**Prompt 1: Llama3.1**

You are a skilled software developer proficient in multiple programming languages. Your task is to re-write the input source code.

Below is the input source code written in C that you should re-write into Python

programming language. You must respond with the Python output code only.

#include <stdio.h>

int main(void){

int i,a,b,c,d,e;

while(scanf("%d", &a)!= EOF) {

b=600/a;

int sum=0;

for(i=1;i<b;i++){

e=i\*a;

d=e\*e;

c=a\*d;

sum+=c;

}

printf("%d\n",sum);

if(a == 0) {

break;

}

}

return 0;

}

You were asked to translate the following  
C code to Python:

#include<stdio.h>

void swapc(int\*, int\*);

int main(){

int a,b,c,n,i;

scanf("%d",&n);

for(i=0;i<n;i++){

scanf("%d %d %d",&a,&b,&c);

swapc(&a,&c);

swapc(&b,&c);

if(a\*a+b\*b == c\*c){

printf("YES\n");

}

else {

printf("NO\n");

}

}

return 0;

}

void swapc(int \*x, int \*c){

int temp;

if(\*x>\*c){

temp = \*x;

\*x = \*c;

\*c = temp;

}

}

You are a skilled software developer proficient in multiple programming languages. Your task is to re-write the input source code.

Below is the input source code written in C that you should re-write into Python. The Python program should:

1. **Read input** from stdin (e.g., through redirection from a file or a testing framework).

2. **Write output** to stdout in the same format as the original C code, without additional prompts or interactivity.

3. Be compatible with **batch processing** or **automated testing tools**, such as subprocess.run.

4. Include appropriate error handling for cases like division by zero or invalid input, ensuring the program does not crash.

**Prompt 2: Llama3.1**

You are a skilled software developer proficient in multiple programming languages. Your task is to re-write the input source code.

Below is the input source code written in C that you should re-write into Python. The Python program must:

1. Read input from stdin, allowing for file redirection (e.g., < input.txt) or integration with automated testing tools like subprocess.run.

2. Write output to stdout in the same format as the original C program. Do not include additional prompts or interactive messages.

3. Be compatible with batch processing, capable of handling multiple test cases if applicable.

4. Include error handling where necessary (e.g., division by zero, invalid input) to ensure the program does not crash unexpectedly.

Here is the input source code in C:

#include <stdio.h>

#include <math.h>

int main(void){

double a,b,c,d,e,f,aa,bb,cc,x,y,bai;

while(scanf("%lf %lf %lf %lf %lf %lf",&a,&b,&c,&d,&e,&f)!=EOF){

bai=d/a;

aa=a\*bai;

bb=b\*bai;

cc=c\*bai;

y=(f-cc)/(e-bb);

x=(c-b\*y)/a;

printf("%.3f %.3f\n",x,y);

}

}

Respond with the Python output code only. Do not include explanations, comments, or additional formatting in your response.

**Prompt 3: Llama3.1**

You are a skilled software developer proficient in multiple programming languages. Your task is to re-write the following C code into Python. The Python program must:

1. Translate all functions and logic from the C code into Python, ensuring they behave equivalently and maintain the same functionality.

2. Preserve the logical structure and the behavior of the original code, including mathematical operations, loops, conditions, and function calls.

3. Handle input from sys.stdin, making it suitable for file redirection (e.g., < input.txt) or automated testing tools like subprocess.run. Avoid interactive prompts like input().

4. Write output to sys.stdout, matching the format of the original C code exactly. Do not add additional messages or unnecessary text.

5. Be compatible with batch processing, capable of handling multiple test cases if applicable, as described in the original C code.

6. Include error handling where necessary (e.g., division by zero, invalid input, or other potential runtime errors) to ensure the program executes robustly.

7. Add all necessary Python imports at the beginning of the program, such as import sys or any other library required for equivalent functionality in Python.

