

PDSP 2025, Lecture 09, 4 September 2025

```
In [1]: matchlist = [  
    ("Chennai", "RCB", "CSK", "RCB", "CSK", 174),  
    ("Mohali", "DC", "PK", "PK", "PK", 175),  
    ("Kolkata", "KKR", "SRH", "SRH", "KKR", 209),  
    ("Jaipur", "RR", "LSG", "RR", "RR", 194),  
    ("Ahmedabad", "GT", "MI", "MI", "GT", 169),  
    ("Bengaluru", "PK", "RCB", "RCB", "RCB", 177),  
    ("Chennai", "CSK", "GT", "GT", "CSK", 207),  
    ("Hyderabad", "SRH", "MI", "MI", "SRH", 278),  
    ("Jaipur", "RR", "DC", "DC", "RR", 186),  
    ("Bengaluru", "RCB", "KKR", "KKR", "KKR", 183),  
    ("Lucknow", "LSG", "PK", "LSG", "LSG", 200),  
    ("Ahmedabad", "SRH", "GT", "SRH", "GT", 163),  
    ("Visakhapatnam", "DC", "CSK", "DC", "DC", 192),  
    ("Mumbai", "MI", "RR", "RR", "RR", 126),  
    ("Bengaluru", "LSG", "RCB", "RCB", "LSG", 182),  
    ("Visakhapatnam", "KKR", "DC", "KKR", "KKR", 273),  
    ("Ahmedabad", "GT", "PK", "PK", "PK", 200),  
    ("Hyderabad", "CSK", "SRH", "SRH", "SRH", 166),  
    ("Jaipur", "RCB", "RR", "RR", "RR", 184),  
    ("Mumbai", "MI", "DC", "DC", "MI", 235),  
    ("Lucknow", "LSG", "GT", "LSG", "LSG", 164),  
    ("Chennai", "KKR", "CSK", "CSK", "CSK", 138),  
    ("Mohali", "SRH", "PK", "PK", "SRH", 183),  
    ("Jaipur", "RR", "GT", "GT", "GT", 197),  
    ("Mumbai", "RCB", "MI", "MI", "MI", 197),  
    ("Lucknow", "LSG", "DC", "LSG", "DC", 168),  
    ("Mohali", "PK", "RR", "RR", "RR", 148),  
    ("Kolkata", "LSG", "KKR", "KKR", "KKR", 162),  
    ("Mumbai", "CSK", "MI", "MI", "CSK", 207),  
    ("Bengaluru", "SRH", "RCB", "RCB", "SRH", 288),  
    ("Kolkata", "KKR", "RR", "RR", "RR", 224),  
    ("Ahmedabad", "GT", "DC", "DC", "DC", 90),  
    ("Mohali", "MI", "PK", "PK", "MI", 193),  
    ("Lucknow", "CSK", "LSG", "LSG", "LSG", 177),  
    ("Delhi", "SRH", "DC", "DC", "SRH", 267),  
    ("Kolkata", "KKR", "RCB", "RCB", "KKR", 223),  
    ("Mohali", "PK", "GT", "PK", "GT", 143),  
    ("Jaipur", "MI", "RR", "MI", "RR", 180),  
    ("Chennai", "CSK", "LSG", "LSG", "LSG", 211),  
    ("Delhi", "DC", "GT", "GT", "DC", 225),  
    ("Hyderabad", "RCB", "SRH", "RCB", "RCB", 207),  
    ("Kolkata", "KKR", "PK", "PK", "PK", 262),  
    ("Delhi", "DC", "MI", "MI", "DC", 258),  
    ("Lucknow", "LSG", "RR", "RR", "RR", 197),  
    ("Ahmedabad", "GT", "RCB", "RCB", "RCB", 201),  
    ("Chennai", "CSK", "SRH", "SRH", "CSK", 213),  
    ("Kolkata", "DC", "KKR", "DC", "KKR", 154),  
    ("Lucknow", "MI", "LSG", "LSG", "LSG", 145),  
    ("Chennai", "CSK", "PK", "PK", "PK", 163),  
    ("Hyderabad", "SRH", "RR", "SRH", "SRH", 202),  
    ("Mumbai", "KKR", "MI", "MI", "KKR", 170),  
    ("Bengaluru", "GT", "RCB", "RCB", "RCB", 148),  
    ("Dharamsala", "CSK", "PK", "PK", "CSK", 168),  
    ("Lucknow", "KKR", "LSG", "LSG", "KKR", 236),
```

```
( "Mumbai", "SRH", "MI", "MI", "MI", 174),
( "Delhi", "DC", "RR", "RR", "DC", 222),
( "Hyderabad", "LSG", "SRH", "LSG", "SRH", 166),
( "Dharamsala", "RCB", "PK", "PK", "RCB", 242),
( "Ahmedabad", "GT", "CSK", "CSK", "GT", 232),
( "Kolkata", "KKR", "MI", "MI", "KKR", 158),
( "Chennai", "RR", "CSK", "RR", "CSK", 142),
( "Bengaluru", "RCB", "DC", "DC", "RCB", 188),
( "Delhi", "DC", "LSG", "LSG", "DC", 209),
( "Guwahati", "RR", "PK", "RR", "PK", 145),
( "Mumbai", "LSG", "MI", "MI", "LSG", 215),
( "Bengaluru", "RCB", "CSK", "CSK", "RCB", 219),
( "Hyderabad", "PK", "SRH", "PK", "SRH", 215),
( "Ahmedabad", "SRH", "KKR", "SRH", "KKR", 160),
( "Ahmedabad", "RCB", "RR", "RR", "RR", 173),
( "Chennai", "SRH", "RR", "RR", "SRH", 176),
( "Chennai", "SRH", "KKR", "SRH", "KKR", 114)
]
```

