**01. the Pyramid of King Djoser**

Write a JS program that calculates how much resources will be required for the construction of a pyramid. It is made out of **stone**, **marble**, **lapis lazuli** and **gold**. Your program will receive an integer that will be the **base** width and length of the pyramid and an **increment**, that is the height of each step. The bulk is made out of stone, while the **outer layer** is made out of marble. **Every fifth step’s** outer layer is made out of lapis lazuli **instead** of marble. The **final step** is made out of gold.

The pyramid is built with **1x1 blocks** with **height** equal to the given **increment**. The first step of the pyramid has **width** and **length** equal to the given **base** and every next step is **reduced by 2 blocks** (1 from each side). The height of every step equals the given **increment**. See the drawing for an example. White steps are covered in marble, blue steps are covered in lapis lazuli (**every fifth layer from the bottom**), and yellow steps are made **entirely** out of gold (**top-most step**).



Since the **outer layer** of each step is made of a decorative material, to calculate the required stone for one step, reduce the width and length by 2 blocks (one from each side), find it’s area and multiply it by the increment. The rest of the step is made out of lapis lazuli for every fifth step from the bottom and marble for all other steps. To find the amount needed, you may, for example, find its perimeter and reduce it by 4 (to compensate for the overlapping corners) and multiply the result by the increment. See the drawing for details (grey is stone, white is decoration).

|  |  |  |
| --- | --- | --- |
| Picture 8 | Picture 9 | Picture 10 |
| 5x5 step  Stone required – 9 x increment  Marble required – 16 x increment | 7x7 step  Stone required – 25 x increment  Marble required – 24 x increment | 8x8 step  Stone required – 36 x increment  Marble required – 28 x increment |

Note the top-most layer is made entirely out of gold, with height equal to the given increment. See the examples for complete calculations.

### Input

You will receive two **number** parameters **base** and **increment**.

### Output

Print on the **console** on separate lines the **total** required **amounts** of each material **rounded up** and the **final height** of the pyramid **rounded down**, as shown in the examples.

### Constraints

* The **base** will always be an integer greater than zero
* The **increment** will always be a number greater than zero
* **Number.MAX\_SAFE\_INTEGER** will **never be exceeded** for any of the calculations

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** | **Explanation** | | | | | |
| 11,  1 | Stone required: 165  Marble required: 112  Lapis Lazuli required: 8  Gold required: 1  Final pyramid height: 6 | Step | Size | Stone | Marble | Lapis | Gold |
| 1st | 11x11 | 81 | 40 | - | - |
| 2nd | 9x9 | 49 | 32 | - | - |
| 3rd | 7x7 | 25 | 24 | - | - |
| 4th | 5x5 | 9 | 16 | - | - |
| 5th | 3x3 | 1 | - | 8 | - |
| 6th | 1x1 | - | - | - | 1 |
| total | Height=6 | 165 | 112 | 8 | 1 |

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Explanation** |
| 11,  0.75 | Stone required: 124  Marble required: 84  Lapis Lazuli required: 6  Gold required: 1  Final pyramid height: 4 | Total **stone** is 81\*0.75+49\*0.75+25\*0.75+9\*0.75+1\*0.75 = 123.75, we round up to 124.  Total **marble** is 40\*0.75+32\*0.75+24\*0.75+16\*0.75=84.  Total **lapis lazuli** is 8\*0.75=6.  Total **gold** is 1\*0.75=0.75, we round up to 1.  Total **height** is 4.5 (6 **steps** times 0.75), we round down to 4. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 12,  1 | Stone required: 220  Marble required: 128  Lapis Lazuli required: 12  Gold required: 4  Final pyramid height: 6 | 23,  0.5 | Stone required: 886  Marble required: 228  Lapis Lazuli required: 36  Gold required: 1  Final pyramid height: 6 |

**02. Jan’s Notation**

Write a program that parses a series of instructions written in **postfix notation** and executes them (postfix means the operator is written **after** the operands). You will receive a **series of instructions** – if the instruction is a **number**, **save it**; otherwise, the instruction is an **arithmetic operator** (**+-\*/**) and you must apply it to the most two **most recently saved** numbers. **Discard** these two numbers and in their place, **save the result** of the operation – this number is now eligible to be an **operand** in a subsequent operation. Keep going until all input instructions have been exhausted, or you encounter an **error**.

In the end, if you’re left with a **single saved number**, this is the **result** of the calculation and you must **print** it. If there are more numbers saved, then the user supplied **too many instructions** and you must print "**Error: too many operands!**". If at any point during the calculation you **don’t have** two number saved, the user supplied **too few instructions** and you must print "**Error: not enough operands!**". *See the examples for more details.*

**Input**

You will receive an array with numbers **and** strings – the numbers will be **operands** and must be saved; the strings will be **arithmetic operators** that must be applied to the operands.

**Output**

Print on the **console** on a single line the **final result** of the calculation or an **error message**, as instructed above.

**Constraints**

* The **numbers** (operands) will be integers
* The **strings** (operators) will always be one of **+-\*/**
* The result of each operation will be in range [-253…253-1] (**MAX\_SAFE\_INTEGER** will **never** be exceeded)

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Explanation** |
| [3,  4,  '+'] | 7 | The first instruction is a **number**, therefor we **save** it. The next one is also a **number**, we **save** it too.  The third instruction is a **string**, so it must be an **operator** – we **remove the last two** numbers we saved, and perform the operation: **3+4=7**. The result of this operation is then **saved** where the two operands **used to be**.  We’ve ran out of instructions, so we check the saved values – we only have **one**, so this must be **final result**. We **print** it on the console. |
| [5,  3,  4,  '\*',  '-'] | -7 | We save in order **5**, **3** and **4**. The result of the operation **3\*4** is **12**, which we **save in place** of **3** and **4**.  Currently we have **5** and **12** saved. The result of the operation **5-12** is **-7**, which we **save in place** of **5** and **12**.  We have no more instructions and **only one** value saved, which we **print**. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| [7,  33,  8,  '-'] | Error: too many operands! | [15,  '/'] | Error: not enough operands! |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** | **Explanation** |  | **Input** | **Output** | **Explanation** |
| [31,  2,  '+',  11,  '/'] | 3 | **(31+2)/11** | [-1,  1,  '+',  101,  '\*',  18,  '+',  3,  '/'] | 6 | **(-1+1)\*101+18/3** |