Python type hint in Competitive Programming

Are Python Type Hints helpful in competitive programming?

Shaokang Jiang[1] and Eric Ma[1]

[1] *University of California, San Diego*

Abstract

Type hinting, which is a way to statically indicate the type of a value in the code, was introduced in Python 3.5. Before Python 3.5, python is a dynamic language in which variable type can only be inferred during runtime. Competitive programmers(CP) are a neglected group of people who need to solve algorithm challenges as fast as possible. In this project, we are trying to assess if using Type Hints is helpful for competitive programmers. We conducted a pilot study with 5 people in different background, 1 of them only provide interview data. We found that type hints may not be useful for CP in Python, and autocomplete suggestions are often not good enough to be accepted. Survey indicates that People also dislike type hints in Python for CP tasks. *Keywords*: Type hint. Competitive programer. Python. Autocomplete.

1 Introduction

Competitive programming (CP) is the sport of solving algorithmic puzzles. The International Collegiate Programming Contest (ICPC) is a global CP contest for college students. ICPC has been around since 1977 and its world finals is highly competitive. Successful contestants must possess different kinds of ability to be able to succeed in CP competitions. On one hand, participants have know a wide variety of algorithms and tricks, so that the program written solves the task correctly and executes within the given time and space bound.On the other hand, the participants must be able to implement programs fast, as they are competing against each other in real time.

Coding interviews questions are often an easier version of CP puzzles, tech companies use these questions to evaluate software developer applicants' proficiency in algorithms and programming. Although it is not competitive, interviewees are still required to be able to come up with a correct algorithm, and produce bug-free code in a short amount of time.

Historically, **Python** is not used or taught as a competitive programming language. Python is an interpreted langauge, and is slower in execution compared to compiled programming langauges such as C++ and Java. For this reason, Python is often ignored when it comes to competitive programming, where execution time and space is limited. The appearance of the JIT compiler pypy reduced the execution time of Python scripts by 5 time compared to CPython [1], which makes doing competitive programming in python a possibility. We observed some advanced competitive programmers switching to Python for easier problems for its expressive syntax and easy to use library.

Another difference between C++ and Python is the type system. While C++ is statically typed, Python is a dynamically typed. In Python version 3.5, **type hints** are introduced for Python. Programmers can annotate variables and functions with type information. Some tools, including the Pylance extension in VSCode, are able to utilize type hints to provide type checker support and generate more comprehensive autocomplete suggestions, effectively making Python statically typed [2].

```
a : int = 0
def isLower(a : str) -> list[bool]:
    return [c.islower() for c in a]
```

There is a lot of resource on the different algorithms used in programming contests and interviews, including online judge Codeforces [3], interview preparation site LeetCode [4] etc.. There are also research about how to effectively teach these algorithms [5] [6]. However, the topic of implementation speed is rarely talked about. In this paper, we focus on how Python type hints affect the speed in which competitive programmers implement algorithms.

In this paper, we conducted a study on the research question:

RQ Is Python Type Hint helpful in competitive programming?

PLATEAU
13th Annual Workshop at the Intersection of PL and HCI

DOI: 10.35699/1983-3652.yyyy.nnnnn

Organizers:
Sarah Chasins, Elena Glassman, and Joshua Sunshine

Based on this research question, we are specifically interested in the following sub questions:

RQ1 Does type hint help contenstants complete problems faster in CP?

RQ2 Does type hint help contenstants do first-time implementation faster in CP?

RQ3 Does type hint help contenstants debug faster in CP?

RQ4 Do autocomplete suggestions help when type hints were added in CP?

RQ5 What is the sentiment towards using type hints, type checker, and autocomplete suggestions in CP?

We designed and conducted a 2-by-2 pilot experiment with four participants. Then, we observed the effect of different aspects of typing in Python on the programmer.

Section 2 will introduce the method of the experiment. Section 3 will present the results from the pilot experiments. Section 4 will mention the limitation and threats to validity of our experiements. Section 5 will discuss and interpret the results in detail.

2 Method

This section will breifly talking about structure of the study and the working environment of our study. A sample material, contains all paper material for one participant, is available at sample study material. The algorithm challenge question statement and strater code is available at question list

2.1 Recruitment process

As a pilot study and limited by the time scope, we didn't do a formal recruit process. But we still try to find people who meet our traits below: a) Collegestudents/recent graduates. b) Have a Codeforcesrating of 1200 to 1500 or "equivalent" Leetcoderating per problem count. c) Did algorithm challenge (Leetcode/Codeforces/etc) in Python before. d) familiar with python. e) Have not used python type hint before. f) The current primary language is python. This study is designed to send pre-study questionair and recruit from Codeforces/Leetcodeplatform, ICPC Clubs, LeetcodePractice groups.

