Online Coding Test

Intro

Outline

1. Online coding test platforms
2. Things that you will prepare
3. Big-O

Outline

1. **Online coding test platforms**
2. Things that you will prepare
3. Big-O

Types of Coding Interview

Name	Online/Offline	Interviewer	Tool
Whiteboard (onsite) coding interview	Offline	Yes	whiteboard
Online coding interview	Online	Yes	notepad
Online coding test	Online	No	Coding test platform

Online Coding Test

I generally refer to the process as a "coding test" rather than a "coding interview." This distinction stems from the interaction style: when an interviewer collaborates closely with the candidate, it resembles an interview; when the candidate completes tasks independently, it's more like a test.

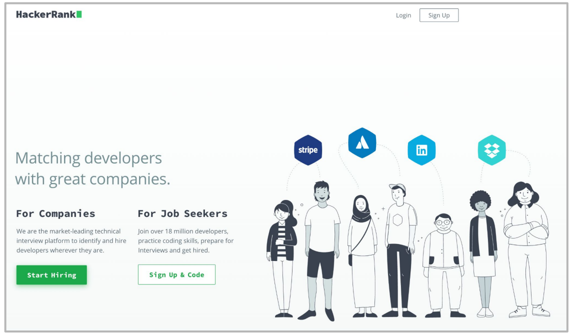
For in-person assessments that involve direct interaction with an interviewer, these are typically referred to as on-site or offline coding interviews. A common format for these is the whiteboard coding interview, where candidates solve problems on a whiteboard—this is prevalent in many on-site coding interviews.

Online, the format often shifts to a coding test conducted without an interviewer's presence. However, if an interviewer is involved, it's aptly called an online coding interview.

Our focus here is on the Online Coding Test.

Online Coding Test Platforms

- Hackerrank
  - https://www.hackerrank.com/
- Codility
  - https://www.codility.com/
- Remoteinterview
  - https://www.remoteinterview.io/
- And so on



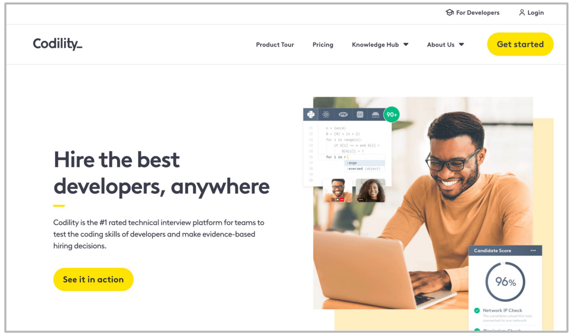
Matching developers
with great companies.

For Companies

We are the market-leading technical interview platform to identify and hire developers wherever they are.

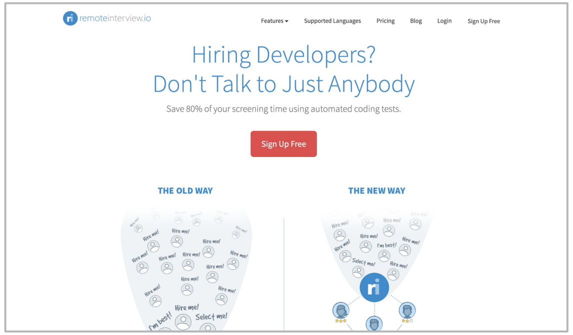
For Job Seekers

Join over 18 million developers, practice coding skills, prepare for Interviews and get hired.



Hire the best
developers, anywhere

Codility is the #1 rated technical interview platform for teams to test the coding skills of developers and make evidence-based hiring decisions.



Hiring Developers?
Don't Talk to Just Anybody

Save 80% of your screening time using automated coding tests.

