

|   |    |
|---|----|
| Pre-Study questions   | 1  |
| Welcome to the study  | 2  |
| Intro, Setup, and test: (10 mins)                             | 2  |
| Part 1: Baseline. (max 15 mins)                               | 2  |
| Type hint study material: (10 mins)                           | 2  |
| Part 2: With type hints (max 15 mins)                         | 2  |
| Interview (15 mins)   | 2  |
| Tools Required:   | 2  |
| Task description - Part A                                     | 3  |
| Environment Setup   | 3  |
| No Type Hint Rules:   | 3  |
| Testing and Submitting  | 3  |
| Additional Rules:   | 4  |
| Warm-up exercise  | 4  |
| Type Hint Tutorial  | 11 |
| Warm-up exercise  | 11 |
| Quick Reference:  | 11 |
| Variables   | 11 |
| Functions   | 11 |
| Collections   | 12 |
| Union Types   | 12 |
| Classes   | 12 |
| Generics  | 13 |
| Task description - Part B                                     | 14 |
| Forced Type Hints Rules:                                      | 14 |
| Environment Setup   | 14 |
| Testing and Submitting  | 14 |
| Additional Rules:   | 15 |
| Warm-up exercise  | 15 |
| After study Semi-Structured interview about python type hints | 19 |

This is a sample for one participant, the actual order of problems will be changed per participant.

Participant Code: \_\_\_\_\_

Date: \_\_\_\_\_

Actual Time used in pre-study: \_\_\_\_\_

Actual Time used in part1: \_\_\_\_\_

Actual Time used in part2: \_\_\_\_\_

# Pre-Study questions

---

1. How many years of experience do you have in python?  
(<1year, 1~2 years, 2~4 years, 4+ years)
2. What is your experience in python?  
(Academic / Industrial / Other:\_\_\_)
3. What is(are) your primary language(s) for your most recent project(s)?
4. Do you have experience in competitive programming or technical interviews (like Leetcode)?
5. Have you been using python in competitive programming or technical interviews? If so, is it your first/second language in this context?
6. Do you have a Codeforces/AtCoder/LeetCode account or other related experience? What is your rating/achievement there? If you are on LeetCode, how many Easy/Medium/Hard problems have you solved?
7. What are the values of `a` and `b` at the end of the following code?

```
a = Counter([1, 2, 3, 3, 3])  
b = list(a.items())
```

8. We are interested in type hints. Do you know what type hints are in python?
9. How would you do the following with type hints in python? Provide a guess if you don't know.  
*Declare a list of integers named `lis` and initialize it with an empty list.*
10. Do you have any experience with programming language other than python? (written > 300 lines of code)

# Welcome to the study

---

Welcome to our study. In this study, we are interested in whether type hints are helpful for competitive programmers. Python is a dynamically typed language, but there are still type checkers trying to infer types of expressions and give suggestions to the developer based on the inferred type. By manually adding type hints, we can help the type checkers in type inferencing. The `strict` mode in the Pylance type checker will force it to check that all variables are well-typed.

<!--

We are specifically interested in

- whether the autocomplete suggestions generated due to type hints are helpful. Regarding autocomplete, we mean the hints IDE provided to you, and we will observe if you accept them.
- whether strict typing is helpful. In terms of strict typing, we mean a type checker implemented to help you ensure all types are correct before passing to runtime. -->

Under a competitive programming environment, you will need to finish as many questions as you can in a limited time scope (15 mins). We will give you instructions on setting up the environment.

Thank you for your time. Below is today's agenda:

## Intro, Setup, and test: (10 mins)

- Introduction
- Setup your environment
- Try a easy problem

## Part 1: Baseline. (max 15 mins)

- A 15 minutes algorithm challenge set in Python
- No type hints allowed

## Type hint study material: (10 mins)

- Study relevant type hint material.

## Part 2: With type hints (max 15 mins)

- A 15 minutes algorithm challenge set in Python
- Type checker in "strict" mode type hints.

