PDSP 2026, Lecture 08, 3 September 2026

Strings

- Also sequences, of characters
- String values can be enclosed in single or double quotes
  - `'Chennai'` or `"Chennai"`
- Allows a value that has a quote to be easily embedded
  - `"Fermat's Last Theorem"`
  - `'He said, "Thank you!"`
- If you need to use both single and double quotes inside the string, use a triple quote!
  - `'''He said, "That's great!"'''`

In [1]:
```
s = '''He said, "That's great!"'''
```

In [2]:
```
s
```

Out[2]:
```
'He said, "That\'s great!"'
```

- Python prefers to render strings using single quote
- Embedded quotes are "escaped" using `\` to remove their special meaning

- Like lists and tuples, can access elements of a string by position, or by slices

In [3]:
```
s = "hello"
```

In [4]:
```
s[1]
```

Out[4]:
```
'e'
```

In [5]:
```
s[2:4]
```

Out[5]:
```
'll'
```

- Some languages have a separate type `char` for a single character
  - A string is then a sequence of `char`
- In Python, there is only the string type `str`
  - A single character is the same as a string of length 1

In [6]:
```
s[1] == "e"  # Logically speaking, s[1] is a single character
```

Out[6]:
```
True
```

- Concatenate strings using `+`

In [7]:
```
s = "hello"
t = "there"
```

In [8]:
```
s+t, s, t
```

Out[8]:
```
('hellothere', 'hello', 'there')
```

- Notice that `s+t` does not have any character between `s` and `t`
- If we want a separator, we need to add it ourselves

In [9]:
```
s + " " + t
```

Out[9]:
```
'hello there'
```

- Like tuples, cannot update parts of a string directly

In [10]:
```
s[3] = "p"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[10], line 1
----> 1 s[3] = "p"

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

- Instead, assemble a new string from the old one

In [11]:
```
s = s[0:3] + "p" + s[4:]
```

In [12]:
```
s
```

Out[12]:
```
'helpo'
```

In [13]:
```
s[0:3] + s[4]
```

Out[13]:
```
'helo'
```

- Notice that `s[0:3]`, which is a string, is compatible with `s[4]`, which is a single item in the string
- This is different from lists and tuples, where the items in the list are one level "lower"

In [14]:
```
l = [1,2,3,4,5]
t = (1,2,3,4,5)
```

In [15]:
```
l[0:3] + l[4]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[15], line 1
----> 1 l[0:3] + l[4]

TypeError: can only concatenate list (not "int") to list
```

In [16]:
```
t[0:3] + t[4]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[16], line 1
----> 1 t[0:3] + t[4]

TypeError: can only concatenate tuple (not "int") to tuple
```

- Can iterate over strings and check membership, like lists
- For iteration, no distinction between a string `"xyz"` and the list `["x","y","z"]`

In [17]:
```
def vowel(c):
    return(c in "aeiou")
```

In [18]:
```
vowel("a"),vowel("b"),
```

Out[18]:
```
(True, False)
```

- For strings, `x in s` is interpreted as substring membership

In [19]:
```
vowel("ae"),vowel("ai"),vowel("io")
```

Out[19]:
```
(True, False, True)
```

- This would not work for lists or tuples
- For instance `[1,2] in [1,2,3]` would return `False`
- But `[1,2] in [[1,2],[2,3]]` would return `True`

In [20]:
```
[1,2] in [1,2,3], [1,2] in [[1,2],[2,3]]
```

Out[20]:
```
(False, True)
```

- Standard example of filtered iteration

In [21]:
```
def countvowels(s):
    count = 0
    for c in s:
        if vowel(c):
            count = count+1
    return(count)
```

In [22]:
```
countvowels("hello")
```

Out[22]:
```
2
```

- `x in s` expects `x` to be a string if `s` is a string

In [23]:
```
vowel(7)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[23], line 1
----> 1 vowel(7)

Cell In[17], line 2, in vowel(c)
      1 def vowel(c):
----> 2     return(c in "aeiou")

