1, 4.4, 'run.py']`	initialize a list
`a.append(elem)`	add `elem` object to the end
`a + [1,3]`	add two lists
`a.insert(i, e)`	insert element `e` before index `i`
`a[3]`	index a list element
`a[-1]`	get last list element
`a[1:3]`	slice: copy data to sublist (here: index 1, 2)
`del a[3]`	delete an element (index `3`)
`a.remove(e)`	remove an element with value `e`
`a.index('run.py')`	find index corresponding to an element's value
`'run.py' in a`	test if a value is contained in the list
`a.count(v)`	count how many elements that have the value `v`
`len(a)`	number of elements in list `a`
`min(a)`	the smallest element in `a`
`max(a)`	the largest element in `a`
`sum(a)`	add all elements in `a`
`sorted(a)`	return sorted version of list `a`
`reversed(a)`	return reversed sorted version of list `a`
`b[3][0][2]`	nested list indexing
`isinstance(a, list)`	is `True` if `a` is a list
`type(a) is list`	is `True` if `a` is a list

Um exemplo sumarizador

`src/misc/Oxford_sun_hours.txt` : arquivo com a quantidade de horas de incidência solar em Oxford, UK, para cada mês desde janeiro de 1929 até dezembro de 2009. Cada linha é um ano, cada coluna é um

```
[
        [43.8, 60.5, 190.2, ...],
        [49.9, 54.3, 109.7, ...],
        [63.7, 72.0, 142.3, ...],
        ...
        ]
```

Exercício:

Realize e envie pelo formulário a ser publicado as soluções para as quesões 2 e 3:

- 1) Calcule a média das horas de incidência do Sol para cada mês durante o período total (1929 - 2009)
- 2) Qual mês teve a maior incidência de luz de acordo com o calculado no exercício anterior?
- 3) Para cada década, 1930-1939, 1949-1949, …, 2000-2009, calcule a média das horas de incidência solar por dia em janeiro e dezembro.

Resolução exercício 1)