Tips

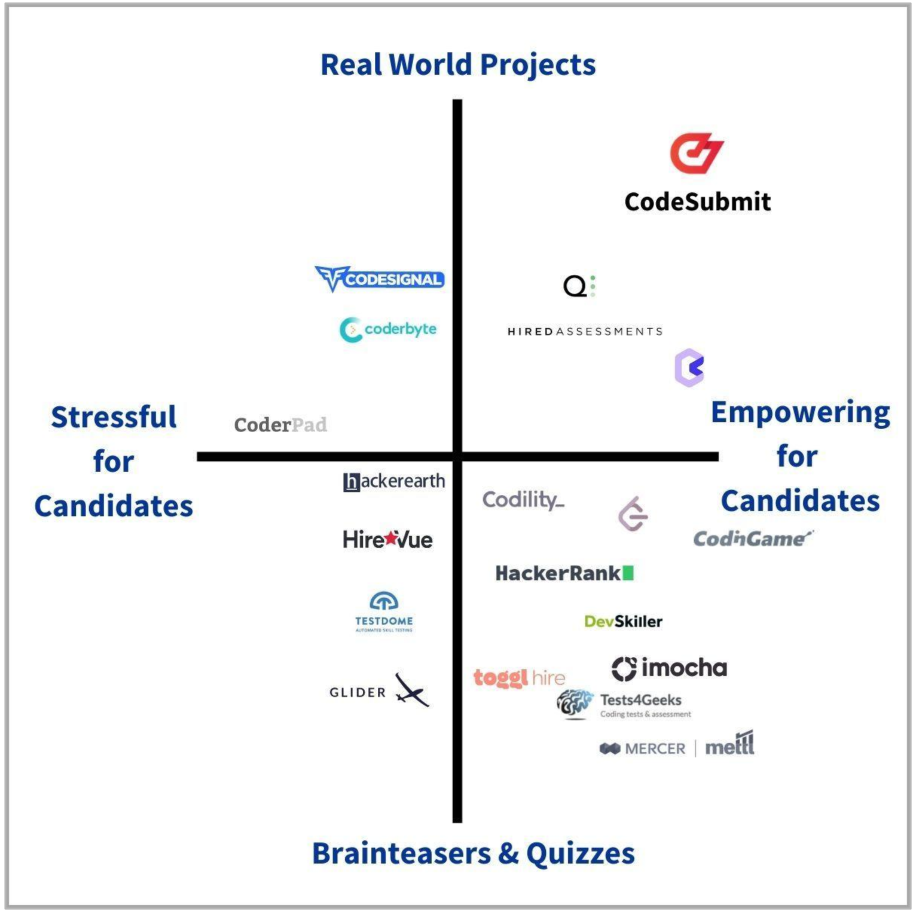
Be aware that coding test platforms **vary significantly** in their features. For instance, some may display test cases while others do not, making it challenging to diagnose errors when they occur.

It's important to research which platform your prospective employer uses and **familiarize** yourself with its specific features ahead of time. Taking the test on a platform you regularly use could give you a distinct advantage.

Typically, the timer starts as soon as you click the start button, and there is a strict **time limit**. To optimize your performance, make sure you're comfortable with the platform to avoid losing time due to unfamiliarity.