- List of teams that played IPL 2024

```
In [2]: 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 [3]: get_teams(matchlist)
```

```
Out[3]: ['CSK', 'DC', 'GT', 'KKR', 'LSG', 'MI', 'PK', 'RCB', 'RR', 'SRH']
```

Map

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

Example

- List full names of teams that played in IPL 2024
- First, a function to map team abbreviations to full names, using a dictionary

```
In [4]: def expand(s):
        teamdict = {'CSK': 'Chennai Super Kings',
                    'PK': 'Punjab Kings',
                    'SRH': 'Sunrisers Hyderabad',
                    'LSG': 'Lucknow Super Giants',
                    'MI': 'Mumbai Indians',
                    'RCB': 'Royal Challengers Bengaluru',
                    'GT': 'Gujarat Titans',
                    'DC': 'Delhi Capitals',
                    'KKR': 'Kolkata Knight Riders',
                    'RR': 'Rajasthan Royals'}
```

```
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 [5]: teams = get_teams(matchlist)
```

```
In [6]: teams
```

```
Out[6]: ['CSK', 'DC', 'GT', 'KKR', 'LSG', 'MI', 'PK', 'RCB', 'RR', 'SRH']
```

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

```
In [7]: map(expand, teams)
```

```
Out[7]: <map at 0x7f0b226e5ff0>
```

- Explicitly convert it to a list to view the output

```
In [8]: list(map(expand, teams))
```

```
Out[8]: ['Chennai Super Kings',
'Delhi Capitals',
'Gujarat Titans',
'Kolata Knight Riders',
'Lucknow Super Giants',
'Mumbai Indians',
'Punjab Kings',
'Royal Challengers Bengaluru',
'Rajasthan Royals',
'Sunrisers Hyderabad']
```

- 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 [9]: list(map(expand, ['xxx', 'yyy', 7]))
```

```
Out[9]: ['No info', 'No info', 'No info']
```

Filter

- Check if each item x in a list satisfies a property $p(x)$
- Retain only such elements
- Filter out elements that do not satisfy $p()$
- In Python, `filter(p, l)`

Example

- List matches where CSK won the toss
- First define the filter function -- returns `True` or `False`