8. Follow Python conventions, making the code idiomatic while ensuring it is logically equivalent to the C code.

Here is the input source code in C:

#include <stdio.h>

int main()

{

int a[1000000]={0};

int i, j, m, ans;

while(scanf("%d", &m)!=EOF) {

for(i=0;i<=m;i++)

a[i]=0;

ans=0;

for(i=2;i<=m;i++) {

if (a[i]==0) {

ans++;

for(j=1;i\*j<=m;j++) {

a[i\*j]=1;

}

}

}

printf("%d\n", ans);

}

return (0);

}

Respond with the Python output code only. Do not include explanations, comments, or additional formatting in your response.

**Prompt 4: Llama3.1**

You are a skilled software developer proficient in multiple programming languages. Your task is to accurately rewrite the following C code into Python while preserving all functionality. The Python program must meet the following criteria:

1. Functionality Preservation

• Translate all functions, loops, conditions, and logic from C to Python exactly so that the behavior remains identical.

• Ensure mathematical operations, data structures, and algorithm logic remain unchanged.

2. Logical Accuracy

• Ensure that the counting logic, iterations, and assignments remain true to the original C code.

• If the C code counts specific values (e.g., primes, occurrences), do not change the counting method.

• Verify whether summing an array or maintaining a counter is logically correct before translating.

3. Input Handling

• Read input from sys.stdin, making it batch-processing friendly (e.g., < input.txt> or subprocess.run() for automation).

• Avoid input(), unless explicitly needed.

4. Output Handling

• Match the original C output format exactly. Do not add unnecessary print messages, extra spaces, or formatting.

5. Memory Management & Data Structures

• If the C code initializes large arrays statically, convert them dynamically in Python.

• Convert fixed-size arrays into Python lists or other appropriate data structures while keeping performance in mind.

6. Error Handling & Edge Cases

• Preserve boundary conditions, including edge cases like zero, negative values, and large inputs.

• Handle division by zero, out-of-bounds errors, and invalid input where necessary.

7. Efficient & Idiomatic Python Code

• Use Python conventions where possible but do not change the original algorithm.

• Avoid unnecessary recursion if it might cause stack overflow, replacing it with loops when needed.

8. Correct Import Usage

• Add necessary Python imports (import sys, import math, etc.) based on the equivalent C functions.

Here is the input source code in C:

#include <stdio.h>

int main(){

int n;

int a,b,c;

a=b=c=0;

for(int i=0;i<10;i++){

scanf("%d",&n);

if(n>=a){

c=b;b=a;a=n;

}else if(n>=b){

c=b,b=n;

}else if(n>c){

c=n;

}

}

printf("%d\n%d\n%d\n",a,b,c);

return(0);

}

Respond with Python output code only. Do not include explanations, comments, or additional formatting in your response. The output should be a fully executable Python program.

**Prompt 5: Llama3.1**

You are a skilled software developer proficient in multiple programming languages. Your task is to accurately rewrite the following C code into Python while preserving all functionality. The Python program must meet the following criteria:

1. Functionality Preservation

• Translate all functions, loops, conditions, and logic from C to Python exactly so that the behavior remains identical.

• Ensure mathematical operations, data structures, and algorithm logic remain unchanged.

2. Logical Accuracy

• Ensure that the counting logic, iterations, and assignments remain true to the original C code.

• If the C code counts specific values (e.g., primes, occurrences), do not change the counting method.

• Verify whether summing an array or maintaining a counter is logically correct before translating.

• Do not impose uniqueness constraints unless explicitly enforced in the original C code. If the C code allows duplicate values in rankings, comparisons, or assignments, the Python translation must also allow duplicates.

• Preserve Variable Order Exactly: Maintain the exact order of variables in function parameters, assignments, and input handling. Ensure input parsing follows the same order as C’s scanf(), preventing swaps or misalignment.

3. Input Handling

• Read input from sys.stdin, making it batch-processing friendly (e.g., < input.txt> or subprocess.run() for automation).

• Avoid input(), unless explicitly needed.

4. Output Handling

• Match the original C output format exactly. Do not add unnecessary print messages, extra spaces, or formatting.

5. Memory Management & Data Structures

• If the C code initializes large arrays statically, convert them dynamically in Python.

• Convert fixed-size arrays into Python lists or other appropriate data structures while keeping performance in mind.

6. Error Handling & Edge Cases

• Preserve boundary conditions, including edge cases like zero, negative values, and large inputs.