Useful Links

https://codesubmit.io/blog/coding-assessment-tools/



Outline

1. Online coding test platforms
2. **Things that you will prepare**
3. Big-O

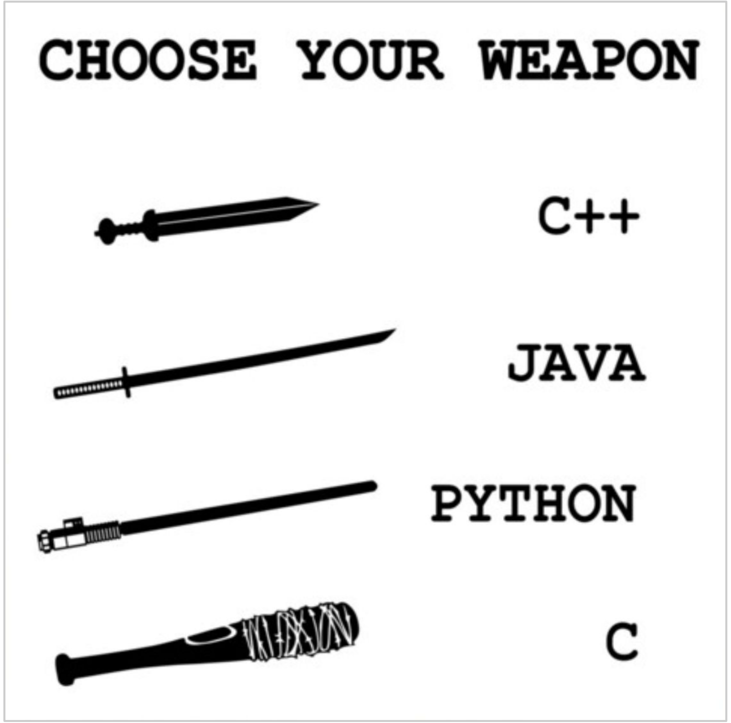
Scratch paper and favorite pen

Actually, it's not essential, but having it can be quite beneficial. This is especially true when dealing with recursive structures, as visualizing these structures mentally can be challenging. It's better to prepare in advance rather than scramble to do so later.

What language are you going to use?

While some companies may specify a required programming language, in most cases, applicants are free to choose their preferred language.

Therefore, it's important to have a good understanding of the characteristics of each language and select the one that best aligns with your skills and the task at hand.

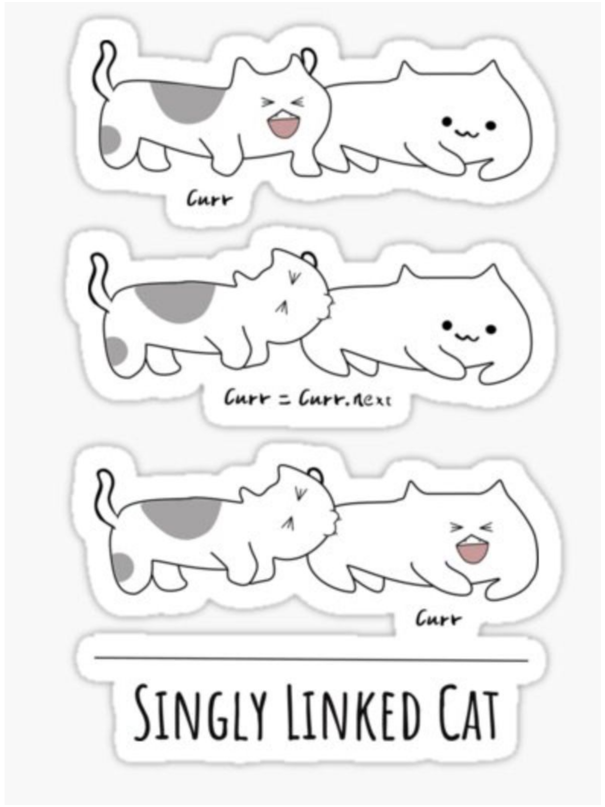


Your own code snippet

A "snippet" in programming refers to a small section of reusable source code that helps users avoid repetitive typing during regular editing tasks. It's useful to prepare frequently used code snippets in advance.