TypeError: 'in <string>' requires string as left operand, not int
```

In [24]:
```
vowel("7")
```

Out[24]:
```
False
```

In [25]:
```
vowel("there")
```

Out[25]:
```
False
```

- We have seen that `list()` and `int()` can be use to convert values from one type to another
- Likewise `str()` converts its argument to a string
  - Almost any value converts sensibly into a "readable" representation
  - `print(v)` implicitly converts `v` to `str(v)` to display on screen

In [26]:
```
str([1,2,3])
```

Out[26]:
```
'[1, 2, 3]'
```

In [27]:
```
str(77)
```

Out[27]:
```
'77'
```

In [28]:
```
str([1,  2,3])
```

Out[28]:
```
'[1, 2, 3]'
```

In [29]:
```
print(str([1,  2,3]))
```

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

In [30]:
```
print([1,2,3]) # via str([1,2,3])
```

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

In [31]:
```
str(range(5))
```

Out[31]:
```
'range(0, 5)'
```

In [32]:
```
s = str(range(5))
s[2]
```

Out[32]:
```
'n'
```

In [33]:
```
str(range(0,5,1))
```

Out[33]:
```
'range(0, 5)'
```

- Can convert a string to a number if the contents can be intepreted sensibly

In [34]:
```
int('77')
```

Out[34]:
```
77
```

In [35]:
```
int('hello')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[35], line 1
----> 1 int('hello')

ValueError: invalid literal for int() with base 10: 'hello'
```

In [36]:
```
int('77.5')  # 77.5 is a number, but not an int
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[36], line 1
----> 1 int('77.5')  # 77.5 is a number, but not an int

ValueError: invalid literal for int() with base 10: '77.5'
```

In [37]:
```
int('77.0')  # Even if the number is semantically an int, the decimal point disallows
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[37], line 1
----> 1 int('77.0')  # Even if the number is semantically an int, the decimal point di
sallows conversion

ValueError: invalid literal for int() with base 10: '77.0'
```

In [38]:
```
float('77.5')
```

Out[38]:
```
77.5
```

In [39]:
```
float('77')  # string representing an int can be converted to a float
```

Out[39]:
```
77.0
```

Dictionary keys

- Dictionary keys must be *immutable*
- Related to how dictionaries are stored
  - Location of value is obtained by applying hash function to the key
  - If they key is allowed to change, hash function will map to another location

In [40]:
```
d = {}
```

In [41]:
```
d[1] = 7
```

In [42]:
```
d[8.5] = 8
```

In [43]:
```
d["MEX"] = "Arlington"
```

In [44]:
```
d[(3,4)] = 5
```

In [45]:
```
d
```

Out[45]:
```
{1: 7, 8.5: 8, 'MEX': 'Arlington', (3, 4): 5}
```

In [46]:
```
d[[6,8]] = 10
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[46], line 1
----> 1 d[[6,8]] = 10