```
In [10]: def csktosswin(t): # t is expected to be one tuple from matchlist
         return(t[3] == 'CSK')
```

- Now, filter matchlist using this function

```
In [11]: list(filter(csktosswin,matchlist))
```

```
Out[11]: [('Chennai', 'KKR', 'CSK', 'CSK', 'CSK', 138),
          ('Ahmedabad', 'GT', 'CSK', 'CSK', 'GT', 232),
          ('Bengaluru', 'RCB', 'CSK', 'CSK', 'RCB', 219)]
```

List comprehension

- Combine map and filter to create a list
- *Set comprehension*: Squares of positive even integers = $\{x^2 \mid x \in \mathbb{Z}, x > 0\}$
- In Python: `[ f(x) for x in l if p(x) ]`

Example

- Full names of all teams in IPL 2024

```
In [12]: [ expand(t) for t in get_teams(matchlist) ]
```

```
Out[12]: ['Chennai Super Kings',
          'Delhi Capitals',
          'Gujarat Titans',
          'Kolata Knight Riders',
          'Lucknow Super Giants',
          'Mumbai Indians',
          'Punjab Kings',
          'Royal Challengers Bengaluru',
          'Rajasthan Royals',
          'Sunrisers Hyderabad']
```

- List both teams in matches where CSK won the toss

```
In [13]: [ (t[1],t[2]) for t in matchlist if t[3] == "CSK" ]
```

```
Out[13]: [('KKR', 'CSK'), ('GT', 'CSK'), ('RCB', 'CSK')]
```

- Same, with full names

```
In [14]: [ (expand(t[1]),expand(t[2])) for t in matchlist if t[3] == "CSK" ]
```

```
Out[14]: [('Kolata Knight Riders', 'Chennai Super Kings'),
          ('Gujarat Titans', 'Chennai Super Kings'),
          ('Royal Challengers Bengaluru', 'Chennai Super Kings')]
```

- Similar notation works for dictionaries
- Create a dictionary matching team abbreviations to full names for teams in IPL 2024

```
In [15]: { t:expand(t) for t in get_teams(matchlist) }
```

```
Out[15]: {'CSK': 'Chennai Super Kings',
          'DC': 'Delhi Capitals',
          'GT': 'Gujarat Titans',
          'KKR': 'Kolata Knight Riders',
          'LSG': 'Lucknow Super Giants',
          'MI': 'Mumbai Indians',
          'PK': 'Punjab Kings',
          'RCB': 'Royal Challengers Bengaluru',
          'RR': 'Rajasthan Royals',
          'SRH': 'Sunrisers Hyderabad'}
```

- Recall that `uniqd` created a list of unique items via keys of a dictionary
- Here is a short way to do this using list comprehension

```
In [16]: list({ x[2]:1 for x in matchlist})
```

```
Out[16]: ['CSK', 'PK', 'SRH', 'LSG', 'MI', 'RCB', 'GT', 'DC', 'KKR', 'RR']
```

- Uses the fact that a dictionary `d` when interpreted as a sequence is implicitly `d.keys()`
- `list(d)` looks for a sequence `d`
- Can also do the equivalent of relational algebra selection and projection
- Project matchlist onto columns `team 1, team 2, target` (columns 1,2,5) where CSK won the toss
 - Filter by CSK winning the toss (select)
 - Project onto columns 1,2,5

```
In [17]: [ (t[1],t[2],t[5]) for t in matchlist if t[3] == "CSK" ]
```

```
Out[17]: [('KKR', 'CSK', 138), ('GT', 'CSK', 232), ('RCB', 'CSK', 219)]
```

- Can have multiple generators
- Like nested loops, the left most generator is the outermost loop

