UDK 004.424
IRSTI 14.25.09
DOI 10.56525/VBVU5385

FEATURES OF USING PYTHON INPUT METHODS IN COMPETITIVE PROGRAMMING

E.A. Abdykerimova*[1], Z.A. Pirnepessov[2]
[1]Yesenov University, Aktau, Kazakhstan
[2]Nazarbayev Intellectual Schools, Aktau, Kazakhstan
e-mail: elmira.abdykerimova@yu.edu.kz; yenispiro@gmail.com
*Корреспондент автор: elmira.abdykerimova@yu.edu.kz

Abstract. In competitive programming, efficiently handling input is a crucial skill, particularly in Python, where input operations can impact execution time and overall performance. This paper explores essential input methods and conversion functions in Python, highlighting their significance in competitive programming contexts. Beginning with the input() function, the paper demonstrates common techniques for reading and managing data types such as integers, floats, strings, and Booleans. It explains why data type conversion is often necessary, especially for performing arithmetic operations on user inputs. Additionally, this study covers the use of map() with input().split() to process multiple values on a single line efficiently, followed by insights into list handling and the list() function for organizing and accessing data. Practical examples are provided to illustrate the benefits of each method. By mastering these techniques, competitive programmers can enhance code efficiency, readability, and adaptability to time constraints commonly found in programming competitions. Understanding the nuances of Python's data handling can help programmers avoid common pitfalls that lead to slower code execution or runtime errors, particularly in time-sensitive contexts. Furthermore, the paper emphasizes how input processing strategies can reduce complexity, allowing programmers to focus on solving core algorithmic challenges rather than managing cumbersome input operations. The methods discussed are adaptable across various programming tasks, from simple exercises to more advanced problem sets. This paper is a valuable resource for programmers aiming to refine their input-handling skills and maximize Python's potential in competitive settings.

Keywords: competitive programming, Python input methods, input() function, data type conversion, map() function, input().split(), list handling, code efficiency.

Introduction

Competitive programming has significantly evolved since its beginnings, which can be traced back to algorithmic challenges in academic environments during the 1970s and 1980s. The field gained broader popularity in the 1990s with international contests like the ACM International Collegiate Programming Contest (ICPC) and the rise of online platforms such as Codeforces, AtCoder, and LeetCode. These platforms not only made competitive programming accessible to a global audience but also established it as a recognized skill set, particularly valuable in technology and engineering roles [1, 2].

As competitive programming grew, so did the variety of programming languages used, with Python quickly becoming one of the most popular choices due to its readability and versatility. Python's simplicity makes it ideal for beginners, while its rich standard library and flexibility provide the depth needed for more complex challenges. This paper explores Python's unique advantages in efficiently handling inputs, which is essential for competitive programming.

To support learning in competitive programming, several resources are particularly beneficial. "Competitive Programming" by Steven Halim and Felix Halim, along with "Programming Challenges" by Steven Skiena and Miguel Revilla, provide foundational insights into key algorithmic techniques and problem-solving approaches relevant to contests. For those specifically focused on

Python, "Python for Data Analysis" by Wes McKinney and "Fluent Python" by Luciano Ramalho explore Python's strengths, covering essential syntax, libraries, and performance considerations. Together, these resources create a well-rounded foundation for both competitive programming and proficiency in Python.

The benefits of understanding the content of this paper are multifaceted for learners. By mastering Python's input methods and data handling techniques, participants in competitive programming can write cleaner and more efficient code, which reduces execution time—a crucial factor in competitions. Moreover, these skills are transferable to real-world applications where efficiency and readability are highly valued. By mastering input methods, learners can sharpen their problem-solving abilities and enhance their adaptability, an essential trait in both academic and professional programming environments. Consequently, this paper serves as a practical guide to one of Python's core competencies within competitive programming, empowering learners to optimize their code and perform with greater confidence in contests [3,4].

Research methods and materials

Input methods in competitive programming using Python

In programming olympiads, after finding a solution in a paper, the important thing is to find a way to get input. In competitive programming, the ability to handle inputs quickly and efficiently can often make the difference between a successful solution and a time-out error. Python is a popular choice among competitive programmers due to its readability and simplicity. However, Python's simplicity doesn't always equate to speed, and input handling is a crucial area where Python programmers need to be strategic to maximize efficiency. In this paper, we'll explore various input methods in Python, focusing on common functions, data types, and structures essential for competitive programming.

