IIT Madras
BSc Degree

Home Lesson-5.4

Lesson-5.3

Lists

Simulating an IPL Innings

Let us return to the problem of recording the number of runs scored in every ball of an IPL match. A typical innings of a T20 match has 20 overs, each over having 6 balls. Let us assume that all balls bowled are fair deliveries that do not concede any extras, a rather liberal assumption. This leaves us with exactly 120 numbers that we need to record, all lying between 0 and 6. How can this information be stored in a Python program that makes it suitable for further processing? A list is a good candidate.

Let us now simulate an innings. For this, we take the help of the `random` library:

```
import random
runs = random.choices([0, 1, 2, 3, 4, 5, 6], k = 120)
print(type(runs))
print(len(runs))
```

`choices` is a function in the `random` library. It uniformly samples from the seven numbers (0 to 6) given in the input list with replacement. If that sounded too cryptic, this is what it does:

- Pick a number from the list `[0, 1, 2, 3, 4, 5, 6]` at random. Each of the seven numbers is equally likely to be picked.
- Add this to the output list. The original list remains undisturbed, i.e., we are not moving an element from the input list to the output list, we are only copying it.
- Repeat this process 120 times.

Let us verify if the counts are approximately the same:

```
for run in [0, 1, 2, 3, 4, 5, 6]:
    print('{} appears {} times'.format(run, runs.count(run)))
```

`runs.count(run)` returns the number of times the element `run` appears in the list `runs`. `count` is a method defined for the `list` type. This gives the following output:

```
0 appears 19 times
1 appears 20 times
2 appears 19 times
3 appears 16 times
4 appears 18 times
5 appears 11 times
6 appears 17 times
```

The counts are quite close. But this is not very practical:

- `5` runs are seldom observed in cricket matches.
- `0`, `1` and `2` are much more common than `3`, `4` and `6`.

We can give our preferences using a weights keyword-argument:

```
import random
# choices is distributed over multiple lines
# this is done to improve readability
runs = random.choices([0, 1, 2, 3, 4, 5, 6],
                      weights = [30, 30, 20, 5, 10, 0, 5],
                      k = 120)
for run in [0, 1, 2, 3, 4, 5, 6]:
    print('{} appears {} times'.format(run, runs.count(run)))
print(f'Total number of runs scored = {sum(runs)}')
```

This gives the following output:

```
0 appears 32 times
1 appears 34 times
2 appears 32 times
3 appears 7 times
4 appears 12 times
5 appears 0 times
6 appears 3 times
Total number of runs scored = 185
```

We have used `sum(runs)` to get the sum of the elements in the list. `sum` is a built-in function. The way to understand the `weights` keyword-argument is using the following table:

Run	Weight
0	30
1	30
2	20
3	5
4	10
5	0
6	5
Total	100

The weight is the importance given to a run. From the table given above, we see that 0 and 1 occur 30% of the times, 6 occurs 5% of the times and so on. `choices` function will keep this distribution in mind while picking up items from the input-list.

Let us now start analyzing this innings. We have already seen how to count the number of occurrences of singles, doubles, fours and sixes. What about the first occurrence of a six? In which ball was the first six scored?

```
first_six_ball = runs.index(6) + 1
print(first_six_ball)
```

`index` is a method that accepts an element as input and returns the first occurrence of this element in the list. For example, `runs.index(6)` returns the first index where a six occurs in the list `runs`. Since the number of balls is one more than the index, `1` has been added. What happens if we pass an input that is not present in the list:

```
first_five_ball = runs.index(5)
print(first_five_ball)
```

In this case, `5` never occurs in the list. So this throws a `ValueError` with the following message: `5 is not in list`. One must be careful while using the `index` method. We could have done this using another method:

```
for ball, run in enumerate(runs):
    if run == 6:
        print(f'The first six was hit at ball number {ball + 1}')
        break
```

The `enumerate` object can be very handy when we want to access both the element and its index while iterating through a list. The enumerate object yields pairs: `(index, list[index])`. In some sense, we have two loop variables: the first is the index of the element in the list while the second is the element itself. Coming back to cricket, what if we want to find the number of balls it took to score the last 50 runs in the innings? It would be easier to reverse the list and then iterate through it:

```
balls = 0
last_runs = 0
for run in reversed(runs):
    last_runs += run
    balls += 1
    if last_runs >= 50:
        print(f'It took {balls} balls to score the last 50 runs.')
        break
```

The `reversed` object helps us iterate through the list in the reversed order. Note that it doesn't make any changes to the original list. One final question: we wish to find if the batsmen have run three runs at any point in the match. We don't want to know at which point in the innings this has happened.

```
three_existence = 3 in runs
print(three_existence)
```

Recall that we used the `in` keyword to check for the presence of one string in another. Something similar is happening here. The code given above prints `True` if 3 is an element in `runs` and `False` otherwise.