```
In [18]: [ (i,j) for i in range(3) for j in range(4) ]
```

```
Out[18]: [(0, 0),
          (0, 1),
          (0, 2),
          (0, 3),
          (1, 0),
          (1, 1),
          (1, 2),
          (1, 3),
          (2, 0),
          (2, 1),
          (2, 2),
          (2, 3)]
```

- **Example:** "Small" Pythagorean triples

```
In [19]: [(x,y,z) for x in range(1,20) for y in range(1,20) for z in range(1,20)]
```

```
Out[19]: [(3, 4, 5),
          (4, 3, 5),
          (5, 12, 13),
          (6, 8, 10),
          (8, 6, 10),
          (8, 15, 17),
          (9, 12, 15),
          (12, 5, 13),
          (12, 9, 15),
          (15, 8, 17)]
```

- Avoid duplicates like (3,4,5) , (4,3,5) -- ensure that `y > x`
- `y in range(x,20)` -- later generator can use value from an earlier one, like in a nested loop

```
In [20]: [(x,y,z) for x in range(1,20) for y in range(x,20) for z in range(x,20)]
```

```
Out[20]: [(3, 4, 5), (5, 12, 13), (6, 8, 10), (8, 15, 17), (9, 12, 15)]
```

- Can report the triples in a different order

```
In [21]: [(y,x,z) for x in range(1,20) for y in range(x,20) for z in range(x,20)]
```

```
Out[21]: [(4, 3, 5), (12, 5, 13), (8, 6, 10), (15, 8, 17), (12, 9, 15)]
```

Mutable and immutable values

- Lists and dictionaries can be updated in place
 - Can reassign `l[i]` or `d[k]`
 - These are *mutable* values
- Numbers (`int` , `float`), booleans, strings, tuples cannot be updated in place
 - Immutable values

Mutability and assignment

- Assigning a mutable value creates an *alias*
- Updating through either the old or the new name indirectly affects the other

```
In [22]: l = [1,2,3]
        newl = l
        newl[0] = 4
```

```
In [23]: l, newl
```

```
Out[23]: ([4, 2, 3], [4, 2, 3])
```

```
In [24]: l[1] = 5
```

```
In [25]: l, newl
```

```
Out[25]: ([4, 5, 3], [4, 5, 3])
```

- For immutable values, assignment behaves as we would expect
- The two names can be updated without affecting each other
 - It is as though assignment copies the value

```
In [26]: x = 17
        y = x
        y = 19
```

```
In [27]: x, y
```

```
Out[27]: (17, 19)
```

```
In [28]: x = 18
```

```
In [29]: x, y
```

```
Out[29]: (18, 19)
```

Mutable and immutable values

- Lists and dictionaries are mutable
- `int`, `float`, `bool`, `str`, tuple are immutable
- For immutable values, assignment copies the value

```
In [30]: x = 5
        y = x
        y = 7 # Does not affect the value of x
```

```
In [31]: x,y
```

```
Out[31]: (5, 7)
```

- For mutable values, assignment *aliases* the new name to point to the same value as the old name
- Updating through either name affects both

```
In [32]: l1 = [1,2,3]
        l2 = l1
        l2[0] = 4
```

```
In [33]: l1, l2
```

```
Out[33]: ([4, 2, 3], [4, 2, 3])
```

```
In [34]: l1[2] = 6
```

```
In [35]: l1, l2
```

```
Out[35]: ([4, 2, 6], [4, 2, 6])
```

Slices and copying lists

- A slice creates a new list
- `l[0:len(l)]` is a faithful copy of `l`
 - Abbreviate as `l[:]`, *full slice*
- Assigning a full slice makes a disjoint copy of a list

```
In [36]: l1 = [1,2,3]
        l2 = l1[:]
```

```
In [37]: l1, l2
```

```
Out[37]: ([1, 2, 3], [1, 2, 3])
```

```
In [38]: l1[2] = 6
        l2[0] = 4
```

```
In [39]: l1, l2
```

```
Out[39]: ([1, 2, 6], [4, 2, 3])
```