The input() Function

One of the most fundamental input methods in Python is the input() function. This function reads a line of input from the standard input (usually the keyboard) and returns it as a string. For example:

data = input()

In this example, data will hold the string entered by the user. If the user types "5", then data will store "5" as a string. This is essential because most competitive programming platforms provide input data as strings, so this method serves as the basis for almost all input handling in Python.

The input() function alone is sufficient for simple programs or single-line inputs. However, in competitive programming, inputs are often multi-line, or they need to be converted to other data types (like integers or floats) for processing. Therefore, input() is often paired with other functions to handle a variety of scenarios.

Data Types and Conversion Functions

One of the first things competitive programmers need to learn about input() is that it always returns a string, regardless of what is entered. This can be limiting, especially when working with numerical data. For instance, if a user enters the number "42" and you store it using data = input(), data will contain the string "42", not the integer 42. If you attempt to add data + 1, Python will raise a TypeError because it cannot add a string to an integer.

To overcome this, we need to convert input data into the appropriate type before performing operations on it. Python provides several built-in functions for type conversion:

int(): Converts a string or float to an integer.

float(): Converts a string or integer to a floating-point number.

str(): Converts a number or other data type to a string.

bool(): Converts a value to a Boolean (True or False) [5,6].

Each of these functions plays an essential role in competitive programming. Let's explore them in more detail:

int(): The int() function is commonly used to convert a string to an integer. For example:

number = int(input())

If the user inputs "5", then number will hold the integer 5, allowing arithmetic operations. Integer conversion is often necessary because most mathematical operations, loops, and array indices require integers.

float(): In some competitive programming problems, data may require floating-point precision, especially in cases where measurements or real numbers are involved. For instance:

value = float(input())

If the input is "5.67", then value will be 5.67 as a float, allowing precise calculations. Floating-point numbers are necessary for handling non-integer values, especially when dealing with divisions, percentages, and scientific calculations.

str(): While str() is not as commonly used in competitive programming (since input is already a string), it can be useful for converting non-string data types into strings for output or concatenation. For example:

result = str(42) + " is the answer."

This can be helpful when preparing formatted output or concatenating numbers with strings.

bool(): The bool() function is less commonly used for direct input, but it can convert values to True or False. For instance:

is_valid = bool(int(input()))

Here, if the user enters 1, is_valid will be True. If the user enters 0, it will be False. This can be useful for binary flags or toggling states.

In competitive programming, it is crucial to effectively utilize type conversion functions to guarantee that the data is appropriately suited for the operations at hand. When programmers properly convert data types, they minimize the risk of encountering errors or producing unexpected results during their calculations. This attention to detail is not just a matter of accuracy, but it can also be a deciding factor in their success; mismanaging data types may lead to significant errors that could potentially result in disqualification from competitions. Therefore, having a thorough understanding of how to handle data types is essential for any serious competitor in the field.

Example Exercises: Basic Input and Conversion

Consider a problem that requires reading an integer, performing a calculation, and printing the result. The problem could look like this:

Problem Statement: Read an integer N from input, calculate N * 2, and output the result.

Solution:

```
N = int(input())
result = N * 2
print(result)
```

In this example, the function `int(input())` is utilized to take user input and convert it from a string format into an integer format. This conversion is crucial because it allows mathematical operations, such as multiplication, to be performed correctly. If we were to skip the conversion and simply use the input directly, the variable ‘N’ would be treated as a string. Consequently, if we attempted an operation like ‘N * 2’, instead of obtaining a numerical result, we would end up with the string being repeated. For instance, if the user enters "5" and we multiply it by 2 without converting it to an integer, the operation would produce "55" rather than the expected numeric result of 10. Therefore, ensuring that the input is converted into an integer is essential for performing accurate arithmetic calculations.

Using map() with input().split() for Multiple Inputs

In competitive programming, it's common to read multiple values on a single line, especially when working with arrays or matrices. To handle this, programmers often use input().split() combined with map().

The input().split() method reads a line of input and splits it into a list of strings based on whitespace. For instance:

data = input().split()

If the user enters "5 10 15", data will be ["5", "10", "15"], a list of strings. To convert these into integers, we use map():

numbers = list(map(int, input().split()))

In this particular scenario, the expression `map(int, input().split())` is utilized to transform the input from the user into a list of integers. Here's how it works in detail: 1. The `input()` function captures a line of user input, which is typically a string containing several numbers separated by spaces; 2. The `split()` method is then called on this string, which divides it into individual substrings (or tokens), each corresponding to one number in the original input. This results in a list of strings; 3. The `map()` function takes two arguments: the first is the `int` function, which converts a string representation of a number into an actual integer, and the second is the list of strings produced by the `split()` method. The `map()` function applies the `int` function to each element of the list, effectively converting each string into its integer equivalent; 4. Finally, wrapping the entire `map()` function call with `list()` creates a new list that contains all the converted integers.