TypeError: unhashable type: 'list'
```

- Key can be `int`, `float`, `bool`, `str`, tuple
- Cannot use a list or a dictionary as a key

Map and filter

In [47]:
```
matchlist = [
  ("Mexico City","MEX","RSA",2,0,0,0,1.84,0.52),
  ("Zapopan","KOR","CZE",2,1,0,0,1.45,1.12),
  ("Toronto","CAN","BIH",1,1,0,0,1.35,0.98),
  ("Inglewood","USA","PAR",4,1,0,0,2.76,0.88),
  ("Foxborough","QAT","SUI",1,1,0,0,0.78,1.54),
  ("East Rutherford","BRA","MAR",1,1,0,0,1.62,1.1),
  ("Philadelphia","HAI","SCO",0,1,0,0,1.25,0.65),
  ("Seattle","AUS","TUR",2,0,0,0,1.48,0.72),
  ("Atlanta","GER","CUW",7,1,0,0,4.82,0.44),
  ("Santa Clara","NED","JPN",2,2,0,0,1.95,1.82),
  ("Houston","CIV","ECU",1,0,0,0,1.1,0.95),
  ("Kansas City","SWE","TUN",5,1,0,0,3.12,0.78),
  ("Miami Gardens","ESP","CPV",0,0,0,0,2.15,0.35),
  ("Arlington","BEL","EGY",1,1,0,0,1.58,1.12),
  ("Guadalupe","KSA","URU",1,1,0,0,0.85,1.76),
  ("Vancouver","IRN","NZL",2,2,0,0,1.88,1.2),
  ("East Rutherford","FRA","SEN",3,1,0,0,2.44,1.05),
  ("Foxborough","IRQ","NOR",1,4,0,0,0.68,2.85),
  ("Inglewood","ARG","ALG",3,0,0,0,2.65,0.42),
  ("Santa Clara","AUT","JOR",3,1,0,0,1.98,0.88),
  ("Atlanta","POR","COD",1,1,0,0,1.78,1.15),
  ("Miami Gardens","ENG","CRO",4,2,0,0,2.92,1.64),
  ("Houston","GHA","PAN",1,0,0,0,1.3,0.75),
  ("Seattle","UZB","COL",1,3,0,0,0.95,2.1),
  ("Atlanta","MEX","KOR",1,0,0,0,1.62,0.58),
  ("Inglewood","CZE","RSA",1,1,0,0,1.35,1.15),
  ("Vancouver","CAN","QAT",6,0,0,0,3.82,0.22),
  ("Mexico City","SUI","BIH",4,1,0,0,2.45,0.88),
  ("East Rutherford","BRA","HAI",3,0,0,0,1.16,0.9),
  ("Arlington","SCO","MAR",0,1,0,0,0.97,0.54),
  ("Toronto","USA","AUS",2,0,0,0,1.21,0.32),
  ("Guadalupe","TUR","PAR",0,1,0,0,0.4,0.3),
  ("Foxborough","GER","CIV",2,1,0,0,1.85,0.72),
  ("Kansas City","ECU","CUW",0,0,0,0,1.92,0.08),
  ("Philadelphia","NED","SWE",5,1,0,0,3.45,0.88),
  ("Seattle","TUN","JPN",0,4,0,0,0.35,3.12),
  ("Houston","BEL","IRN",0,0,0,0,1.45,0.62),
  ("Miami Gardens","NZL","EGY",1,3,0,0,0.55,2.15),
  ("Santa Clara","ESP","KSA",4,0,0,0,3.28,0.25),
  ("East Rutherford","URU","CPV",2,2,0,0,1.65,1.1),
  ("Arlington","FRA","IRQ",3,0,0,0,2.21,0.45),
  ("Toronto","NOR","SEN",3,2,0,0,1.95,1.48),
  ("Guadalupe","ARG","AUT",2,0,0,0,1.78,0.55),
  ("Foxborough","JOR","ALG",1,2,0,0,0.88,1.92),
  ("Kansas City","POR","UZB",5,0,0,0,3.54,0.42),
  ("Philadelphia","COL","COD",1,0,0,0,1.62,0.58),
  ("Seattle","ENG","GHA",0,0,0,0,1.12,0.89),
  ("Mexico City","PAN","CRO",0,1,0,0,0.42,1.48),
  ("Mexico City","CZE","MEX",0,3,0,0,1.1,1.27),
  ("Zapopan","RSA","KOR",1,0,0,0,1.26,1.39),
  ("Vancouver","SUI","CAN",2,1,0,0,0.71,1.35),
  ("Toronto","BIH","QAT",3,1,0,0,1.48,0.95),
  ("East Rutherford","SCO","BRA",0,3,0,0,0.45,2.18),
  ("Foxborough","MAR","HAI",4,2,0,0,2.35,1.12),
  ("Inglewood","TUR","USA",3,2,0,0,1.62,1.45),
  ("Santa Clara","PAR","AUS",0,0,0,0,0.55,0.48),
  ("Atlanta","ECU","GER",2,1,0,0,1.28,1.55),
  ("Houston","CUW","CIV",0,2,0,0,0.32,1.68),
  ("Kansas City","TUN","NED",1,3,0,0,0.62,2.15),
  ("Seattle","JPN","SWE",1,1,0,0,1.12,0.95),
```