## Interview (15 mins)

- An interview to let you talk about their experience and feelings

## Tools Required:

- Tampermonkey - extension for website (remote): <https://www.tampermonkey.net/>
- Use <https://greasyfork.org/en/scripts/455257-survey-helper> to install tampermonkey extension on chrome.
- Chrome
- Zoom

# Task description - Part A

You will be given a list of competitive programming/technical interview tasks. Aim to solve as many of them as possible in 15 minutes. Completing a task means you pass all hidden tests within the time limit and get an "Accepted" judgment. You don't have to read all of this document before entering part A. Let us know when you are ready.

Write your solution in `x.py` where `x` is the task's id. The starter code is provided in the file, and the problem statement for each task is given in a pdf file separately. Hints are provided at the end of the description.

## Environment Setup

To help us collect and analyze the data, do the following setup steps. We will guide you through this:

1. Join the provided `vscode.dev` link anonymously. Do not click `start` unless guided by us.
2. Install the `survey_helper` VSCode extension and configure the submission endpoint in settings.
3. In the options for the `pylance` extension, set the type checker mode to `off`.
4. When time runs out, you will be guided to a scene with a button to download data. Please download and send it to us.
5. For our study purpose and to reduce your computer load, please use your mouse to switch files

## No Type Hint Rules:

Please don't put type hints in your code for this set of tasks. The type checker will be turned off, meaning you won't see type errors in your IDE.

## Testing and Submitting

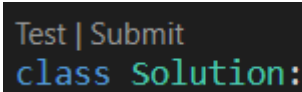
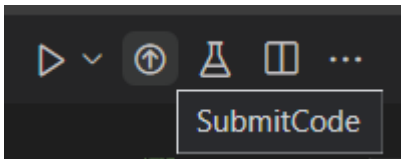
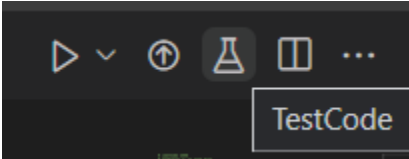
To *test* your code corresponding to `x.in`, where `x` is the task's id. Remember to save your code and the test input before testing. You can use the following methods to test:

- Use `Ctrl+Shift+P` to open the command palette and type `survey-helper.TestCode`
- Click the `test` button above `class Solution:`
- Click the `TestCode` button in the navigation bar

To *submit*:

- Use `Ctrl+Shift+P` to open the command palette and type `survey-helper.SubmitCode`
- Click the `submit` button above `class Solution:`
- Click the `SubmitCode` button in the navigation bar

Sample place to `test` and `submit`

| Above code   | In navigation bar   | In navigation bar   |
|--|---|---|
|  |  |  |

The input in `x.in` has a special format. See the "Custom Test Format" sections of the problem statements for details.

During *testing*, stdout will be printed, and stderr will be ignored. You will see the following judgments:

- Correct: Your answer is correct on the custom input, and your answer will be printed.
- Wrong Answer: Your answer is incorrect on the custom input, and your answer and the expected answer will be printed.
- Invalid Input: Your custom input does not conform to the constraints specified or does not follow the special format.
- Time Limit Exceeded: Your program runs for too long on the custom input.
- Runtime Error: Your program crashed on the custom input. Stacktrace will be printed.

During a *submission*, stdout and stderr are both ignored. For all judgments except "Accepted", you will see a short description of the hidden test you failed, but you won't see the content of the hidden test. You will see the following judgments.

- Accepted: You completed the task successfully.
- Time Limit Exceeded: Your program ran for too long on a hidden test.
- Wrong Answer: Your program is not correct on a hidden test.
- Runtime Error: Your program crashed on the custom input. Stacktrace will *not* be printed.

## Additional Rules:

---

Don't interact with the os filesystem in your solution.

You can consult the official Python documentation here: <https://docs.python.org/3.10/index.html>.