• Handle division by zero, out-of-bounds errors, and invalid input where necessary.

• If the C code allows repeated values in comparisons, sorting, or ranking logic, the Python translation must also allow duplicates without unnecessary filtering.

7. Efficient & Idiomatic Python Code

• Use Python conventions where possible but do not change the original algorithm.

• Avoid unnecessary recursion if it might cause stack overflow, replacing it with loops when needed.

8. Correct Import Usage

• Add necessary Python imports (import sys, import math, etc.) based on the equivalent C functions.

9. Strictly Maintain Sorting, Comparisons, and Assignments

• Ensure that all conditions (if, else if, else), comparisons (>=, >), and assignments are translated exactly as they are in the C code.

• Do not introduce uniqueness constraints or filters unless explicitly present in the C program. If the C code allows ranking elements with duplicate values, Python must maintain the same ranking order and count duplicates correctly.

Here is the input source code in C:

#include <stdio.h>

int main(void) {

double x[3], y[3], xp, yp, cp[3];

int i, j;

while (scanf("%lf%lf%lf%lf%lf%lf%lf%lf",

&x[0], &y[0], &x[1], &y[1], &x[2], &y[2], &xp, &yp ) == 8) {

for (i = 0; i < 3; i++) {

j = (i + 1) % 3;

cp[i] = (x[j] - x[i]) \* (yp - y[i]) - (y[j] - y[i]) \* (xp - x[i]);

}

if ((cp[0] > 0 && cp[1] > 0 && cp[2] > 0) ||

(cp[0] < 0 && cp[1] < 0 && cp[2] < 0)) {

puts("YES");

}

else {

puts("NO");

}

}

return 0;

}

Respond with Python output code only. Do not include explanations, comments, or additional formatting in your response. The output should be a fully executable Python program.

**LATEST PROMPT:**  
  
You are a skilled software developer proficient in multiple programming languages. Your task is to accurately rewrite the following C code into Python while preserving all functionality and logic exactly. The Python program must meet the following criteria:

1. Functionality Preservation

* Translate all functions, loops, conditions, and logic from C to Python exactly so that the behavior remains identical.
* Ensure mathematical operations, data structures, and algorithm logic remain unchanged.
* Do not modify the fundamental algorithm or introduce alternative methods unless explicitly required.

2. Logical Accuracy

* Preserve the original logic for counting, summing, and assignments.
* Maintain the same variable order in function parameters, assignments, and input handling.
* Avoid modifying the sequence of operations, ensuring Python executes each step in the exact same order as C.
* Do not impose uniqueness constraints or filters unless explicitly present in the C code. If C allows duplicates, Python must also.

3. Input Handling (Critical for Accurate Conversion)

* Ensure input reading follows the same structure as C’s scanf():
  + If scanf() reads all input values in a single line, replicate this behavior in Python.
  + If scanf() reads multiple lines, replicate the exact line-by-line reading process.
* Use sys.stdin.read() or sys.stdin.readline() instead of input() for batch processing.
* Preserve space-separated input reading, avoiding unnecessary use of splitlines().
* Ensure floating-point values are read correctly without truncation or type conversion errors.

4. Output Handling

* The Python output must match the C output exactly:
  + Do not add extra spaces, newlines, or unnecessary print messages.
  + Ensure conditional branches (if/else) preserve output logic precisely.

5. Memory Management & Data Structures

* Convert fixed-size arrays in C to dynamic lists in Python only if needed.
* Preserve array indexing logic, ensuring no off-by-one errors.

6. Error Handling & Edge Cases

* Preserve all C boundary conditions, including:
  + Negative values, zero values, and large inputs.
  + Division by zero, out-of-bounds errors, and incorrect input handling.
* Ensure invalid input is handled the same way as C:
  + If scanf() in C stops reading on invalid input, Python should do the same.
  + Avoid premature termination unless explicitly required.

7. Efficient & Idiomatic Python Code (without changing logic)

* Use Python conventions where possible but do not change the algorithm.
* Avoid unnecessary recursion if it might cause stack overflow; replace it with loops when needed.