```
  ("Arlington","NZL","BEL",1,5,0,0,0.48,3.25),
  ("Guadalupe","EGY","IRN",1,1,0,0,0.85,0.92),
  ("Miami Gardens","URU","ESP",0,1,0,0,0.78,1.35),
  ("Philadelphia","CPV","KSA",0,0,0,0,0.42,0.38),
  ("Foxborough","NOR","FRA",1,4,0,0,1.3,0.96),
  ("East Rutherford","SEN","IRQ",5,0,0,0,3.01,0.14),
  ("Santa Clara","JOR","ARG",1,3,0,0,0.76,2.13),
  ("Inglewood","ALG","AUT",3,3,0,0,1.62,1.44),
  ("Miami Gardens","COL","POR",0,0,0,0,1.62,0.73),
  ("Atlanta","COD","UZB",3,1,0,0,2.35,0.2),
  ("East Rutherford","PAN","ENG",0,2,0,0,0.69,1.39),
  ("Philadelphia","CRO","GHA",2,1,0,0,0.42,0.74),
  ("Inglewood","RSA","CAN",0,1,0,0,0.13,1.32),
  ("Houston","BRA","JPN",2,1,0,0,1.69,0.23),
  ("Foxborough","GER","PAR",1,1,3,4,0.72,0.42),
  ("Guadalupe","NED","MAR",1,1,2,3,0.23,1.4),
  ("Arlington","CIV","NOR",1,2,0,0,1.15,2.02),
  ("East Rutherford","FRA","SWE",3,0,0,0,3.17,0.67),
  ("Mexico City","MEX","ECU",2,0,0,0,1.02,0.73),
  ("Atlanta","ENG","COD",2,1,0,0,2.04,0.76),
  ("Seattle","BEL","SEN",3,2,0,0,1.74,3.58),
  ("Santa Clara","USA","BIH",2,0,0,0,0.92,0.25),
  ("Inglewood","ESP","AUT",3,0,0,0,2.84,0.32),
  ("Toronto","POR","CRO",2,1,0,0,2.18,1.34),
  ("Vancouver","SUI","ALG",2,0,0,0,2.52,0.73),
  ("Arlington","AUS","EGY",1,1,2,4,0.87,1.36),
  ("Miami Gardens","ARG","CPV",3,2,0,0,2.16,0.46),
  ("Kansas City","COL","GHA",1,0,0,0,2.19,0.26),
  ("Philadelphia","PAR","FRA",0,1,0,0,0.3,1.8),
  ("Houston","CAN","MAR",0,3,0,0,0.8,1.2),
  ("Arlington","BRA","NOR",1,2,0,0,2.61,1.05),
  ("Mexico City","MEX","ENG",2,3,0,0,1.88,1.61),
  ("Miami Gardens","POR","ESP",0,1,0,0,0.63,1.69),
  ("Santa Clara","USA","BEL",1,4,0,0,0.67,2.15),
  ("Kansas City","ARG","EGY",3,2,0,0,2.12,1.54),
  ("Seattle","SUI","COL",0,0,4,3,1.15,1.46),
  ("Foxborough","FRA","MAR",2,0,0,0,3.1,0.13),
  ("Inglewood","ESP","BEL",2,1,0,0,2.08,0.37),
  ("Miami Gardens","NOR","ENG",1,2,0,0,0.77,0.96),
  ("Kansas City","ARG","SUI",3,1,0,0,2.0,0.53),
  ("Arlington","FRA","ESP",0,2,0,0,0.3,1.63),
  ("Atlanta","ENG","ARG",1,2,0,0,0.54,1.80),
  ("Miami Gardens","FRA","ENG",4,6,0,0,2.88,2.88),
  ("East Rutherford","ESP","ARG",1,0,0,0,0.52,0.09)
]
```

- List of teams that played FIFA 2026

In [48]:
```
def get_teams(l):
    teamdict = {}
    for m in l:
        team1,team2 = m[1],m[2]
        teamdict[team1] = 1
        teamdict[team2] = 2
    return(sorted(list(teamdict)))
```

In [49]:
```
get_teams(matchlist)
```

Out[49]:
```
['ALG',
 'ARG',
 'AUS',
 'AUT',
 'BEL',
 'BIH',
 'BRA',
 'CAN',
 'CIV',
 'COD',
 'COL',
 'CPV',
 'CRO',
 'CUW',
 'CZE',
 'ECU',
 'EGY',
 'ENG',
 'ESP',
 'FRA',
 'GER',
 'GHA',
 'HAI',
 'IRN',
 'IRQ',
 'JOR',
 'JPN',
 'KOR',
 'KSA',
 'MAR',
 'MEX',
 'NED',
 'NOR',
 'NZL',
 'PAN',
 'PAR',
 'POR',
 'QAT',
 'RSA',
 'SCO',
 'SEN',
 'SUI',
 'SWE',
 'TUN',
 'TUR',
 'URU',
 'USA',
 'UZB']
```

- We would like this list in terms of full country names rather than 3 letter abbreviations
- Replace each abbreviation by the corresponding country name

Map

- Apply a function $f()$ to each element in a list
- Convert $[x_0, x_1, \ldots, x_{n-1}]$ to $[f(x_0), f(x_1), \ldots, f(x_{n-1})]$
- In Python, `map(f,l)` applies `f` to each element of `l`

Example

- List full names of teams that played in FIFA 2026