This approach is not only highly efficient but also concise. It minimizes the number of lines of code needed to achieve the conversion and enhances readability by clearly outlining the process of transforming input into usable integer values. Overall, it demonstrates a clean and Pythonic way to handle user input for numerical data.

Why map() and split() Are Essential in Competitive Programming

The combination of `map()` and `input().split()` is crucial in competitive programming for a variety of reasons that can significantly enhance the coding experience and efficiency:

Efficiency: One of the standout features of using `map()` in conjunction with `input().split()` is its ability to eliminate the need for repetitive iteration through data for type conversion. In a typical scenario, a programmer might read input as a string and then have to loop through each element to convert it to the desired data type, such as integers or floats. By directly applying `map()`, programmers can transform the data types of all elements in a single, seamless operation. This not only reduces the amount of code but also enhances performance, which is crucial in competitive settings.

Readability: The syntax combining `map()` and `input().split()` results in a more concise code structure. This brevity aids in improving readability, which is particularly valuable during competitions where time is of the essence. When reviewing code under stress, the clarity provided by this combination allows programmers to quickly recall their thought processes, understand the flow of data within the application, and locate potential errors more easily.

Performance**: Competitive programming often features stringent time constraints where seconds can make a significant difference in the outcome. By utilizing `map()` and `input().split()`, programmers can minimize the execution time associated with processing input. Since this approach reduces the need for line-by-line execution and nested loops, it allows the program to handle larger inputs or more complex data structures efficiently. Thus, every millisecond saved can be critical in gaining an edge over competitors [7, 8].

The effective use of `map ()` and `input ().split ()` not only streamlines the coding process but also plays a vital role in enhancing the overall performance and readability of the program, making it a nearly indispensable technique in competitive programming. Consider an example problem requiring multiple inputs on a single line:

Problem Statement: Read three integers a, b, and c from a single line of input and print their sum.

Solution:

a, b, c = map(int, input().split())

print(a + b + c)

Here, map(int, input().split()) converts each input to an integer, enabling arithmetic operations directly.

Lists and the list() Function

Lists are an essential data structure in competitive programming, allowing programmers to store multiple values in a single variable. The list() function is often used to create lists from map() results or to convert data types into lists. For example:

data = list(map(int, input().split()))

In this case, data is a list of integers, providing easy access to individual elements and allowing for efficient storage of multiple values. Lists are especially useful in problems involving sequences, arrays, or matrices.

Consider a more complex problem where a list is necessary:

Problem Statement: Read a series of integers on a single line and output their sum.

Solution:

numbers = list(map(int, input().split()))

print(sum(numbers))

Here, numbers is a list of integers created from input, and sum(numbers) efficiently computes their total.

Results and discussion

Problem 1: Two friends, Pete and Billy, want to split a watermelon weighing n kilos. They want to divide it into two parts, each with an even weight greater than zero. Help them determine if this is possible [9].

Input: A single integer n ($1 \leq n \leq 100$), representing the weight of the watermelon.

Output: Print "YES" if they can split the watermelon into two even parts with positive weights, otherwise print "NO".

Table 1. Input and expected output of the problem 1.

input
8
output
YES

Solution: (In the solution, each line is numbered to enhance clarity and allow readers to follow the Python code's logic step-by-step).

```
1    n = int(input())
2    if n % 2 == 0  and n > 2:
3      print('YES')
4    else:
5      print('NO')
```

In the solution, the code begins with the use of int(input()), a critical component for reading and converting user input into an integer format. This step is essential, as it ensures that the input is interpreted as an integer, allowing for precise mathematical operations and comparisons necessary for solving the problem. By converting the raw input into a usable integer type, the solution can handle the computational logic effectively, providing a reliable foundation for subsequent steps in the algorithm.

Problem 2: Little Chris has a toy box with n columns of cubes, where the *i*-th column has a_i cubes.