8. Correct Import Usage

* Import only necessary modules (e.g., sys, math).
* Do not introduce unnecessary dependencies.

9. Strictly Maintain Sorting, Comparisons, and Assignments

* Ensure all conditions (if, else if, else), comparisons (>=, >, <), and assignments are translated exactly.
* Do not modify ranking logic, sorting behavior, or duplicate handling unless explicitly required.

10. Input Validation & Edge cases

* If scanf() in C checks for a specific number of inputs (e.g., == 8), Python must strictly enforce this check.
* The Python code must skip invalid input cases just like C and should not assume inputs will always be correct.

Here is the input source code in C:

*#include <stdio.h>*

*int main(){*

*char c;*

*while(scanf("%c",&c)!=EOF){*

*printf("%c",toupper(c));*

*}*

*return(0);*

*}*

Respond with Python output code only. Do not include explanations, comments, or additional formatting in your response. The output should be a fully executable Python program.

**Prompt 1:** Deepseek-r1

Translate the following code from C to Python:

*#include <stdio.h>*

*int main(){*

*char c;*

*while(scanf("%c",&c)!=EOF){*

*printf("%c",toupper(c));*

*}*

*return(0);*

*}*

Respond with Python output code only. Do not include explanations, comments, or additional formatting in your response. The output should be a fully executable Python program.

**Prompt 2:** Deepseek-r1

You are a skilled software developer proficient in multiple programming languages. Your task is to re-write the input source code.

Below is the input source code written in C that you should re-write into Python programming language.

Make sure to:

- Maintain the same logic as the C program.

- Properly handle \*\*comma-separated input values\*\* in Python using `split(',')` when reading integers.

- Respond with the Python output code only.

Here is the C program:

#include<stdio.h>

int main(void){

int a[100];

int w,n,i,j,b,c,d,e;

scanf("%d",&w);

for(int i=0; i<w;i++){

a[i]=i+1;

}

scanf("%d",&n);

for(int j = 0; j<n; j++){

scanf("%d,%d",&b,&c);

if(a[c-1] == c && a[b-1] == b){

a[c-1] = b;

a[b-1] = c;

}

else if(a[c-1] != c && a[b-1] == b){

a[b-1] = a[c-1];

a[c-1] = b;

}

else if(a[c-1] == c && a[b-1] != b){

a[c-1] = a[b-1];

a[b-1] = c;

}

else if(a[c-1] != c && a[b-1] != b){

d=a[b-1];

e=a[c-1];

a[b-1]=e;

a[c-1]=d;

}

}

for(int i = 0; i<w; i++){

printf("%d\n",a[i]);

}

return 0;

}

**Prompt 3:** Deepseek-r1

You are a skilled software developer proficient in multiple programming languages. Your task is to \*\*re-write\*\* the given C program in \*\*Python\*\* while maintaining the same functionality.

### \*\*Guidelines for Translation:\*\*

- Ensure \*\*logical equivalence\*\* between the C and Python code.

- Convert \*\*C-style input handling (`scanf`)\*\* to \*\*Pythonic input handling (`input().strip()`)\*\*.

- Handle \*\*comma-separated values (`%d,%d`)\*\* using `split(',')` where applicable.

- Maintain \*\*array indexing differences\*\* (C uses zero-based indexing but may treat some indices as 1-based).

- Replace \*\*C-style loops (`for(int i=0; i<w; i++)`)\*\* with Python’s `for i in range(w):`.

- Convert \*\*printf-style output (`printf("%d\n", x)`)\*\* to Python's `print(x)`.

- Ensure memory management and variable initialization follow Python’s best practices.

- Do not include explanations—respond with \*\*Python code only\*\*.