## Warm-up exercise

---

You can use Question E as a warm-up question to get familiar with the testing and submission tools.

## E. Median of List

Given a list of integer, return the median of the list.

### Input

- A list `arr` of integers.

### Output

Return float that is the median of numbers in `arr`.

### Constraints

- $1 \leq \text{len}(\text{arr}) \leq 3 \times 10^5$
- $1 \leq \text{arr}[i] \leq 1 \times 10^9$

### Custom test format

In E.in:

- Put the list of integers on the first line, separated by space.

### Example 1

Input:  
`arr = [1, 2, 3, 4, 5]`

Output:  
3

Custom Input:  
1 2 3 4 5

### Example 2

Input:  
`arr = [1, 2, 3, 4]`

Output:  
2.5

Custom Input:  
1 2 3 4

## Tutorial

First sort the list. Then, do case analysis based on the parity (even/odd) of length of the list.

### A. Most Popular Video Creator

Online video platform A has a lot of videos uploaded to their platform.

You are given a dictionary that stores the video information. Specifically, input `creator_videos_dict` is a dictionary of creators to a list of their videos. Each video is a tuple of its view count and its id.

For example, `creator_videos_dict = {"Alice": [(1, "a"), (3, "b")], "Bob": [(4, "c")]}` means that

- Alice uploaded two videos with id “a” and “b”, which have 1 and 3 views respectively.
- Bob uploaded a video with id “c”, which has 4 views.

The *popularity* of a video creator is the total number of views across all videos they have created. In the example above, Alice and Bob both have a popularity of 4.

Please find the creators with the highest popularity on the platform, and the id of most viewed video by those creators.

If there are multiple creators with the highest popularity, find all of them.

Each video have a unique view count.

#### Input

- A dictionary `creator_videos_dict` with creators as keys and list of `(view_count, video_id)` tuples as values.

#### Output

Return a list of tuples of two strings. The  $i^{th}$  tuple `(creatori, idi)` represents one of the most popular creators `creatori` and the id of the creator’s most viewed video `idi`.

The returned list can be in any order.

#### Constraints

- $1 \leq n \leq 2 \times 10^5$
- There are  $n$  videos in total.
- There are at most  $n$  creators.
- Creator names and video ids are unique strings of length 10 or less.
- Creator names and video ids consists of english letters and numbers.
- Sum of all view counts is less than  $10^9$ .
- View counts are unique.

## Custom Test Format

In A.in, input one video on each line. For each video, put the name of the creator, id of the video, then the view count of the video separated by spaces.

```
creator video_id view_count
```

### Example 1

Input:

```
creator_videos_dict = {  
    "alice": [(6, "one"), (5, "two")],  
    "bob": [(11, "two")],  
    "chris": [(4, "four")]  
}
```

Output:

```
[("alice", "one"), ("bob", "two")]
```

Note:

alice and bob both have 11 popularity.

alice's video "one" has more views than her video "two".

custom.in:

```
alice one 6  
bob two 10  
alice three 5  
chris four 4
```

### Example 2

Input:

```
creator_videos_dict = {  
    "alice": [(1, "a"), (2, "b"), (3, "c")],  
    "bob": [(4, "d")]  
}
```

Output:

```
[("alice", "c")]
```

custom.in:

```
alice a 1  
alice b 2  
alice c 3  
bob d 4
```

## Tutorial

Go through the input dictionary and create a list of tuples with (creator, popularity). Then, we find the maximum popularity and filter the creators with the highest popularity.



For each of the most popular creators, we find their most popular video, and put the `(creator, video_id)` in the returning list.

## G. Coin Change

You are given a list of coin denominations. You have an infinite supply of coins of each listed denominations. You are also given a target value. Return the minimum number of coins needed to make up the target value exactly. Return -1 if it can't be done.

### Input

- A non-empty list `coins` of unique positive integers. It represents the denomination of the coins.
- A positive integer `target`.

### Output

Return the minimum number of coins needed to make up the target value. Or -1 if it can't be done.