To assure participants meet the requirement, we provide a pre-study questionair, which includes some python coding questions, like correct python codes, generate expected output. And some questions asking about their experience with python.

2.2 Study Procedure

Entire procedure at a glance Fig. 1 shows the basic procedure for the entire study. An experiement is a 15 minute algorithm challenge and participants need to finish it on our provided platform. They are able to read question and understand algorithm before they click start. At the very beginning of the study, participants are required to 1) Install Zoom, Chromium based browser, Tampermonkey extension for Chrome 2) Install survey-helper Tampermonkey extension 3) Join the provided link. 4) Install the survey_helper VSCode extension and configure the submission endpoint in settings.

Training Problem A training problem will be provided in this part to help participants get familiar with our study environment, which is built on top of VScode, refer to Fig. 2 for more info.

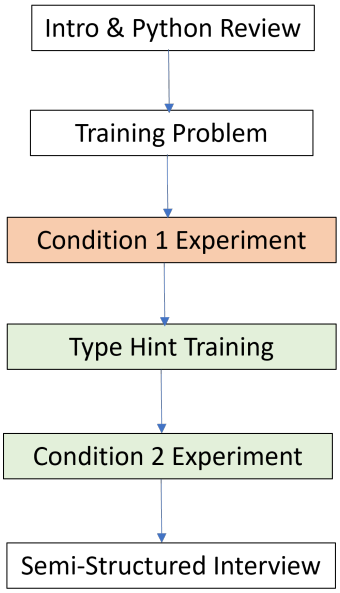


Figure 1. the entire procedure

Condition 1 experiement An experiement where using type hints is not permitted. Any function related to type hints will be turned off. We are going to provide a strater code which doesn't have any type hints as well. Pylance type checking mode will be set to "basic".

Type Hints Training In this part, we teach what is the type hints and how to use the desired features. We also provide one warm-up questions at the end for participants to practice before the formal start.

Condition 2 experiement An experiement where using type hints is permitted. We are going to provide a strater code which type hints. And Pylance type checking mode will be set to "basic". And participants are required to resolve all type checking warnings

In the actual study, condition1 and condition2 might be swapped to reduce learning effect.

Semi-structured interview This interview contains around 10 questions to let participants talk about their experience and feelings. The detailed questions list can be checked from our sample study material.

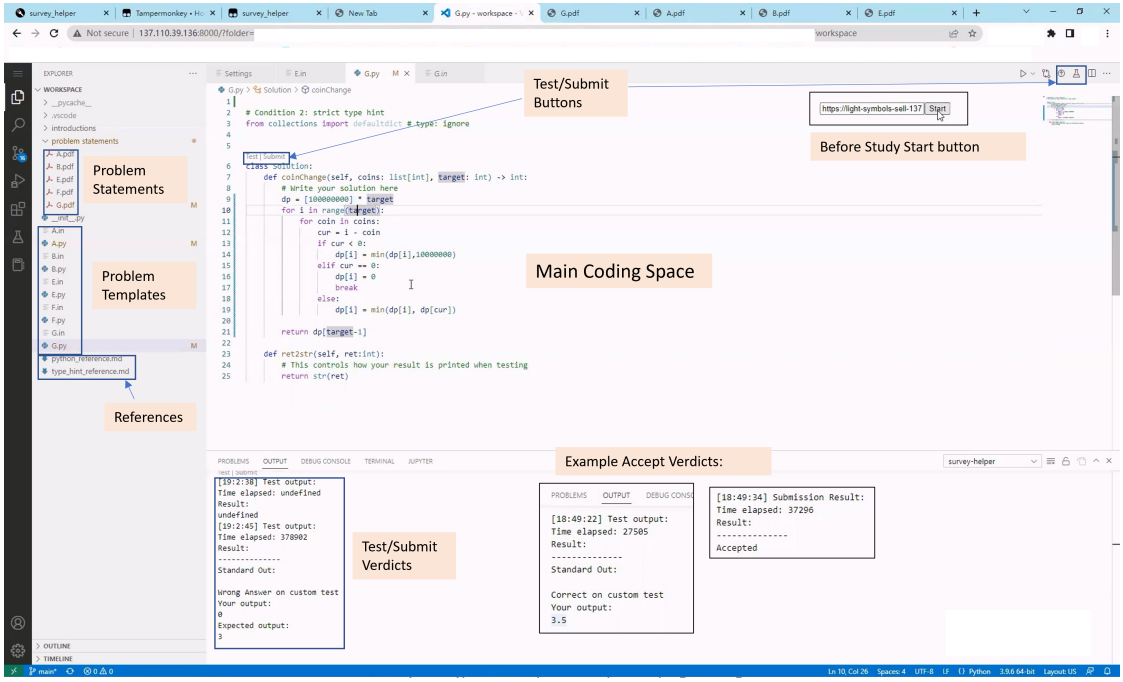


Figure 2. sample coding interface

Workspace Fig. 2 shows a sample participant screen. When participants are ready, they can click the start button. During the study, they need to press test/submit button in multiple places to send their code to our server and judge. The server result will be printed in the "survey-helper" output channel in their working space. Participants will receive notifications when there are 5 minutes left and 10 minutes left.

