

PDSP 2025, Lecture 08, 2 September 2025

Nested collections

- List of lists, list of tuples, dictionary whose values are lists ...
- `matchlist` is a list of tuples
- Use two indices to extract a value
 - `matchlist[3]` is `("Jaipur", "RR", "LSG", "RR", "RR", 194)`
 - `matchlist[3][1]` is `"RR"`

```
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 venues where each team played IPL 2024

- Initially, build a dictionary of dictionaries
 - Outer keys are team names
 - Inner dictionary keys are venues where team played
- Finally, convert dictionary of dictionaries into dictionary of lists

```
In [2]: def get_venues(l):
        venues = {}
        for m in l:
            venue, team1, team2 = m[0], m[1], m[2]
            for t in team1, team2:
                if t in venues:
                    venues[t][venue] = 1
                else:
                    venues[t] = {venue:1} # Create a dictionary for venues[t]
                                           # Equivalent to
                                           # venues[t] = {}
                                           # venues[t][venue] = 1

        venuelists = {}
        for t in sorted(venues): # Scan teams in alphabetical order
            venuelists[t] = sorted(list(venues[t])) # Sort venues in alphabetical order
        return(venuelists)
```

```
In [3]: get_venues(matchlist)
```

```
Out[3]: {'CSK': ['Ahmedabad',
                'Bengaluru',
                'Chennai',
                'Dharamsala',
                'Hyderabad',
                'Lucknow',
                'Mumbai',
                'Visakhapatnam'],
         'DC': ['Ahmedabad',
                'Bengaluru',
                'Delhi',
                'Jaipur',
                'Kolkata',
                'Lucknow',
                'Mohali',
                'Mumbai',
                'Visakhapatnam'],
         'GT': ['Ahmedabad',
                'Bengaluru',
                'Chennai',
                'Delhi',
                'Jaipur',
                'Lucknow',
                'Mohali'],
         'KKR': ['Ahmedabad',
                 'Bengaluru',
                 'Chennai',
                 'Kolkata',
                 'Lucknow',
                 'Mumbai',
                 'Visakhapatnam'],
         'LSG': ['Bengaluru',
                 'Chennai',
                 'Delhi',
                 'Hyderabad',
                 'Jaipur',
                 'Kolkata',
                 'Lucknow',
                 'Mumbai'],
         'MI': ['Ahmedabad',
                'Delhi',
                'Hyderabad',
                'Jaipur',
                'Kolkata',
                'Lucknow',
                'Mohali',
                'Mumbai'],
         'PK': ['Ahmedabad',
                'Bengaluru',
                'Chennai',
                'Dharamsala',
                'Guwahati',
                'Hyderabad',
                'Kolkata',
                'Lucknow',
                'Mohali'],
         'RCB': ['Ahmedabad',
                 'Bengaluru',
                 'Chennai',
                 'Dharamsala',
                 'Hyderabad',
                 'Jaipur',
                 'Kolkata',
                 'Mumbai'],
         'RR': ['Ahmedabad',
                'Chennai',
                'Delhi',
                'Guwahati',
                'Hyderabad',
                'Jaipur',
                'Kolkata',
                'Lucknow',
                'Mohali',
                'Mumbai'],
         'SRH': ['Ahmedabad',
```

```
'Bengaluru',  
'Chennai',  
'Delhi',  
'Hyderabad',  
'Kolkata',  
'Mohali',  
'Mumbai']}]}
```

Matrices

- Matrix is a list of lists
- Each row is a list of values `[v1,v2,...vm]`
- Matrix is a list of rows `[ row1, row2, ..., rown ]`

Transpose a matrix

- Columns of original matrix are rows of transpose
- If input is square, can swap `M[i][j]` and `M[j][i]`
- If not square, need to construct transpose of correct dimension

```
In [4]: def transpose(mat):  
        # Extract dimensions of input -- assume well-formed, non-empty  
        nrows = len(mat)  
        ncols = len(mat[0])  
        # Create an empty matrix for the transpose  
        trmat = []  
        for i in range(ncols):  
            trmat.append([])  
        # Populate the transpose -- append each M[i][j] to row j of transpose  
        for i in range(nrows):  
            for j in range(ncols):  
                trmat[j].append(mat[i][j])  
        return(trmat)
```

```
In [5]: m = [[0,3],[1,4],[2,5]]  
        mt = transpose(m)
```

```
In [6]: m, mt
```

```
Out[6]: ([[0, 3], [1, 4], [2, 5]], [[0, 1, 2], [3, 4, 5]])
```

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

```
In [8]: s
```

```
Out[8]: '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 [9]: s = "hello"
```

```
In [10]: s[1]
```

```
Out[10]: 'e'
```

```
In [11]: s[2:4]
```

```
Out[11]: '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 [12]: s[1] == "e" # Logically speaking, s[1] is a single character
```

```
Out[12]: True
```

- Concatenate strings using `+`

```
In [13]: s = "hello"  
t = "there"
```

```
In [14]: s+t, s, t
```

```
Out[14]: ('hellothere', 'hello', 'there')
```

- Like tuples, cannot update parts of a string directly

```
In [15]: s[3] = "p"
```

```
-----  
TypeError                                Traceback (most recent call last)  
Cell In[15], line 1  
----> 1 s[3] = "p"  
TypeError: 'str' object does not support item assignment
```

- Instead, assemble a new string from the old one

```
In [16]: s = s[0:3] + "p" + s[4:]
```

```
In [17]: s
```

```
Out[17]: 'helpo'
```

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

```
In [18]: def vowel(c):  
         return(c in "aeiou")
```

```
In [19]: vowel("a"), vowel("b"),
```

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

- For string `x in s` seems to be interpreted as substring membership

```
In [20]: vowel("ae"), vowel("ai"), vowel("io")
```

```
Out[20]: (True, 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[18], line 2, in vowel(c)
      1 def vowel(c):
----> 2     return(c in )

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'
```

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

```
In [28]: int('77')
```

```
Out[28]: 77
```

```
In [29]: int('hello')
```

```
-----
ValueError                                Traceback (most recent call last)
Cell In[29], line 1
----> 1 int( )

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

```
In [30]: int('77.5') # 77.5 is a number, but not an int
```

```
-----  
ValueError                                Traceback (most recent call last)  
Cell In[30], line 1  
----> 1 int( ) # 77.5 is a number, but not an int  
  
ValueError: invalid literal for int() with base 10: '77.5'
```

```
In [31]: float('77.5')
```

```
Out[31]: 77.5
```

```
In [32]: float('77')
```

```
Out[32]: 77.0
```