In [ ]:

```
data = [
[43.8, 60.5, 190.2, 144.7, 240.9, 210.3, 219.7, 176.3, 199.1, 109.2, 78.7, 67.0
],
[49.9, 54.3, 109.7, 102.0, 134.5, 211.2, 174.1, 207.5, 108.2, 113.5, 68.7, 23.3
],
[63.7, 72.0, 142.3, 93.5, 150.1, 158.7, 127.9, 135.5, 92.3, 102.5, 62.4, 38.5],
[51.0, 57.9, 133.4, 110.9, 112.4, 199.3, 124.0, 178.3, 102.1, 100.7, 55.7, 58.0
],
[69.5, 94.3, 187.6, 152.5, 170.2, 226.9, 237.6, 242.7, 177.3, 101.3, 53.9, 59.0
],
[65.9, 96.6, 122.5, 124.9, 216.3, 192.7, 269.3, 184.9, 149.1, 81.5, 48.7, 31.3],
[48.1, 62.0, 121.5, 127.3, 188.5, 196.3, 274.3, 199.9, 144.7, 102.6, 65.4, 48.9
],
[43.4, 89.2, 71.4, 133.2, 179.5, 166.2, 119.2, 184.7, 79.3, 103.1, 48.9, 62.3],
[50.9, 66.6, 99.7, 103.1, 185.0, 181.3, 140.1, 202.3, 143.0, 79.1, 65.9, 41.2],
[41.2, 66.9, 172.3, 180.9, 144.9, 190.6, 133.5, 151.3, 110.9, 118.1, 70.0, 52.4
],
[46.4, 104.9, 86.2, 171.7, 184.9, 227.9, 139.7, 153.7, 147.0, 94.3, 41.1, 46.0],
[83.1, 22.8, 128.3, 118.1, 215.4, 273.4, 165.1, 199.5, 179.5, 95.5, 76.8, 46.5],
[41.7, 67.9, 118.7, 106.9, 141.9, 210.3, 227.5, 163.7, 123.7, 120.2, 47.1, 46.9
],
[45.1, 53.9, 69.4, 202.5, 209.4, 234.0, 150.1, 132.7, 124.5, 84.6, 57.8, 51.0],
[54.7, 79.3, 132.9, 166.6, 244.1, 192.9, 196.7, 178.3, 142.5, 84.9, 72.3, 49.5],
[41.2, 62.4, 142.7, 147.0, 235.6, 170.3, 97.5, 185.2, 143.8, 102.0, 49.3, 64.1],
[51.5, 65.7, 152.6, 209.1, 156.1, 182.4, 159.0, 144.8, 64.9, 111.7, 31.0, 46.6],
[49.9, 78.7, 107.2, 203.3, 162.9, 149.8, 197.6, 134.8, 98.5, 79.3, 42.9, 74.7],
[59.5, 26.3, 70.9, 150.5, 147.3, 185.9, 144.5, 274.9, 159.9, 107.3, 75.4, 37.9],
[45.7, 92.9, 160.2, 205.2, 237.1, 124.2, 174.7, 133.7, 146.4, 93.7, 68.6, 65.4],
[51.0, 115.1, 112.5, 182.5, 233.3, 242.1, 262.5, 210.3, 151.1, 125.0, 76.2, 65.4
],
[40.6, 67.5, 138.8, 163.7, 174.1, 244.5, 174.0, 171.1, 112.7, 96.6, 56.9, 55.3],
[48.9, 58.6, 92.6, 200.4, 152.1, 251.9, 216.7, 174.7, 110.8, 105.6, 75.1, 69.8],
[94.1, 96.7, 105.0, 178.2, 207.0, 217.6, 194.0, 180.5, 140.3, 105.0, 72.1, 77.7
],
[42.5, 75.9, 140.7, 183.3, 223.0, 139.7, 203.4, 237.4, 151.7, 84.1, 54.4, 28.4],
[75.7, 79.7, 107.9, 202.4, 145.9, 157.1, 157.1, 123.5, 168.8, 94.5, 60.1, 54.5],
[40.1, 86.3, 161.4, 173.7, 217.5, 155.3, 268.3, 188.0, 153.1, 119.7, 71.5, 47.3
],
[50.3, 78.9, 149.7, 158.7, 246.6, 145.0, 168.0, 161.4, 94.3, 116.5, 77.9, 18.2],
[50.8, 83.1, 110.2, 168.0, 205.6, 297.1, 157.9, 170.5, 102.6, 92.9, 76.4, 62.3],
[54.6, 55.4, 110.7, 145.2, 196.0, 145.7, 188.1, 119.6, 118.0, 93.7, 51.8, 29.5],
[85.8, 65.5, 102.0, 153.8, 228.0, 226.3, 272.7, 245.6, 213.9, 144.2, 70.6, 45.0
],
[37.8, 82.3, 78.0, 164.9, 182.3, 274.9, 129.7, 147.1, 122.8, 60.9, 73.4, 54.5],
[43.6, 65.1, 173.2, 86.9, 225.2, 231.2, 196.5, 185.7, 135.8, 118.2, 63.4, 76.5],
[70.0, 70.6, 126.3, 143.3, 177.5, 280.3, 137.3, 154.5, 142.3, 108.8, 32.7, 72.6
],
[58.9, 66.4, 85.8, 119.1, 193.4, 199.4, 188.2, 142.6, 129.7, 78.8, 60.4, 49.8],
[37.2, 57.3, 65.9, 128.5, 190.8, 156.1, 214.7, 217.7, 210.4, 134.5, 55.0, 51.1],
[83.7, 31.1, 137.7, 141.6, 179.6, 188.7, 122.8, 181.2, 122.9, 109.9, 77.4, 71.9
],
[42.5, 41.5, 121.5, 81.5, 234.9, 199.0, 149.7, 188.6, 168.0, 90.4, 61.0, 41.7],
[64.2, 88.2, 174.6, 130.8, 184.2, 232.0, 234.4, 167.1, 116.5, 95.1, 69.2, 70.6],
[50.0, 54.0, 148.5, 184.5, 155.0, 206.6, 136.2, 124.0, 114.9, 66.5, 47.9, 35.9],
[40.0, 78.1, 70.5, 221.3, 161.9, 276.9, 243.8, 157.5, 97.4, 112.0, 84.6, 35.6],
[34.5, 115.9, 120.8, 132.7, 224.8, 270.9, 192.4, 185.6, 157.3, 106.2, 64.7, 43.8
],
[42.1, 69.5, 106.0, 122.9, 228.9, 143.5, 259.3, 134.2, 166.5, 135.2, 102.0, 29.8
],
```

```
[41.8, 27.3, 144.0, 117.6, 141.9, 150.4, 168.7, 160.9, 129.1, 91.6, 80.6, 47.6],