Once 15 minutes time countdown finished, workspace will do a final submission and workspace will be closed automatically. A new window will pop out for participants to download the recorded data from their local browser. And we will kindly ask participants to send data back to us. Participants always have the right to check the recorded data before sending to us.

Questions selection and arrangement As a competitive programming challenging, algorithm challengings are the core. For all problems, we follow icpc problem format and select simple (600 1000 CF rating) but require non-trivial work (e.g.use special data structure) to complete. We will provide enough challenges that the participants won't be able to finish in the 15-mins time window. Because we are assessing type hints' usage instead of really challenging participants, we also provide tutorials for each question, which might include text/pseudocode. So participants won't get stuck if they don't know the answer. In order to make sure participants want to finish as much question as possible, we provide incentive for each successfully completed challenge.

In terms of question arrangement, We gave each participant a different question configuration to reduce the problem difficulty effect. We also reordered the questions to avoid learning effect. Table. 1 shows the actual question order participants was doing.

	Part 1	Part 2
P1	Condition 1: B	Condition 2: G+A
P2	Condition 2: B	Condition 1: G+A
P3	Condition 2: G	Condition 1: B
P4	Condition 1: G	Condition 2: B

Table 1. Participant Question arrangement

Short question description:
A. Most Popular Creators (Data Structure)
B. Equation Satisfiability (Data Structure)
E. Median of List (Easy - Training Problem)
G. Coin Change (Dynamic Programming)

Function signature In the experiment, we want to assess a situation where people need to manually type type hints to reach functionality related to type hints. The intuition of doing this is that we are kindly considering whether type hint is useful under a worst case scenario, where participants need to manually type it. In most cases in the real world, people have the ability to reach some intelligent tools, such as pylance, which will infer some types automatically in the background. Sample provided function signature's different is shown in Fig. 3.

In order to enforce participants not using any type hints in condition1 and using type hints in condition2, we provide different function header for each part for the same question. For condition1, we want to simulate an environment with the minimal intelligent typeinference. We observed that removing type hints in the function signature greatly reduces Pylance's ability to infer types. By using this way, we minimize variable-relevant autocomplete. In part2, by adding type hints in the function signature and permit participant to use type hints, we can observe if type hints is useful.

```
class Solution:
def arrayMedium(self, arr):
    # Write your solution here
    return 0
```

```
class Solution:
def arrayMedium(self, arr: list[int]) -> float:
    # Write your solution here
    return 0
```

Figure 3. **upper**: method signature for condition1 (without type hints) **lower**: method signature for condition2 (with type hints)

2.3 Data Collection

Fig 4 shows the data collection flowchart of our study. During the study, participants need to connect to our provided judge server to test and submit their code and the judge server will record test and submit actions and the corresponding results. Zoom will do a video record to record the entire study. The locally tampermonkey extension will record keystrokes, clicks and autocomplete suggestion status. In terms of the recorded keystrokes, we are most interested in the keypress in Table. 2 with autocomplte observed.

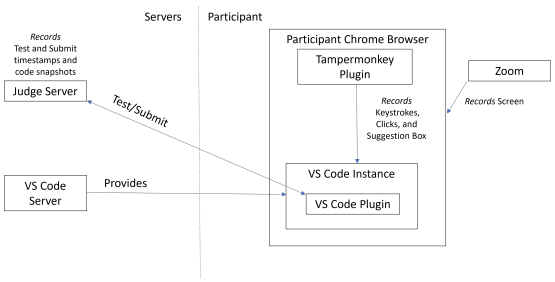


Figure 4. Data collection overview

Keybinding	Event
Ctrl+Space	Call autocomplete
Esc	Cancel autocomplete
Enter, Tab	Accept autocomplete
Arrow key	Navigate autocomplete
any other key	Ignore autocomplete

Table 2. keystrokes we are focusing on with autocomplete observed

2.4 Data analysis

Quantitative : The majority of quantitative data comes from the recorded data during the experiments, we also consult video recording when we find abnormal events. We use java, python, excel to categorize, analyze, and plot the recorded data.