Typically, gravity exerts a downward force on the cubes, causing them to rest at the bottom of the box. However, when Chris alters the gravitational pull, it shifts direction, now drawing the cubes to the right side instead. To illustrate this dramatic change in gravitational force, the cubes must be rearranged accordingly. Figure 1 provides a visual comparison of the initial and final arrangements of the cubes within the box, with the cubes that have shifted positions clearly highlighted in orange [10].

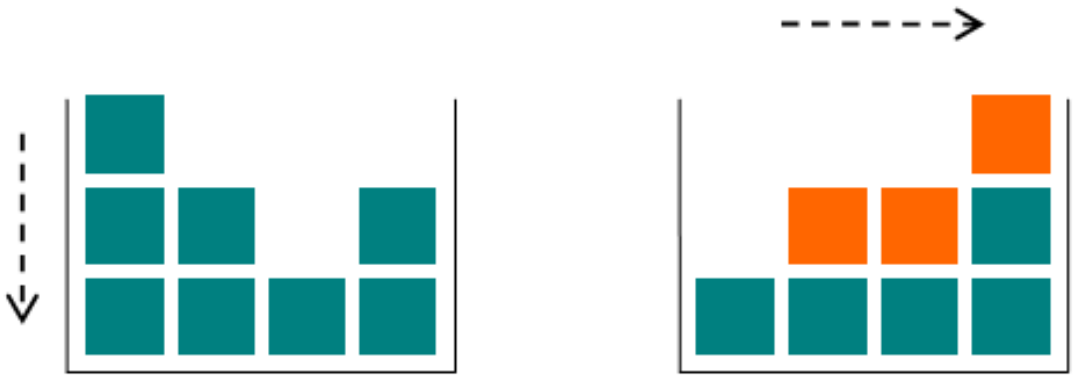


Figure 1. Change of cubes

Given the initial configuration of the toy cubes in the box, find the amounts of cubes in each of the n columns after the gravity switch!

Input: The first line of input contains an integer n ($1 \leq n \leq 100$), the number of the columns in the box. The next line contains n space-separated integer numbers. The i-th number a_i ($1 \leq a_i \leq 100$) denotes the number of cubes in the i-th column.

Output: Output n integer numbers separated by spaces, where the i-th number is the amount of cubes in the i-th column after the gravity switch.

Table 2. Input and expected output of the problem 2.

input
4 3 2 1 2
output
1 2 2 3

input
3 2 3 8
output
2 3 8

Solution: **(**In the solution, each line is numbered to enhance clarity and allow readers to follow the Python code's logic step-by-step)

```
1    n = int(input())
2    a = list(map(int, input().split()))
3    a.sort()
4    print(*a)
```

In this solution, the lines n = int(input()) and a = list(map(int, input().split())) are essential for setting up the input correctly. The first line reads an integer (n), representing the number of columns, ensuring that we understand the size of the input array. The second line reads (n) integers in one line, converting each value into an integer and storing them in a list (a). This use of map() and split() allows us to quickly parse multiple values from a single input line, which is efficient and suitable for competitive programming where concise input handling is crucial. Together, these lines provide a solid foundation for implementing the solution logic.

Conclusion

In summary, the ability to handle input efficiently is of paramount importance for success in competitive programming using Python. Aspiring programmers should focus on mastering the `input()` function, along with key type conversion functions such as `int()` and `float()`, which transform user inputs into the desired data types. For more advanced scenarios, leveraging techniques like `map()` combined with `input().split()` allows for the seamless processing of multiple values in a single line, streamlining the input handling process. Lists, which can be created effortlessly using the `list()` function, serve as fundamental data structures for organizing and accessing the inputs collected during a program's execution. Together, these tools establish a comprehensive framework that equips programmers to tackle various competitive programming challenges with both accuracy and efficiency. By honing their skills in effective input handling, programmers can significantly reduce unnecessary complexity in their code. This results in solutions that are not only more efficient but also easier to read and understand, particularly in high-pressure competitive environments where time is of the essence. Furthermore, adopting these best practices significantly decreases the likelihood of runtime errors, especially those that stem from type mismatches or invalid data formats-issues that frequently arise in the fast-paced context of programming contests. By implementing these input handling strategies, programmers can shift their focus from troubleshooting potential input issues to dedicating more time and energy to designing innovative algorithms. These foundational skills are not only beneficial for Python, but they are also versatile and can be applied across different programming languages and platforms. Ultimately, mastering these input handling techniques is crucial for gaining a competitive advantage in coding competitions, empowering programmers to perform at their best [11, 12].