### Constraints

- $1 \leq \text{len}(\text{coins}) \leq 500$
- $1 \leq \text{target} \leq 10^5$
- $1 \leq \text{coins}[i] \leq 1 \times 10^9$

### Custom test format

In G.in:

```
coins (space separated)
amount
```

### Example 1

```
Input:
coins = [3, 5, 2]
target = 11
```

```
Output:
3
```

```
Explanation:
11 = 5 + 5 + 3
```

### Example 2

```
Input:
arr = [3]
target = 5
```

```
Output:
-1
```

## Example 3

Input:  
arr = [5]  
target = 3

Output:  
-1

## Tutorial

Dynamic programming: For integer  $i$ , let  $dp_i$  represent the minimum amount of coins to make up  $i$ , or  $\infty$  if it can't be made up using denominations in **coins**.

Then,

$$dp_i = \begin{cases} \infty & i < 0 \\ 0 & i = 0 \\ \min_{c \in \text{coins}} (dp_{i-c} + 1) & i > 0 \end{cases}$$

Calculate  $dp_i$  from  $i = 0$  to **target** and return  $dp_{\text{target}}$ .

# Type Hint Tutorial

Python 3.5 added type hint which can give Python type checkers hints for type inference. This document serves as a tutorial for using type hints in competitive programming in Python3.10.

## Warm-up exercise

You can use Question F as a warm-up question to get familiar with type hints.

## Quick Reference:

```
# primitives
count: int = 0
pi: float = 3.141593
nada: None = None

# function
def area(radius: int) -> float:
    return pi * radius ** 2

# containers
side_lengths : tuple[int, bool] = (1, False)
arr: list[float] = [0.0, 1.2, 3.5, pi]
ages: dict[str, int] = {"Alice": 28, "Bob": 73}
counter: defaultdict[str, int] = defaultdict(int)
```

## Variables

To hint that variable `a` will have type `T`, we write `a: T`

For example:

```
# count is an integer initialized to 0
count: int = 0
# pi is a float initialized to 3.141593
pi: float = 3.141593
# nada is None, this will be useful later when we have Union types.
nada: None = None
```

## Functions

To hint that a function `func` take parameters `param1`, `param2` of type `T1`, `T2` and returns type `R`, we write

```
def func(param1: T1, param2: T2) -> R:
    <function body>
```

For example:

```
# computes area of circle with integer radius, returns float
def area(radius: int) -> float:
    return pi * radius ** 2

# no return is equivalent to returning None, and returning None can be omitted
def send(message: str):
    send_queue.append(message)
```

## Collections

In Python 3.10, collection item types can be added to brackets, preceeded by the collection name. For example:

```
# tuple of (integer, boolean)
side_lengths : tuple[int, bool] = (1, False)
# list of floats
arr: list[float] = [0.0, 1.2, 3.5, pi]
# dictionary of string keys and int values
ages: dict[str, int] = {"Alice": 28, "Bob": 73}
# dictionary of string keys, int values, and default value 0
counter: defaultdict[str, int] = defaultdict(int)
```

## Union Types

We can hint for multiple possible types using the `|` operator. To hint a type might be `T1` or `T2`, use `T1|T2`. Unioning `T1` with `None` type is the same as an optional type `Optional[T1]`.

For example:

```
# Returns an integer or None type. int|None is the same as Optional[int]
def get(key: str) -> int|None:
    if key not in private_dictionary:
        return None
    return private_dictionary[key]
```

## Classes

Name of classes can be use directly as the type of its objects.

For example:

```
class A:
    pass

a: A = A() # a is an object of class A
```

Instance variables types are declared in the class body

For example:

```
class A:
    # instance variable integer x with default value 100
    x: int = 100

print(A().x) # prints 100
```

# Generics

You probably won't need this, but in case you want a generic type variable, use `TypeVar` to declare one.