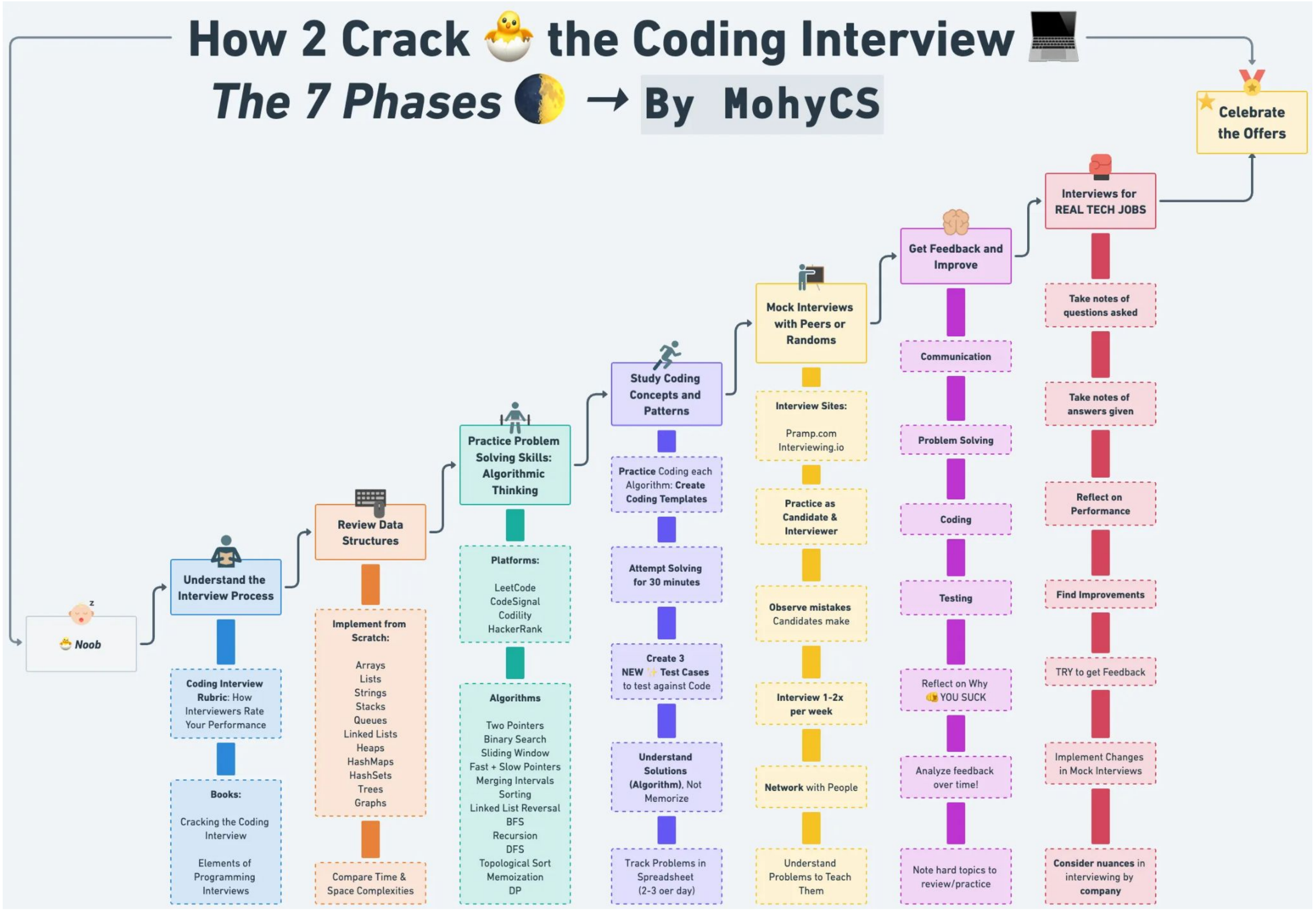
For instance, if you need to implement complex operations like inverting or deleting a linked list on the fly, it can be confusing and time-consuming. Having these snippets ready can save time and reduce errors.

Additionally, you can organize and store these snippets in your GitHub repository for easy access and management.



Coding tool

There's no reason to avoid using effective tools. Although it's crucial to have the capability to code without an Integrated Development Environment (IDE), the time constraints in online coding tests make it advantageous to use helpful tools. Personally, I find PyCharm and Jupyter Notebook/Colab to be excellent IDEs. However, it's best to use an IDE with which you are most familiar to maximize your efficiency during the test.