- First, a function to map team abbreviations to full names, using a dictionary

In [50]:
```
def expand(s):
    teamdict = {
        'ALG':'Algeria',
        'ARG':'Argentina',
        'AUS':'Australia',
        'AUT':'Austria',
        'BEL':'Belgium',
        'BIH':'Bosnia Herzegovina',
        'BRA':'Brazil',
        'CAN':'Canada',
        'CIV':'Ivory Coast',
        'COD':'Congo',
        'COL':'Colombia',
        'CPV':'Cape Verde',
        'CRO':'Croatia',
        'CUW':'Curacao',
        'CZE':'Czech Republic',
        'ECU':'Ecuador',
        'EGY':'Egypt',
        'ENG':'England',
        'ESP':'Spain',
        'FRA':'France',
        'GER':'Germany',
        'GHA':'Ghana',
        'HAI':'Haiti',
        'IRN':'Iran',
        'IRQ':'Iraq',
        'JOR':'Jordan',
        'JPN':'Japan',
        'KOR':'Korea',
        'KSA':'Saudi Arabia',
        'MAR':'Morocco',
        'MEX':'Mexico',
        'NED':'Netherlands',
        'NOR':'Norway',
        'NZL':'New Zealand',
        'PAN':'Panama',
        'PAR':'Paraguay',
        'POR':'Portugal',
        'QAT':'Qatar',
        'RSA':'South Africa',
        'SCO':'Scotland',
        'SEN':'Senegal',
        'SUI':'Switzerland',
        'SWE':'Sweden',
        'TUN':'Tunisia',
        'TUR':'Turkey',
        'URU':'Uruguay',
        'USA':'United States of America',
        'UZB':'Uzbekistan'}
    if (s in teamdict):
        return(teamdict[s])
    else:
        return('No info')
```

- Now, we can `map` this function to the outcome of our earlier function

In [51]:
```
teams = get_teams(matchlist)
```

In [52]:
```
teams
```

Out[52]:
```
['ALG',
 'ARG',
 'AUS',
 'AUT',
 'BEL',
 'BIH',
 'BRA',
 'CAN',
 'CIV',
 'COD',
 'COL',
 'CPV',
 'CRO',
 'CUW',
 'CZE',
 'ECU',
 'EGY',
 'ENG',
 'ESP',
 'FRA',
 'GER',
 'GHA',
 'HAI',
 'IRN',
 'IRQ',
 'JOR',
 'JPN',
 'KOR',
 'KSA',
 'MAR',
 'MEX',
 'NED',
 'NOR',
 'NZL',
 'PAN',
 'PAR',
 'POR',
 'QAT',
 'RSA',
 'SCO',
 'SEN',
 'SUI',
 'SWE',
 'TUN',
 'TUR',
 'URU',
 'USA',
 'UZB']
```

- Output of `map` is a sequence, but not a list, like `range`

In [53]:
```
map(expand,teams)
```

Out[53]:
```
<map at 0x7f55b42846a0>
```

- Explicitly convert it to a list to view the output

In [54]:
```
list(map(expand,teams))
```

Out[54]:
```
['Algeria',
 'Argentina',
 'Australia',
 'Austria',
 'Belgium',
 'Bosnia Herzegovina',
 'Brazil',
 'Canada',
 'Ivory Coast',
 'Congo',
 'Colombia',
 'Cape Verde',
 'Croatia',
 'Curacao',
 'Czech Republic',
 'Ecuador',
 'Egypt',
 'England',
 'Spain',
 'France',
 'Germany',
 'Ghana',
 'Haiti',
 'Iran',
 'Iraq',
 'Jordan',
 'Japan',
 'Korea',
 'Saudi Arabia',
 'Morocco',
 'Mexico',
 'Netherlands',
 'Norway',
 'New Zealand',
 'Panama',
 'Paraguay',
 'Portugal',
 'Qatar',
 'South Africa',
 'Scotland',
 'Senegal',
 'Switzerland',
 'Sweden',
 'Tunisia',
 'Turkey',
 'Uruguay',
 'United States of America',
 'Uzbekistan']
```

- Since `expand` returns `No info` for unknown keys, the following works
- Note that keys need not be of uniform type: `7` is merely an unknown key, not an invalid one because it is not a string

In [55]:
```
list(map(expand,['xxx','yyy',7]))
```

Out[55]:
```
['No info', 'No info', 'No info']
```

- `map(f,l)` is a shortcut for the following iteration