For example:

```
T = TypeVar('T')
# The identity function, the returned type is the same as input type.
def identity(x: T) -> T:
    return x
```

# Task description - Part B

You will be given a list of competitive programming/technical interview tasks. Aim to solve as many of them as possible in 15 minutes. Completing a task means you pass all hidden tests within the time limit and get an "Accepted" judgment. You don't have to read all of this document before entering part A. Let us know when you are ready.

Write your solution in `x.py` where `x` is the task's id. The starter code is provided in the file, and the problem statement for each task is given in a pdf file separately. Hints are provided at the end of the description.

## Forced Type Hints Rules:

For this set of tasks, we will set the type checker to `strict` mode. You should resolve all type warnings from the type checker before testing or submission.

## Environment Setup

To help us analyze the data and for your convenience, setup the following before actual start, let us know if you need any help:

1. Join the provided `vscode.dev` link anonymously. Do not click `start` unless guided by us.
2. Install the `survey_helper` VSCode extension and configure the submission endpoint in settings.
3. In the options for the `pylance` extension, set the type checker mode to `strict`.
4. When time runs out, you will be guided to a scene with a button to download data. Please download and send it to us.
5. For our study purpose and to reduce your computer load, please use your mouse to switch files.

## Testing and Submitting

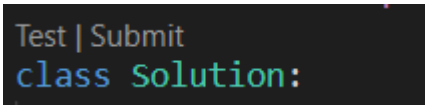
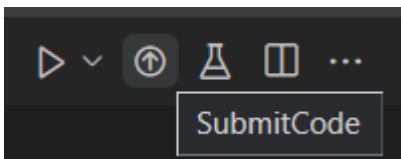
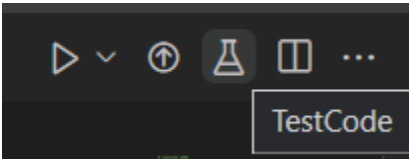
To *test* your code corresponding to `x.in`, where `x` is the task's id. Remember to save your code and the test input before testing. You can use the following methods to test:

- Use `Ctrl+Shift+P` to open the command palette and type `survey_helper.TestCode`
- Click the `test` button above `class Solution:`
- Click the `TestCode` button in the navigation bar

To *submit*:

- Use `Ctrl+Shift+P` to open the command palette and type `survey_helper.SubmitCode`
- Click the `submit` button above `class Solution:`
- Click the `SubmitCode` button in the navigation bar

Sample place to `test` and `submit`

| Above code   | In navigation bar   | In navigation bar   |
|--|---|---|
|  |  |  |

The input in `x.in` has a special format. See the "Custom Test Format" sections of the problem statements for details.

During *testing*, stdout will be printed, and stderr will be ignored. You will see the following judgments:

- Correct: Your answer is correct on the custom input, and your answer will be printed.
- Wrong Answer: Your answer is incorrect on the custom input, and your answer and the expected answer will be printed.
- Invalid Input: Your custom input does not conform to the constraints specified or does not follow the special format.
- Time Limit Exceeded: Your program runs for too long on the custom input.
- Runtime Error: Your program crashed on the custom input. Stacktrace will be printed.

During a *submission*, stdout and stderr are both ignored. For all judgments except "Accepted", you will see a short description of the hidden test you failed, but you won't see the content of the hidden test. You will see the following judgments.

- Accepted: You completed the task successfully.
- Time Limit Exceeded: Your program ran for too long on a hidden test.
- Wrong Answer: Your program is not correct on a hidden test.
- Runtime Error: Your program crashed on the custom input. Stacktrace will *not* be printed.

## Additional Rules:

---

Don't interact with the os filesystem in your solution.

You can consult the official Python documentation here: <https://docs.python.org/3.10/index.html>.

## Warm-up exercise

---

You can use Question E again as a warm-up question to get familiar with the testing and submission tools.



## E. Median of List

Given a list of integer, return the median of the list.

### Input

- A list `arr` of integers.

### Output

Return float that is the median of numbers in `arr`.