Qualitative : The majority of qualitative data comes from the semi-structured interview conducted at the end of the entire experiment, we also consult video recording when needed. This study is designed to be using semantic analysis by first generating codes from participants' transcription. And then categorize, summarize over all codes. Due to time constraints, we generate codes and themes mainly from notes we took during piloting and look at the transcription when needed. Transcription is generated by Zoom, Microsoft Service, and iflytek. We corrected some errors in transcription using video recording.

3 Results

Since we only have 4 participants, none of the quantitative results are statistically significant. But we can still see trends from them and infer potential next steps.

3.1 Participants

We recruited 5 participants for the experiment. Their experience background information is shown in Table 3. P0 did the experiement before we implemented the data collection mechanism, so the experiement results will be discarded for P0, we will only use the interview results from P0. We collected quantitative data from P1, P2, P3, and P4, which will be presented later.

It is unfortunate that we did not recruit competitive programmers who mains in Python. Instead, P1 and P2 are both competitive programmers who mains in C++, while P0, P3 and P4 have less experience in algorithm puzzles, but more experience in Python.

	Background Info about Python	Background Info about Coding Interview/Competitive Programming)
P1	Undergraduate CS student. Written some Web applications (Django, Flask) 3 months ago for his internship.	Primarily uses C++ for CP. Codeforces rating about 1750. ICPC-North America Championship participant Uses Vim as editor.
P2	Undergraduate CS student. Uses Python for deobfuscation and text manipulations. Last used last month. Does security/reverse-engineering as a hobby.	Primarily uses C++ for CP. Codeforces rating about 1500 (probably underrated). ICPC-North America Championship participant Uses Clion as editor
P3	Software Developer at a tech company. Use Python for some interview preparation. Use Python for scripting during work.	Did interview preparation on LeetCode for 1 month in Python 2 months ago, and in C++ for 6 months a year ago. Use Clion and Pycharm as editor.
P4	Software Developer at a tech company. Often use Python to write scripts for parsing and data analysis work. Industry Work. Use Visual Studio as editor	Not much experience in CP/coding interview.
P0	Graduate CS Student. Using Python for everything.	Prepared coding interview on Leetcode using Python for 6 months starting 3 months ago. Use Jupyter Notebook as editor.

Table 3. Participant background information

3.2 Quantitative Result

Figure 5 shows the timeline of observed activities from the experiments. The top half is result from Condition 1. The bottom half is result from Condition 2. The horizontal coordinate is the time of the experiement. The vertical coordinate is the 4 participants. Background color signifies the problem they are working on. Vertical lines are test and submit actions, and color represent the judge verdict. Markers in the middle on the center lines are instances of autocomplete suggestions, where different marker represent if participant reject, accept, or ignored the suggestions. Using the result, we tries to answer the following research questions:

RQ1 Does type hint help contenstants complete problems faster in CP?

RQ2 Does type hint help contenstants do first-time implementation faster in CP?

RQ3 Does type hint help contenstants debug faster in CP?

RQ4 Do autocomplete suggestions help when type hints were added in CP?

Given the research qeustions, we gathered corresponding information from the raw data and presented in Figure 6.

A **Completion time**

Completion time is the total time used to implement the algorithm correctly. It is represented by the total amount of time taken to receiving an "accepted" verdict. Figure 6b shows the completion time. Data suggests that Condition 1 has faster completion time.

B **First-pass implementation time**

We hope to see if type hint influence how fast a programmer write their first version of code. First-pass implementation time is the time to program the initial version of the solution. We used two quantities to approximate the first-pass implementation time: the first test time, and the first submit time. Both of them are presented side-by-side in Figure 6c. As we can see, the first pass time is generally higher for Condition 2 than Condition 1.

C **Debugging time**

Debugging time is the time used on debugging. We used the time from first meaningful test to problem completion to approximate the debugging time. The first meaningful test is determined by looking at the code they tested. If it is very clear that an early test is on an incomplete code, then we discard this test and consider the next test as first meaningful test. Figure 6d shows the debugging time. The result is inconclusive. It seems that for problem B, Condition 1 has higher debugging time. And for problem G, Condition 2 has higher debugging time.

D **Autocomplete acceptance ratio**

We want to study the usefulness of autocomplete suggestions. Autocomplete acceptance ratio is the ratio of the count of accepted suggestions to the count of all given suggestions. Data is shown in Figure 6a. The acceptance rate is generally low. The highest acceptance rate is 17.07% from P1 in Condition 2. The acceptance ratio is also generally higher for Condition 2 than Condition 1.

3.3 Qualitative result

This section contains the result from the post-study interview. We answer the research question:

RQ5 What is the sentiment towards using type hints, type checker, and autocomplete suggestions in CP?