How 2 Crack 🐣 the Coding Interview 💻

The 7 Phases 🌓 → By MohyCS

🐣 *Noob*

Understand the Interview Process
- **Coding Interview Rubric**: How Interviewers Rate Your Performance
- **Books:** Cracking the Coding Interview; Elements of Programming Interviews

Review Data Structures
- **Implement from Scratch:** Arrays, Lists, Strings, Stacks, Queues, Linked Lists, Heaps, HashMaps, HashSets, Trees, Graphs
- Compare Time & Space Complexities

Practice Problem Solving Skills: Algorithmic Thinking
- **Platforms:** LeetCode, CodeSignal, Codility, HackerRank
- **Algorithms:** Two Pointers, Binary Search, Sliding Window, Fast + Slow Pointers, Merging Intervals, Sorting, Linked List Reversal, BFS, Recursion, DFS, Topological Sort, Memoization, DP

Study Coding Concepts and Patterns
- **Practice** Coding each Algorithm: **Create Coding Templates**
- **Attempt Solving for 30 minutes**
- **Create 3 NEW ✨ Test Cases** to test against Code
- **Understand Solutions (Algorithm)**, Not Memorize
- Track Problems in Spreadsheet (2-3 oer day)

Mock Interviews with Peers or Randoms
- **Interview Sites:** Pramp.com, Interviewing.io
- **Practice as Candidate & Interviewer**
- **Observe mistakes** Candidates make
- **Interview 1-2x per week**
- **Network** with People
- Understand Problems to Teach Them

Get Feedback and Improve
- **Communication**
- **Problem Solving**
- **Coding**
- **Testing**
- Reflect on Why 👊 YOU SUCK
- Analyze feedback over time!
- Note hard topics to review/practice

Interviews for REAL TECH JOBS
- **Take notes of questions asked**
- **Take notes of answers given**
- **Reflect on Performance**
- **Find Improvements**
- TRY to get Feedback
- Implement Changes in Mock Interviews
- **Consider nuances** in interviewing by **company**

Celebrate the Offers

https://mohycs.medium.com/how-to-crack-the-coding-interview-the-7-phases-of-interview-prep-975569fbc410

Outline

1. Online coding test platforms
2. Things that you will prepare
3. **Big-O**

Big-O

Big-O notation is a crucial concept found in algorithm textbooks. It describes how the execution time (**time complexity**) and space requirements (**space complexity**) of an algorithm scale with increasingly large inputs.

Time complexity specifically refers to the duration required to execute an algorithm, also known as **computational complexity**.

To express this complexity, consider a function that represents the computational complexity for a given **input size n**. In this function, only **the highest order term** is taken into account, while coefficients are disregarded. For instance, if the computational complexity is **$4n^2+3n+4$**, you would focus on **n^2** from the term $4n^2$ as it represents the highest order. In Big-O notation, this complexity is represented as **$O(n^2)$**.

Possible Big-Os

- **O(1)** - No matter how large the input value, the execution time remains constant. It is the best algorithm. Algorithms with constant time are like the Holy Grail, which is surprisingly valuable if you can find it, but you may have to spend a lifetime to find it. Hash table lookup and insertion correspond to this.

- **O(log n)** - Execution time starts to be proportional to the input value. However, the logarithm is not affected by very large input values, so it can be said to be a good algorithm. This is binary search.

- **O(n)** - Such an algorithm is called a linear time algorithm. This is the case for finding the maximum or minimum value in an unsorted list.

- **O(n log n)** - This is the case for parallel sort. Comparison-based sorting algorithms can't be faster than this. If the input is the best, it may be O(n) possible by skipping the comparison. Timsort is one such example.

- **O(n^2)** - Inefficient algorithms such as bubble sort fall into this category.

- **O(2^n)** - This is a case of recursively calculating Fibonacci numbers. Not to be confused with n^2.