### \*\*Here is the C program:\*\*

#include <stdio.h>

#include <string.h>

#define LEN 128

void convert(char charNum[], int intNum[]) {

int i, j = 0;

for (i = (int)strlen(charNum) - 1; i >= 0 ; i --) {

intNum[j] = (int)charNum[i] - 48;

j ++;

}

}

int main(int argc, const char \* argv[]) {

int dataSet;

int count = 1;

scanf("%d", &dataSet);

char firstNum[LEN], secondNum[LEN];

int fNum[LEN], sNum[LEN], sum[LEN];

int i, j, k, l;

while (count <= dataSet) {

count ++;

while (scanf("%s", firstNum) != EOF) {

break;

}

while (scanf("%s", secondNum) != EOF) {

break;

}

if (strlen(firstNum) > 80 || strlen(secondNum) > 80) {

printf("overflow\n");

continue;

}

for (i = 0; i < LEN; i ++) {

fNum[i] = 0;

sNum[i] = 0;

sum[i] = 0;

}

convert(firstNum, fNum);

convert(secondNum, sNum);

for (i = 0; i < LEN; i ++) {

sum[i] = fNum[i] + sNum[i];

}

for (i = 0; i < LEN; i ++) {

if (sum[i] >= 10) {

sum[i] %= 10;

sum[i + 1] += 1;

}

}

k = 0;

j = LEN - 1;

while (j >= 0) {

if (sum[j] == 0 && j != 0) {

k ++;

j --;

} else {

break;

}

}

if (k < LEN - 80) {

printf("overflow");

} else {

for (l = LEN - k - 1; l >= 0; l --) {

printf("%d", sum[l]);

}

}

printf("\n");

}

return 0;

}

**Additional Prompts:** Llama3.1

You are a highly skilled software developer proficient in both C and Python. Your task is to rewrite the following C code into Python while preserving all functionality, logic, and behavior exactly.

🔹 Key Translation Rules:

• Preserve Logic: Translate all functions, loops, conditions, and logic exactly as in C.

• Correct Input Handling:

• Replicate scanf() behavior precisely using sys.stdin.readline() or sys.stdin.read(), ensuring no unnecessary modifications.

• Maintain space-separated input handling and multi-line input as in C.

• Array Handling:

• Convert fixed-size C arrays (e.g., int arr[128]) to pre-allocated Python lists ([0] \* 128).

• Ensure list modification happens in place when needed (avoid returning new lists unless required).

• Loop & Indexing Accuracy:

• Preserve iteration logic (for, while) without off-by-one errors.

• Adapt zero-based indexing properly from C to Python where necessary.

• Output Handling:

• Match C’s printf() exactly—no extra spaces, lines, or formatting differences.

• Memory Management Adaptations:

• Replace C-style null-terminated strings with Python string handling.

• Ensure list boundaries are respected to prevent out-of-range errors.

• Do Not Introduce New Methods:

• Do not optimize or rewrite the algorithm unless explicitly instructed.

• Do not use Python built-in functions to replace fundamental C logic unless necessary.

Output only the translated Python code. Do not provide explanations, comments, or extra formatting.

Here is the C code:

#include <stdio.h>

#include <string.h>

#define MAX 81

int get\_line(char str[]);

int judge(char word[]);

void shift\_word(char word[]);

void shift\_line(int n, char line[]);

int main(void)

{

char str[MAX];

char line[MAX];

char \*word;

int shift\_num;

while(get\_line(str) != 0) {

strcpy(line, str);

word = strtok(str, " ");

if ((shift\_num = judge(word)) != -1) {

shift\_line(shift\_num, line);

} else {

while((word = strtok(NULL, " ")) != NULL) {

if ((shift\_num = judge(word)) != -1) {

shift\_line(shift\_num, line);

break;

}

}

}

printf("%s\n", line);

}

return 0;

}

int get\_line(char str[])

{

int i = 0;

char c;

while ((c = getchar()) != '\n' && c != EOF) {

str[i++] = c;

}

str[i] = '\0';

return i;

}

int judge(char word[])

{

int i;

for (i = 0; i < 26; i++) {

if (strcmp(word, "the") == 0 ||

strcmp(word, "this") == 0 || strcmp(word, "that") == 0) {

return i;

}

shift\_word(word);

}

return -1;

}

void shift\_word(char word[])

{

int i;

int len = strlen(word);

for (i = 0; i < len; i++) {

if ('a' <= word[i] && word[i] <= 'z') {

word[i] = (word[i] - 'a' + 1) % 26 + 'a';

}

}

if (word[len - 1] == '.') {

word[len - 1] = '\0';

}

}

void shift\_line(int n, char line[])

{

int i;

for (i = 0; i < (int)strlen(line); i++) {

if ('a' <= line[i] && line[i] <= 'z') {

line[i] = (line[i] - 'a' + n) % 26 + 'a';

}

}

}

**Additional Prompts:** Llama3.1

You are a skilled software developer proficient in multiple programming languages. Your task is to accurately rewrite the following {source\_language} code into {target\_language} while preserving all functionality. The {target\_language} program must meet the following criteria:

1️. Functionality Preservation

• Translate all logic exactly so that behavior remains identical.