We summarize codes from notes taken from the interview, and refer back to the video recording of the interview when needed. The following are some common topics we found.

A **Autocomplete Suggestions are sometimes annoying**

P2 suggests autocomplete is distracting, "When you are thinking about complex problems, [autocomplete suggestion box] popping up may disrupt your thinking."

P2 also suggested that autocomplete boxes interfere with the action of inputting end-of-line character. Usually, one would press enter to get a new line. However, when autocomplete is up, pressing enter will accept the autocomplete, instead of giving you a new line. You'll need to either reject the suggestions beforehand, or erase the extra characters that autocomplete give you.

P0 and P4 suggest there is no impact of having autocomplete or not.

From the recording, we know that P4 is the only person using arrow key to select elements. In

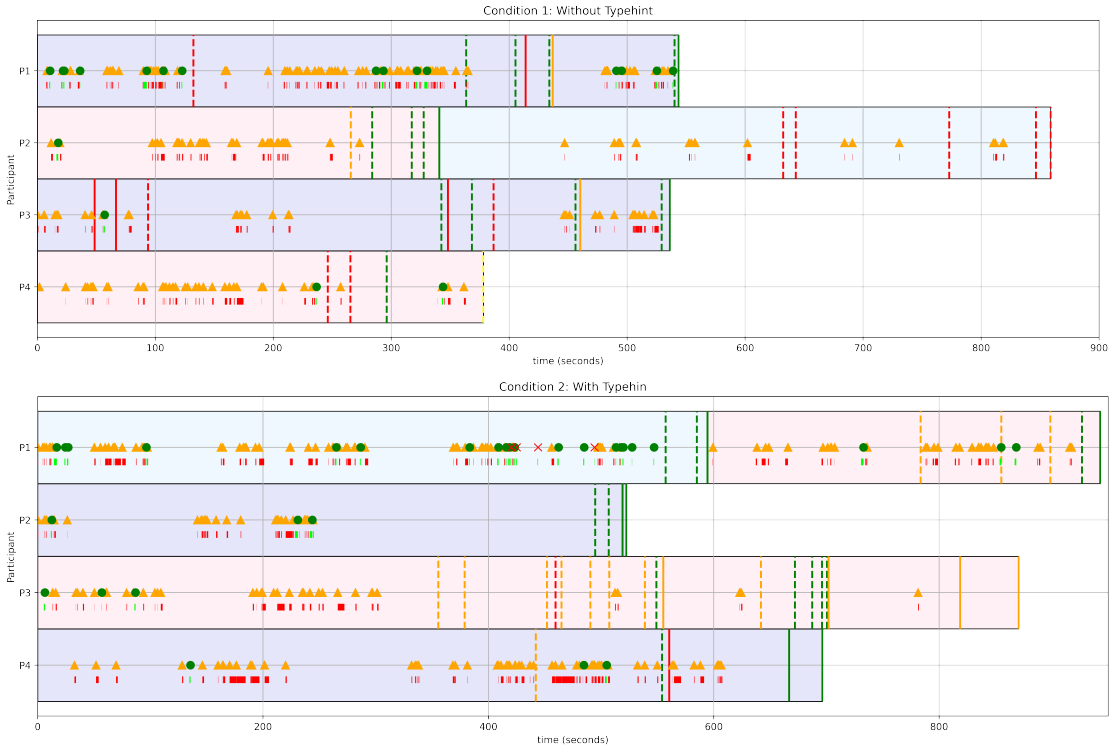
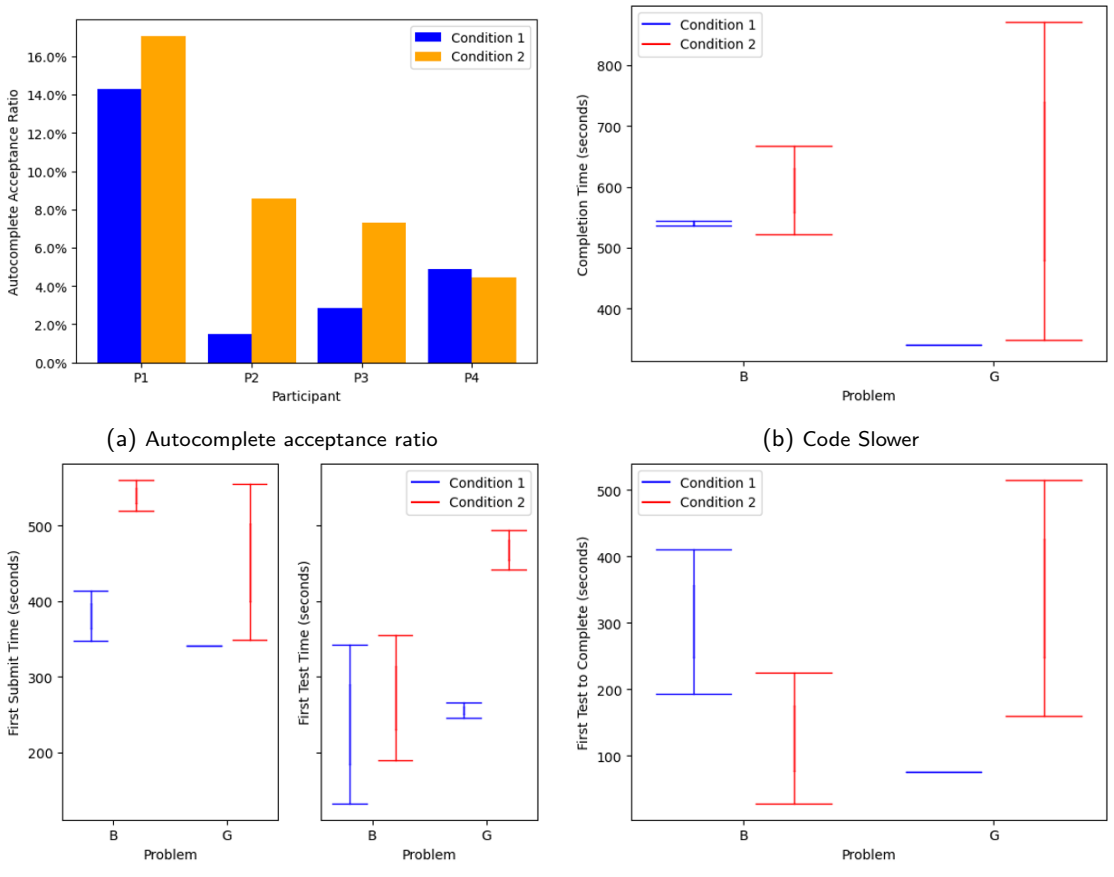