- **O(n!)** - TSP problem. Even if the input value is slightly increased, it is difficult to calculate in time. The slowest algorithm.

(skip) Python Modules you may need to install

```
import collections

import heapq

import functools

import itertools

import re

import sys

import math

import bisect

from typing import *
```

String Manipulation

Level 1 - Part 1

List

- **Dynamic array** - changeable size
- **Mutable list** - changeable element
- Very common and useful data type
- Of course, C++ and Java have it too

List

Most importantly, you don’t need to stress over choosing between a queue or a stack when using lists, as lists in many programming languages provide all the operations available on both queues and stacks. Many of these operations can be performed in constant time, *O*(1), such as accessing elements by index.

However, it's important to note that not all operations are this efficient. For example, removing the first element of a list with pop(0)—an operation typical of queues—actually has a time complexity of O(n) because it requires shifting all the subsequent elements one position to the left. Therefore, while lists are versatile, you should use caution and consider efficiency when employing them for queue-like operations.

The possible operations are:

Operations	Complexity
`len(a)`	*O(1)*
`a[i]`	*O(1)*
`a[i:j]`	*O(k)*
`elem in a`	*O(n)*
`a.count(elem)`	*O(n)*
`a.index(elem)`	*O(n)*
`a.append(elem)`	*O(1)*
`a.pop()`	*O(1)*
`a.pop(0)`	*O(n)*

Operations	Complexity
`del a[i]`	*O(n)*
`a.sort()`	*O(n log n)*
`min(a), max(a)`	*O(n)*
`a.reverse()`	*O(n)*

Slicing

```
s = 'hello'
print(s[1:4])
print(s[1:-2])
print(s[1:])
print(s[:])
print(s[1:100])
print(s[-1])
print(s[-4])
print(s[:-3])
print(s[-3:])
print(s[::1])
print(s[::-1])
print(s[::2])
```

Useful functions

```
char.isalnum()

str.lower()

re.sub()
```

String Manipulation

Level 1 - Part 2

isdigit()

```
s1 = "34"

s2 = "12ab"

print(s1.isdigit())

print(s2.isdigit())
```

Lambda function

A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression. For example,

A lambda function that adds `10` to the number passed in as an argument, and print the result:

```
x = lambda a : a + 10

print(x(5))
```

Lambda function with multiple arguments

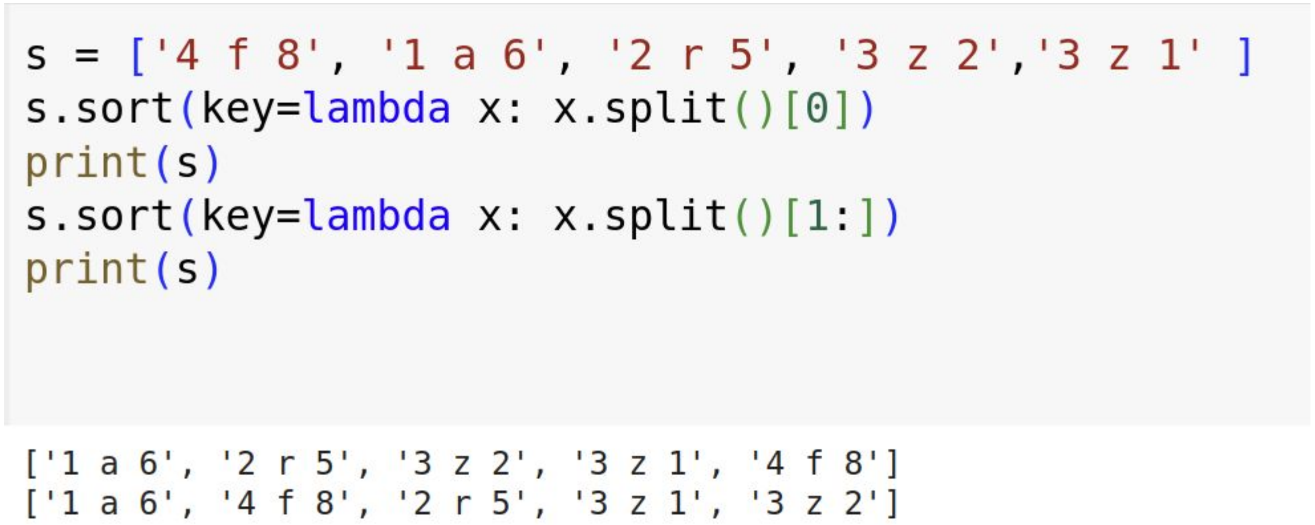
A lambda function that multiplies argument `a` with argument `b` and print the result:

```
x = lambda a, b : a * b

print(x(5, 6))
```