REFERENCES

1. Уэс Маккинни. Python и анализ данных. Второе издание. - ДМК Пресс, 2019. – 540 с.

2. Чернышев С.А. Алгоритмы и структуры данных на Python: учебное пособие. – Санкт-Петербург: «КноРус», 2024. – 328 с.

3. Steven Halim, Felix Halim. Competitive Programming. 2020. - 600 p.

4. Steven Skiena, Miguel Revilla. Programming challenges: The programming contest training manual. 2005. – 416 p.

5. Luciano Ramalho. Fluent Python, 2nd edition. O'Reilly, 2022. – 898 p.

6. Cormen Thomas, Leiserson Charles, Rivest Ronald, Clifford Stein Introduction to algorithms (Введение в алгоритмы). – Publisher: MIT Press, 1990. – 1312 p.

7. Laaksonen Antti Competitive Programmer’s Handbook. - University of Helsinki, 2018. – 296 p.

8. Sedgewick Robert, Wayne Kevin Algorithms, 4th Edition. - Princeton University Press, 2011

9. Heintz F., Lange D. Evaluating Code Performance in Programming Contests // IEEE Frontiers in Education Conference, 2012.

10. Verma K., Sinha N. Optimized Algorithms for Real-Time Input Handling in Programming Competitions // International Journal of Computer Science and Network Security, 2020.

11. Khan, M.M., Yu, Z. Approaching code search for python as a translation retrieval problem with dual encoders. Empir Software Eng 30, 12, 2024.

12. Ozan Coskunserce, Comparing the use of block-based and robot programming in introductory programming education: Effects on perceptions of programming self-efficacy, 2023.

REFERENCES

1. Ues Makkinni. Python i analiz dannykh. Vtoroye izdaniye. - DMK Press, 2019. – 540 s. [in Russian]

2. Chernyshev S.A. Algoritmy i struktury dannykh na Python: uchebnoye posobiye. – Sankt-Peterburg: «KnoRus», 2024. – 328 s. [in Russian]

3. Steven Halim, Felix Halim. Competitive Programming. 2020. - 600 p.

4. Steven Skiena, Miguel Revilla. Programming challenges: The programming contest training manual. 2005. – 416 p.

5. Luciano Ramalho. Fluent Python, 2nd edition. O'Reilly, 2022. – 898 p.

6. Cormen Thomas, Leiserson Charles, Rivest Ronald, Clifford Stein Introduction to algorithms (Введение в алгоритмы). – Publisher: MIT Press, 1990. – 1312 p.

7. Laaksonen Antti Competitive Programmer's Handbook. - University of Helsinki, 2018. – 296 p.

8. Sedgewick Robert, Wayne Kevin Algorithms, 4th Edition. - Princeton University Press, 2011

9. Heintz F., Lange D. Evaluating Code Performance in Programming Contests // IEEE Frontiers in Education Conference, 2012.

10. Verma K., Sinha N. Optimized Algorithms for Real-Time Input Handling in Programming Competitions // International Journal of Computer Science and Network Security, 2020.

11. Khan, M.M., Yu, Z. Approaching code search for python as a translation retrieval problem with dual encoders. Empir Software Eng 30, 12, 2024.

12. Ozan Coskunserce, Comparing the use of block-based and robot programming in introductory programming education: Effects on perceptions of programming self-efficacy, 2023.

PYTHON ТІЛІНДЕГІ ЕНГІЗУ ӘДІСТЕРІН БӘСЕКЕЛЕСТІК БАҒДАРЛАМАЛАУДА ҚОЛДАНУ ЕРЕКШЕЛІКТЕРІ

Э.А. Абдыкеримова*[1], Ж.А. Пирнепесов[2]
[1]Есенов университеті, Ақтау, Қазақстан
[2]Назарбаев зияткерлік мектебі, Ақтау, Қазақстан
e-mail: elmira.abdykerimova@yu.edu.kz; yenispiro@gmail.com
*Корреспондент автор: elmira.abdykerimova@yu.edu.kz