### Constraints

- $1 \leq \text{len}(\text{arr}) \leq 3 \times 10^5$
- $1 \leq \text{arr}[i] \leq 1 \times 10^9$

### Custom test format

In E.in:

- Put the list of integers on the first line, separated by space.

### Example 1

Input:  
`arr = [1, 2, 3, 4, 5]`

Output:  
3

Custom Input:  
1 2 3 4 5

### Example 2

Input:  
`arr = [1, 2, 3, 4]`

Output:  
2.5

Custom Input:  
1 2 3 4

## Tutorial

First sort the list. Then, do case analysis based on the parity (even/odd) of length of the list.

## B. Equality Satisfiability

You are given a list of equalities and inequalities among symbols. Determine whether the list of equalities and inequalities can be satisfied by assigning symbols with integers.

### Input

**equations:** A list of tuples containing three items  $(l, op, r)$ :

- $l$  is a string, the symbol of the left operand
- $op$  is either "==" or "!="
- $r$  is a string, the symbol of the right operand

### Output

Return **True** if the list of equations can be satisfied by some assignment of integers to symbols. Otherwise return **False**.

### Constraints

- There are  $n$  distinct symbols.
- $1 \leq n \leq 3 \times 10^5$
- $1 \leq \text{len}(\text{equations}) \leq 2n$
- Length of the symbols `equations[i][0]`, `equations[i][2]` are at most 10 characters.
- The characters in the symbols are either English letters or numbers.
- The operands `equations[i][1]` are either "==" or "!="

### Requirement

Please use the given Disjoint-Set (Union-Find) data structure to complete this problem.

### Custom Test Format

In **B.in**:

- Put one equation on each line.
- For each equation, separate the symbols and the operand with at least one space.

### Example 1

Input:

```
equations = [  
("a", "==", "b"),  
("b", "==", "c"),
```

```
("c", "==", "a")
]
```

Output:  
True

Note:  
We can assign "a", "b", and "c" with 1 to satisfy all equations.

```
custom.in:
a == b
b == c
c == a
```

## Example 2

```
Input:
equations = [
("a", "==", "b"),
("b", "==", "c"),
("c", "!=", "a")
]
```

Output:  
False

Note:  
There is no assignment that satisfy the equations

```
custom.in:
a == b
b == c
c != a
```

## Tutorial

Use the disjoint set (union find) data structure.

Firstly, for each equality, union the two operands.

After dealing with all equalities, for each inequality, check if the two operands are in the same sets.

If the two operands are in the same set for one inequality, then the equations cannot be satisfied. Otherwise, it can.

# After study Semi-Structured interview about python type hints

---

1. How good did we simulate the interview/CP environment today?
2. Have you ever seen type hints before?
  - Do you find them useful before?
  - In what context?
  - Why or why not?
3. Are type hints useful based on today's experience? Why or why not?
  - This question is meant to get some initial thoughts from the participants. Ask the following questions instead if they don't know what to say.
4. Cons: Do find getting the type hints right wastes a lot of your time?
  - What aspect of it wastes time?
  - Writing the type hints down?
  - Editing type hints later?
5. Are type hints helping you in some way?
  - Does type errors given help you eliminate errors before testing/submission, or does it give you unnecessary type errors that won't come up in testing/submission? Examples?
  - Are autocomplete suggestions helpful or are they blocking you from doing things? Examples?
  - What could be improved for type hints? Like in the interacting process between you and IDE, what might need to be improved in your view?
6. (Especially if their main language for interview/CP is not Python.) Compare type hints with type systems in your main language.
7. Are you going to use type hints in future interview/CP contexts? What about other situations?
  - Data Analysis
  - ML
  - Web dev
  - Scrapers
  - Education
  - Software prototyping
  - etc.
8. To whom would you recommend type hints? Which feature of type hints will you recommend people to use?
9. Would you like to learn more about type hints? Any specific features in mind?
10. Supplemental question: What else do you want to say?