`sort()` and `lambda` function

```
s = ['4 f 8', '1 a 6', '2 r 5', '3 z 2','3 z 1' ]

s.sort(key=lambda x: x.split()[0])

print(s)

s.sort(key=lambda x: x.split()[1:])

print(s)
```

sort() and lambda function



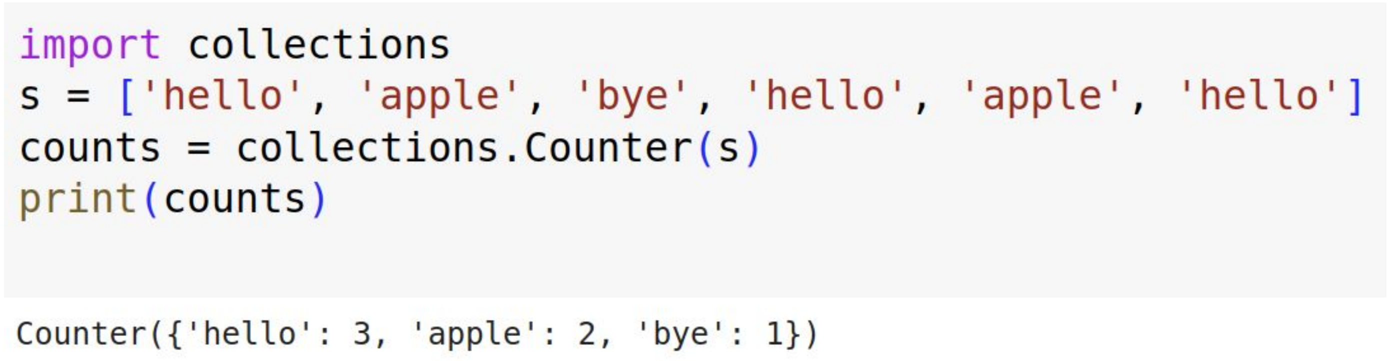
```python
s = ['4 f 8', '1 a 6', '2 r 5', '3 z 2','3 z 1' ]
s.sort(key=lambda x: x.split()[0])
print(s)
s.sort(key=lambda x: x.split()[1:])
print(s)
```

```
['1 a 6', '2 r 5', '3 z 2', '3 z 1', '4 f 8']
['1 a 6', '4 f 8', '2 r 5', '3 z 1', '3 z 2']
```

collections.Counter()

```
import collections

s = ['hello', 'apple', 'bye', 'hello', 'apple', 'hello']

counts = collections.Counter(s)

print(counts)
```

collections.Counter()



```python
import collections
s = ['hello', 'apple', 'bye', 'hello', 'apple', 'hello']
counts = collections.Counter(s)
print(counts)
```

```
Counter({'hello': 3, 'apple': 2, 'bye': 1})
```

sort() vs sorted()

```
s = [3, 2, 1]

print(sorted(s))

print(s)

print(s.sort())

print(s)

d = 'fjsqcdz'

print(sorted(d))

print("".join(sorted(d)))
```