Figure 5. Timeline of observed activities legends as below:
Background color: aliceblue as Problem A, lavender as Problem B, lavenderblush as Problem G,
Test/submit actions: vertical dashed line as test, vertical solid line as submit, vertical green line as Accept, vertical orange line as Wrong answer, vertical red line as Runtime error,vertical yellow line as Crash,
autocomplete status: Yellow triangle as Ignored, Green dot as Accept Red cross as Reject



(a) Autocomplete acceptance ratio

(b) Code Slower

(c) First passing time

(d) Debugging Time

Figure 6. Quantitave Analysis Result Graph

condition 2, they used arrow key 3 times but ignored or cancelled at last.

B **Autocomplete Suggestions is useful for completing variable names**

P1, P2 and P3 all suggested that the suggestions to complete variable names are good. However, P2 pointed out that for short variable names, the autocomplete suggestion won't show up, an thus autocomplete is useless in this case. They also pointed out that a lot of people write short variable names in CP contests, which seems true from a quick scan over a few Codeforces submissions from high-rated contestants.

Based on our video observations, most of autocomplete acceptance also happens at completing variable name.

P1 and P3 gave a positive feedback for autocomplete. But data showed that only P1 have a high acceptance rate. P1 tends to have longer variable name than P2. And P1 seems not familiar with python syntax, as in our notes.

Even though P3 suggests autocomplete is useful during interview, considering the relatively low acceptance rate, 2.85% in condition 1 and 7.21% in condition 2, their feeling might not accurate. P4 and P0 says autocomplete has no impact.

C **Writing type hints wastes time**

P4 suggests Type hint syntax is weird: "So to tell the truth, the only thing I feel uncomfortable about is that its syntax is very strange." They say it's different from the typing syntax in C++ and Java, which are their main langauges at work.

P3 suggests that typing Type hint is a waste of time.

P2 didn't need to use any type hints during the experiement.

P1 also thinks type hints wastes time. "But during the competition, you may not want to use type hints. If you spent time on writing types, you may lose your time advantage [in using Python]."

P0 suggest writing type is okay but changing type is boring and time-consuming: "If you are just writing this type [for once], it's OK. It's not a waste of time." and "But I think, especially just now, for example, if I want to change [the definition of] one [variable], [...], I have to change its type hint again when I come back. This is very troublesome."

D **Investigating type error wastes time**

P4 suggest that using type checker in a limited time condition is wasting time.

P2 suggest that they only use the type checker when they are finding the specific position of some bugs, "OK, but for type checker, [...] If there is a compilation error, you can enable [type checker] and take a look at it. However, I usually disable it when writing code." He give the reason that investigating an error is time-consuming and maybe not worth it, "For example, if [the type checker] gives a warning, I will be drived to read it. After reading it, I find that it is actually is a false alarm. This wastes time."