```
outputlist = []
for x in l:
   outputlist.append(f(x))
```

In [56]:
```
expandedlist = []
for t in teams:
    expandedlist.append(expand(t))
```

In [57]:
```
expandedlist
```

Out[57]:
```
['Algeria',
 'Argentina',
 'Australia',
 'Austria',
 'Belgium',
 'Bosnia Herzegovina',
 'Brazil',
 'Canada',
 'Ivory Coast',
 'Congo',
 'Colombia',
 'Cape Verde',
 'Croatia',
 'Curacao',
 'Czech Republic',
 'Ecuador',
 'Egypt',
 'England',
 'Spain',
 'France',
 'Germany',
 'Ghana',
 'Haiti',
 'Iran',
 'Iraq',
 'Jordan',
 'Japan',
 'Korea',
 'Saudi Arabia',
 'Morocco',
 'Mexico',
 'Netherlands',
 'Norway',
 'New Zealand',
 'Panama',
 'Paraguay',
 'Portugal',
 'Qatar',
 'South Africa',
 'Scotland',
 'Senegal',
 'Switzerland',
 'Sweden',
 'Tunisia',
 'Turkey',
 'Uruguay',
 'United States of America',
 'Uzbekistan']
```

Filter

- Extract items from a list that satisfy a property
  - For instance, extract the even numbers from a list of numbers
- A *property* is a function $p()$ that maps each input to `True` or `False
  - Check if each item x in a list satisfies a property $p(x)$
  - Retain only such elements
  - Filters out elements that do not satisfy $p()$
- In Python, `filter(p,l)`

Example

- List matches where MEX was the home team
- First define the filter function -- returns `True` or `False`

In [58]:
```
def myfilter(m):            # Take one match tuple as input
    return (m[1] == 'MEX')  # Check if the home team in this match is MEX
```

- Now, filter matchlist using this function
- As with `map`, the output is a sequence, but not a list

In [59]:
```
filter(myfilter,matchlist)
```

Out[59]:
```
<filter at 0x7f55b4286e30>
```

In [60]:
```
list(filter(myfilter,matchlist))
```

Out[60]:
```
[('Mexico City', 'MEX', 'RSA', 2, 0, 0, 0, 1.84, 0.52),
 ('Atlanta', 'MEX', 'KOR', 1, 0, 0, 0, 1.62, 0.58),
 ('Mexico City', 'MEX', 'ECU', 2, 0, 0, 0, 1.02, 0.73),
 ('Mexico City', 'MEX', 'ENG', 2, 3, 0, 0, 1.88, 1.61)]
```

- Note that `filter` does not modify any values in the list
- It only copies those elements that satisfy the given property

- Like `map`, `filter(p,l)` is equivalent to an iteration

```
outputlist = []
for x in l:
    if p(x):
        l.append(x)
```

In [61]:
```
filteredlist = []
for m in matchlist:
    if myfilter(m):
        filteredlist.append(m)
```

In [62]:
```
filteredlist
```

Out[62]:
```
[('Mexico City', 'MEX', 'RSA', 2, 0, 0, 0, 1.84, 0.52),
 ('Atlanta', 'MEX', 'KOR', 1, 0, 0, 0, 1.62, 0.58),
 ('Mexico City', 'MEX', 'ECU', 2, 0, 0, 0, 1.02, 0.73),
 ('Mexico City', 'MEX', 'ENG', 2, 3, 0, 0, 1.88, 1.61)]
```