Аңдатпа. Бәсекелестік бағдарламалауда енгізу деректерін тиімді өңдеу - бұл маңызды дағды, әсіресе Python тілінде, мұнда енгізу операцияларының орындалу уақыты мен жалпы өнімділікке ықпалы бар. Бұл мақала Python тіліндегі негізгі енгізу әдістері мен түрлендіру функцияларын қарастырып, олардың бәсекелестік бағдарламалаудағы маңыздылығын көрсетеді. Мақала input() функциясынан бастап, енгізу деректерін жол, бүтін сан, нақты сан және логикалық типтерде оқу және түрлендірудің қажеттілігін, сонымен қатар пайдаланушы енгізулеріне арифметикалық операциялар орындауда деректер типін түрлендірудің маңыздылығын түсіндіреді. Сонымен қатар, бір жолда бірнеше мәндерді тиімді өңдеу үшін map() және input().split() әдістерін қолдану, сондай-ақ деректерді ұйымдастыру және қолжетімді ету үшін тізімдер мен list() функциясын пайдалану қарастырылады. Әрбір әдістің артықшылықтарын көрсету үшін практикалық мысалдар берілген. Осы техникаларды игеру арқылы бәсекелестік бағдарламашылар бағдарламалық кодтың тиімділігін, оқылымдылығын және уақыт шектеулеріне бейімделуін жақсарта алады. Python тілінде деректерді өңдеу ерекшеліктерін меңгеру бағдарламашыларға және оқушыларға кодты орындаудың баяулауына немесе орындау уақытындағы қателерге, әсіресе уақытты сезіну контекстінде жиі кездесетін қателіктерден аулақ болуға көмектеседі. Сонымен қатар, мақалада енгізуді өңдеу стратегияларының күрделілікті қалай азайтуға болатынына баса назар аударылады, бұл бағдарламашыларға күрделі енгізу операцияларын басқаруға емес, негізгі алгоритмдік мәселелерді шешуге назар аударуға мүмкіндік береді. Талқыланатын әдістер қарапайым жаттығулардан жетілдірілген есептер жинақтарына дейін әртүрлі бағдарламалау тапсырмаларына бейімделеді. Бұл мақала енгізуді өңдеу дағдыларын жетілдіруге және

бәсекеге қабілетті жағдайларда Python әлеуетін арттыруға бағытталған бағдарламашылар үшін құнды ресурс болып табылады.

Түйін сөздер: бәсекелестік бағдарламалау, Python тілі, енгізу әдістері, input() функциясы, деректер типін түрлендіру, map() функциясы, input().split(), тізімдерді өңдеу, код тиімділігі.

ОСОБЕННОСТИ ПРИМЕНЕНИЯ МЕТОДОВ ВВОДА НА ЯЗЫКЕ PYTHON В СОРЕВНОВАТЕЛЬНОМ ПРОГРАММИРОВАНИИ

Э.А. Абдыкеримова*[1], Ж.А. Пирнепесов[2]
[1]Университет Есенова, Актау, Казахстан
[2]Назарбаев Интеллектуальная школа, Актау, Казахстан
e-mail: elmira.abdykerimova@yu.edu.kz; yenispiro@gmail.com
*Correspondent author: elmira.abdykerimova@yu.edu.kz

Аннотация. Соревновательное программирование требует эффективной обработки входных данных - это важный навык, особенно на языке Python, где время выполнения операций ввода и их влияние на общую производительность имеют большое значение. Эта статья рассматривает основные методы ввода и функции преобразования в Python, демонстрируя их важность в соревновательном программировании. В статье рассматривается использование функции input(), необходимость чтения и преобразования входных данных в строки, целые числа, вещественные числа и логические типы, а также важность преобразования типов данных при выполнении арифметических операций с пользовательскими вводами. Кроме того, рассматриваются методы map() и input().split() для эффективной обработки нескольких значений в одной строке, а также использование списков и функции list() для организации и доступности данных. Для демонстрации преимуществ каждого метода приводятся практические примеры. Освоив эти техники, конкуренты-программисты могут улучшить эффективность, читаемость кода и его адаптацию к ограничениям по времени. Освоение особенностей обработки данных в Python помогает программистам и учащимся избежать замедления выполнения кода или ошибок во время выполнения, особенно часто встречающихся в контексте чувствительности к времени. В статье также акцентируется внимание на том, как стратегии обработки ввода могут снизить сложность, позволяя программистам сосредоточиться не на управлении сложными операциями ввода, а на решении основных алгоритмических задач. Рассматриваемые методы адаптированы для различных программных задач, от простых упражнений до сложных наборов задач. Эта статья является ценным ресурсом для программистов, направленным на улучшение навыков обработки ввода и повышение потенциала Python в конкурентных условиях.

Ключевые слова: соревновательное программирование, язык Python, методы ввода, функция input(), преобразование типов данных, функция map(), input().split(), обработка списков, эффективность кода.