[38.8, 74.1, 150.7, 167.7, 168.0, 249.5, 171.1, 192.0, 153.9, 95.1, 89.1, 62.9],
[56.3, 58.3, 101.7, 142.1, 191.4, 206.2, 187.8, 198.7, 146.5, 105.4, 52.9, 58.8
],
[44.7, 57.8, 72.7, 131.4, 159.1, 301.0, 242.4, 218.6, 147.0, 120.7, 85.1, 34.4],
[70.3, 42.6, 107.8, 148.7, 172.0, 261.4, 254.2, 257.4, 118.2, 43.6, 54.1, 58.6],
[35.4, 74.4, 87.2, 157.9, 217.5, 123.2, 193.6, 123.4, 101.8, 107.3, 102.4, 45.2
],
[53.6, 58.9, 128.1, 113.6, 202.8, 171.7, 146.4, 157.4, 159.3, 87.5, 77.3, 34.3],
[72.0, 67.3, 92.9, 126.4, 190.9, 166.6, 192.2, 167.4, 171.0, 117.0, 70.0, 59.7],
[84.0, 58.8, 86.7, 165.4, 228.7, 186.7, 168.9, 169.0, 136.4, 111.8, 61.4, 64.4],
[50.9, 75.3, 57.8, 110.4, 98.3, 122.8, 129.0, 199.8, 157.3, 101.9, 43.7, 57.5],
[55.0, 33.8, 144.7, 164.4, 187.2, 148.4, 151.4, 159.8, 141.0, 66.3, 68.5, 60.4],
[54.9, 74.0, 89.5, 150.5, 126.8, 180.3, 257.5, 214.5, 92.4, 119.7, 44.3, 62.9],
[86.1, 66.0, 48.8, 236.8, 143.4, 244.3, 249.4, 199.8, 99.6, 88.6, 53.8, 57.6],
[51.1, 78.0, 112.4, 138.3, 178.3, 165.0, 216.0, 164.9, 143.3, 100.8, 86.1, 44.2
],
[76.4, 71.2, 127.3, 139.6, 205.6, 222.7, 201.2, 147.0, 171.6, 119.7, 77.9, 64.8
],
[68.8, 67.8, 111.6, 158.7, 168.7, 129.3, 179.4, 158.2, 132.3, 109.5, 43.9, 42.9
],
[47.7, 103.7, 85.2, 132.0, 178.1, 142.3, 138.8, 178.8, 136.9, 120.0, 91.4, 46.7
],
[68.2, 107.0, 100.9, 133.9, 300.8, 244.4, 280.4, 269.5, 141.6, 90.8, 104.1, 26.5
],
[58.3, 95.7, 144.1, 234.4, 285.0, 121.1, 268.5, 236.6, 164.7, 124.4, 83.7, 58.8
],
[66.9, 60.0, 87.2, 156.6, 142.9, 150.0, 217.3, 241.4, 165.4, 79.4, 57.4, 58.4],
[46.8, 67.3, 73.3, 139.2, 262.7, 212.2, 164.0, 173.6, 120.2, 101.3, 61.5, 47.2],
[38.0, 54.7, 135.0, 111.9, 196.7, 231.4, 190.0, 238.3, 107.7, 120.0, 76.3, 55.0
],
[87.0, 77.6, 127.6, 177.7, 162.1, 254.9, 248.3, 191.8, 113.1, 137.5, 46.7, 68.2
],
[61.8, 74.9, 198.7, 190.1, 233.5, 194.4, 247.6, 285.1, 135.3, 139.9, 78.1, 40.9
],
[29.3, 103.4, 76.4, 148.3, 185.7, 290.7, 256.6, 211.6, 125.3, 130.8, 101.0, 55.5
],
[51.4, 64.2, 150.2, 189.9, 261.4, 137.1, 231.7, 172.0, 169.7, 153.8, 47.1, 55.7
],
[64.0, 113.0, 77.5, 105.8, 199.8, 114.0, 157.0, 225.0, 133.8, 94.5, 66.3, 38.1],
[51.1, 81.3, 97.4, 147.6, 153.6, 202.1, 235.4, 159.2, 155.2, 144.8, 81.1, 60.9],
[82.1, 104.9, 112.6, 143.4, 189.8, 164.6, 161.2, 209.4, 126.1, 83.9, 69.2, 51.9
],
[83.3, 85.0, 74.1, 148.2, 198.3, 226.8, 206.1, 184.1, 123.0, 100.9, 86.9, 79.2],
[44.4, 80.5, 101.1, 210.0, 177.5, 163.3, 178.8, 166.2, 167.1, 104.8, 52.3, 41.3
],
[87.7, 94.4, 154.8, 169.8, 191.2, 213.6, 192.0, 228.4, 175.3, 134.8, 78.9, 53.6
],
[62.7, 79.1, 101.5, 150.3, 195.5, 223.6, 169.5, 194.1, 174.4, 102.4, 52.4, 58.3
],
[65.4, 66.3, 79.3, 136.3, 226.4, 177.6, 192.0, 235.7, 155.4, 92.0, 88.0, 55.7],
[54.9, 73.1, 95.5, 152.5, 165.7, 246.2, 303.7, 167.2, 156.5, 109.0, 101.2, 42.7
],
[79.8, 67.6, 165.4, 210.7, 165.5, 149.0, 195.1, 209.2, 142.6, 102.5, 86.9, 57.2
],
[62.4, 124.1, 115.2, 161.2, 173.2, 223.8, 198.5, 141.8, 113.5, 132.2, 67.0, 73.5
],
[69.3, 64.5, 161.4, 168.4, 226.1, 203.3, 212.3, 190.6, 163.7, 109.7, 73.5, 61.5
],
]
monthly_mean = [0]*12
```

```
for month in range(1, 13):
    m = month - 1   # corresponding list index (starts at 0)
    s = 0           # sum
    n = 2009 - 1929 + 1  # no of years
    for year in range(1929, 2010):
        y = year - 1929  # corresponding list index (starts at 0)
        s += data[y][m]
    monthly_mean[m] = s/n
month_names = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
               'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
# nice printout:
for name, value in zip(month_names, monthly_mean):
    print ('%s: %.1f' % (name, value))
```