• Preserve loop structures unless absolutely necessary to change.

• Ensure math operations, data structures, and algorithms remain identical.

2️. Correct Handling of Mutable vs Immutable Types

• If {target\_language} has immutable equivalents (e.g., strings, tuples), functions must return new values instead of modifying in place.

• If {target\_language} has mutable equivalents (lists, dicts, sets), modifications can be done in place.

• For arrays in C that are modified in place, use lists in Python only if needed.

3️. Strict Function Translation

• If a function modifies an argument in {source\_language}, it must return the modified value in {target\_language}.

• Do not assume direct modification works unless explicitly possible in {target\_language}.

• Preserve return values to ensure correct updates.

4️. Avoid Overcomplication

• Do not replace simple loops with iterators or generators unless absolutely required.

• Do not use list comprehensions where a simple loop exists in {source\_language}.

• Preserve step-by-step transformations from {source\_language} to {target\_language}.

5️. Input Handling Must Match {source\_language}

• Use sys.stdin.readline() for line-based input.

• Use sys.stdin.read() for batch input processing.

• Preserve input structure (e.g., space-separated input should not be split unnecessarily).

6️. Output Handling

• Ensure {target\_language} output matches {source\_language} output exactly (no extra spaces or newlines).

• Do not introduce print statements that do not exist in {source\_language}.

7️. Efficient & Idiomatic {target\_language} Code (Without Changing Logic)

• Use {target\_language} conventions only if needed, but do not rewrite the algorithm in a different way.

• Maintain all sorting, comparisons, and assignments exactly as in {source\_language}.

8️. Memory Management & Edge Cases

• If {source\_language} function stops reading input after a certain condition, {target\_language} must follow the same behavior.

• Handle negative values, boundary conditions, and invalid input exactly as in {source\_language}.

• Avoid premature termination unless required.

Here is the {source\_language} code that needs to be translated into {target\_language}:

{source\_code}

Respond with Python output code only. Do not include explanations, comments, or formatting. The output should be a fully executable {target\_language} program.

**Codellama**

Translate the following C code to Python while preserving its logic, functionality, and efficiency. Ensure that the translated code follows Pythonic conventions, such as using list comprehensions, built-in functions, and appropriate data structures where applicable. Maintain readability and avoid unnecessary complexity.

#include <stdio.h>

int main(void) {

   double x[3], y[3], xp, yp, cp[3];

   int i, j;

   while (scanf("%lf%lf%lf%lf%lf%lf%lf%lf",

                &x[0], &y[0], &x[1], &y[1], &x[2], &y[2], &xp, &yp ) == 8) {

       for (i = 0; i < 3; i++) {

           j = (i + 1) % 3;

           cp[i] = (x[j] - x[i]) \* (yp - y[i]) - (y[j] - y[i]) \* (xp - x[i]);

       }

       if ((cp[0] > 0 && cp[1] > 0 && cp[2] > 0) ||

           (cp[0] < 0 && cp[1] < 0 && cp[2] < 0)) {

           puts("YES");

       }

       else {

           puts("NO");

       }

   }

   return 0;

}

**Requirements:**

1. Ensure the translated Python code produces the same output as the original C code.

2. Replace C-specific constructs (e.g., printf, scanf, pointers) with their Python equivalents.

3. Use idiomatic Python features where appropriate (e.g., for loops instead of while, list comprehensions).

4. Avoid using third-party libraries unless explicitly required.

After translation, I will test the Python code with test cases to verify correctness. Please ensure the logic remains intact.

Only return the translated Python code with no explanation or comments.