Hash Tables

- we do have built-in hash tables which are dictionaries
- as we know dictionaries are made up key - value pairs
- so we do need a hash function
- then we should perform hash on the key
- hash function will return us the address
- `{"pens":1000}` ---> 2

Data	Adress
	0
`{'pens', 300}`	1
	2
	3
	4
	5
	6

Hash Characteristics :

- It's one way only , so when we get `"books"` and put it in the hash and we do get 1 , the thing that we cannot do is to take 1 and try to get the `{"pens":300}` (❌ it's not possible)
  - `"pens"` -- ⚙️ Hash Function ⚙️ --> 1 ✅
  - 1 -- ⚙️ Hash Function ⚙️ --> ❌
- It's Deterministic , means that for a particular hash function every time we put `"pens"` in we expect to get the number 1 every time.

Build Our Own Hash Tables

we want to build our own hash table so we do create a method: `set_item`

Data	Adress
`['notebooks', 300]`	0
`['pens', 300]`	1
`['markers', 100]`, `['pencils', 300]`	2
	3
	4
	5
`["books",1000]`	6

- now we have some items in our hash table we can create a method to call these items: `get_item`
- what if we have multiple key value pairs at that address ❓
  - so we know we're going to have to create some sort of loop to reach the item.

Collisions

- TR : çarpışmalar

Two from of solutions to the collisions :

- Separate Chaining (lists , linkedlists, ...)
- Probing
  - a) **Linear Probing**
  - b) Quadratic Probing
  - c) Double Hashing

Separate Chaining

Separate Chaining is the way we gonna do in this course

✅ we should always have a prime number of addresses (0-6)

- the reason is a prime number increases the amount of randomness for how the key value pairs are going to be distributed through the hash table and this action reduces the collisions.

how dose __hash function works ?

`def __hash(self,key):`

```
`my_hash = 0`

`for letter in key:`

    `my_hash = (my_hash + ord(letter)*23) % len(self.data_map)`

`return my_hash`
```

`my_hash = 0`

- 'a' → 97 * 23 = 2231 → (0 + 2231) % 7 = 2231 % 7 = 6
- 'b' → 98 * 23 = 2254 → (6 + 2254) % 7 = 2260 % 7 = 5
- 'c' → 99 * 23 = 2277 → (5 + 2277) % 7 = 2282 % 7 = 1
- Final hash = 1

www.github.com/ebgdev