C# Dictionary Cheat Sheet

Namespace: using System.Collections.Generic;

1. Creating a Dictionary

- **Standard:**
  Dictionary<string, int> scores = new Dictionary<string, int>();

- **With Initial Data:**
  var scores = new Dictionary<string, int>
  {
  { "Alice", 90 },
  { "Bob", 85 }
  };

2. Adding & Updating Data

There are two ways to add data. Know the difference!

- **.Add(key, value)**: Strict. Throws an error if the key already exists.
  scores.Add("Charlie", 95);
  // scores.Add("Charlie", 100); // CRASH! Key already exists.

- **Index Syntax dict[key] = value**: Flexible. If key exists, it updates it. If not, it adds it. (**Best for Frequency Counters**).
  scores["Alice"] = 100; // Updates Alice
  scores["David"] = 50; // Adds David

3. Checking for Data (Crucial)

Before you try to read a key, you must check if it exists to avoid a crash.

- **.ContainsKey(key)**: Returns true if the key exists. **Fast** $O(1)$.
  if (scores.ContainsKey("Alice")) { ... }

- **.ContainsValue(value)**: Returns true if the value exists. **Slow** $O(n)$ (Avoid inside loops).

4. Accessing Data (Reading)

- **dict[key]**: Returns the value. Crashes if key is missing.
  int s = scores["Alice"];

- **.TryGetValue(key, out val)**: The "Safe" read. Tries to get the value; if missing, returns false (doesn't crash).
  ```
  if (scores.TryGetValue("Alice", out int val))
  {
      Console.WriteLine($"Alice's score is {val}");
  }
  ```

5. Removing Data

- **.Remove(key)**: Removes the item. Returns true if found and removed.
  ```
  scores.Remove("Bob");
  ```

- **.Clear()**: Deletes **everything**. Count becomes 0.

6. Looping (Iteration)

You can loop through the whole pair, or just keys/values.

- **Looping Pairs (Most Common):**
  ```
  foreach(KeyValuePair<string, int> entry in scores)
  {
      Console.WriteLine($"Name: {entry.Key}, Score: {entry.Value}");
  }
  // Short version: foreach(var entry in scores)
  ```

- **Looping Keys only:**
  ```
  foreach(string name in scores.Keys) { ... }
  ```

- **Looping Values only:**
  ```
  foreach(int score in scores.Values) { ... }
  ```

7. Useful Properties

- **.Count**: Returns how many items are currently in the dictionary.
- **.Keys**: Returns a collection of all keys.
- **.Values**: Returns a collection of all values.

💡 Pro-Pattern: The Frequency Counter

This is the logic used in 90% of Dictionary coding questions (e.g., "Count letters," "Count words," "Find duplicates").

```
string text = "apple banana apple";
string[] words = text.Split(' ');
Dictionary<string, int> freq = new Dictionary<string, int>();
```

```
foreach(string w in words)
{
   if (freq.ContainsKey(w))
   {
      freq[w]++; // Increment existing count
   }
   else
   {
      freq[w] = 1; // Start new count
   }
   // OR One-Liner shortcut:
   // freq[w] = freq.ContainsKey(w) ? freq[w] + 1 : 1;
}
```