P0 suggest when using Pylance, type checking with "basic" mode might be better than "strict" mode. And "basic" mode is better than no type inferrence at all.

E **Being "Pythonic" is important**

P3 suggests type hints feature for python is not meaningful compared with other programming languages, "Python plus Type Hint is a strictly worse than using a statically typed language like C++."

P4 recommends "Python is just used for scripting and rapid prototyping.". He also suggests that Python is a scripting language instead of a "formal" one. So, people shouldn't make it strictly typed.

F **Project scale affect whether type hints are useful**

P3 suggest debugging in this length of codes may not appeal the fancy part of type hints, saying "Small tasks can be debugged with repeated test runs instead of typing. "

P0 argues that people are able to remember the detailed information of each variable easily for a task we provided, "I can remember all the type information for tasks of size [similar to these algorithm puzzles]."

Upon review of the interview recording, all of the participants, in one way or another, mentioned that type hints might be useful for larger projects.

G **Computer resource limitation**

An interesting point brought by P2 is that their own laptop is not fast enough for type checkers to respond in real time. The delayed messages and suggestions are annoying and interrupt his thinking, so he turns these features off while coding on the said laptop.

4 Limitations

Every study has limitations. Describe any threats to validity in your study. This section may be brief; perhaps just a paragraph.

Except small sample size, we identify several other threats to internal validity of the study.

Participant Background We wish to recruit competitive programmers with similar experience (preferrably about Codeforces rating 1700) competitive programmers who mains in Python. However, we cannot find such a person. Thus, the results might be bias due to participants lower familiarity with algorithms or Python. In fact, in Figure 7, we categorize participant in terms of their algorithm proficiency, we see that the competitive programmers are much faster in completing the tasks.

Even if we could find Python competitive programmers, they still need to learn to use Python Type Hints. Since experience is an important factor, we would ideally want to perform the experiments with participants who are very familiar with type hints. We can achieve this through a more long-term experiment.

Task Selection We selected relative easy problems compared to real CP contest tasks. Thus, our results cannot be generalized to more complex problems.

Environment Setup Our environment setup is quite different from the one used in CP or interview. Participants often use different editors and tools, but we only provided VSCode and Pylance.

Compared to CP contests, we handle input/output differently. CP contests require contestants to read input from stdin and write outputs to stdout, adding two layers of text manipulation. Compared to interviews, as P1 and P3 suggested, we are giving participants access to tools that is normally absent in coding interviews, like autocomplete and type checker. P3 purposefully turns off autocomplete and type checker when practicing coding interview to simulate real interview environment.

Variety of Bugs We gave participants an entire algorithm puzzle to do. It is unpredictable what kind of bugs will be introduced. Even though we required the parcipant to understand the solution algorithm before starting the timer, they might still waste time on logical errors that is unrelated to typing. We might be able to reduce the threat by choosing debugging tasks. However, CP is a comprehensive process. Debugging other's code might be very different from debugging the code that one wrote 5 minutes ago. We kept the monolithic task structure to keep it true to the nature of CP. Other methods need to be taken to reduce this validity threat.

Ignored Suggestions Most of the autocomplete suggestions are ignored instead of rejected by pressing the escape button. However, we treated ignored suggestions the same as rejected suggestions. P1 suggested that autocomplete suggestions have a guidance effect. While they might not press enter to accept a suggestion, they would look at the suggestions and type-copy what he thought was the correct token to use. It is not known if other participants are doing the same without noticing.

5 Discussion

Interpret your results. Why do you think you found what you did? Were there any surprises?

The quantitative result is not so convincing due to the nature of the small pilot study. We don't have a statistically significant sample size to say things for sure. We also lost some data since our judge server crashed on the experiement given to P4. If we ignore the limited sample size, we can infer the following about the research questions:

RQ1 Type hint does not help reduce the overall coding time in CP.

RQ2 People wastes time on using type hints even before debugging.

RQ3 The result about effect of type hint on debugging time is inconclusive.

RQ4 Autocomplete is often ignored. But its usage is higher when used with type hints than without.

From the interview results, we can infer the answer to the fifths research question:

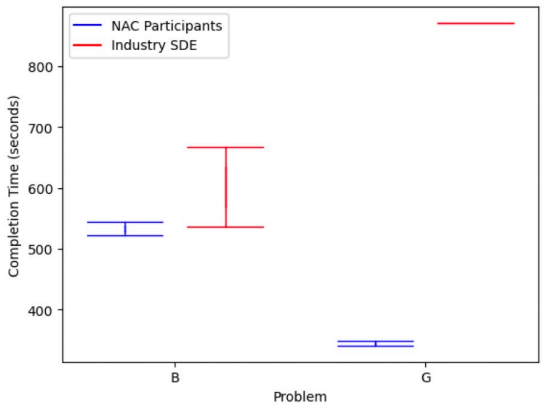


Figure 7. Completion Time by Experience

RQ5 Contestants dislikes type hint and the type checker during competitive programming. They have mixed feeing towards the autocomplete feature.

It is interesting to see that type hints are not helping people in CP problems.

We originally assumed that people would take longer to program in the first pass since they need to figure out the type of hard-to-infer types and type it out. We also expect programmers to easily find and fix some bugs related to types as they were coding the first time since the type checker give red underline to parts that doesn't type check. This is confirmed by our experiement.

We also thought that it would take less time for programmers to fix their program after testing their program, since the bugs related to types were eliminated in the first-pass. However, our experiement result neither support nor deny the claim. Upon closer look at the submission data, we found that P2 did not encounter a bug when coding problem B in condition 2. Participants all encountered bugs in one way or another in other cases. P2 is one of the ICPC-NAC participatns, who is very familiar with B's algorithm. It is not suprising for him to not have a logical error in this case. If participant B were to encounter a logical error in problem B, his debugging time will increase and our result might be different. We might conclude that "Type hint does not help with debugging time." This is suprising to us. But at the same time, none of the participants are familiar with python type hints, so maybe the cause is not type hints itself, but the experience with it.

The result for autocomplete shows that the VSCode feature has a lot to improve. It is not giving the wanted suggestion at the right time. Some participants expressed negative opinion towards it later in the interview, because of its intrusive property.

The interview results largely confirmed the results from the qualitative data, that it is not advantages to use Python type hints in competitive programming.

6 Future Work

We have to discuss what is actually worth studying before we dive deeper into the field of CP. In lower levels, people benefit from learning some basic data structure and algorithms. Higher level CP might just be a niche sport that might not be so beneficial to optimize for. It might not be completely worth it to study the tools and programming languages just to improve the efficiency by a bit. We have to evaluate the benefit of doing such studies. As discussed in limitation, we might need to perform long-term experiements to fully determine the affect of programming langauge or tools on CP so that the participants have enough experience with the tools they are assigned to. Long-term experiements costs more and should be weighted more carefully.

To continue studying about tools and programming languages for CP. Firstly, one could improve this experiment and by considering the limitation described in the previous section, find ways to reduce the threats, and apply it to other scenarios.

As we observed before, the ratio of autocomplete suggestions being accepted is very low. One might carry out additional studies about the tool to investigate what kinds of suggestions have a positive impact on competitive programmer's efficiency.

Although we studied type hints in Python, it is not clear at all if people should use Python in CP. Additional study on topic of "programming language choice for CP" can be done before we dig deeper into Python.

7 Conclusion

Summarize what you learned.

In this paper, we designed an experiment to verify the research question

- Is Python Type Hint helpful in competitive programming?

We pilotted our experiment with four participants, and found that

- Python Type Hint might not be useful in competitive programming.
- Programmer dislike type hints when doing competitive programming tasks.

We also found some potential issues of the VSCode editor and Pylance extension:

- Autocomplete suggestions are not accurate and often ignored.
- Autocomplete suggestions is sometimes counter-productive.

Lastly, we identified the limitations of our work and suggested some directions of future work about tools and programming langauge use in competitive programming.

References

[1] A. Rigo and S. Pedroni, "Pypy's approach to virtual machine construction," in *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, 2006, pp. 944–953.

[2] *Static Typing with Python — typing documentation*, https://typing.readthedocs.io/en/latest/, [Online; accessed 2022-12-01], 2022.

[3] M. MIRZAYANOV, O. PAVLOVA, P. MAVRIN, *et al.*, "Codeforces as an educational platform for learning programming in digitalization," *Olympiads in Informatics*, vol. 14, pp. 133–142, 2020.

[4] *The world's leading online programming learning platform*. [Online]. Available: http://www.leetcode.com/.

[5] T. Di Mascio, L. Laura, and M. Temperini, "A framework for personalized competitive programming training," in *2018 17th International Conference on Information Technology Based Higher Education and Training (ITHET)*, 2018, pp. 1–8. DOI: 10.1109/ITHET.2018.8424620.

[6] W. Di Luigi, P. Fantozzi, L. Laura, *et al.*, "Learning analytics in competitive programming training systems," in *2018 22nd International Conference Information Visualisation (IV)*, 2018, pp. 321–325. DOI: 10.1109/iV.2018.00061.