Task 1

Intro

An extra day is added to the calendar almost every four years as February 29, and the day is called a leap day. It corrects the calendar for the fact that our planet takes approximately 365.25 days to orbit the sun. A leap year contains a leap day.

In the Gregorian calendar, three conditions are used to identify leap years:

- The year can be evenly divided by 4, is a leap year, unless:
- The year can be evenly divided by 100, it is NOT a leap year, unless:
- The year is also evenly divisible by 400. Then it is a leap year.

This means that in the Gregorian calendar, the years 2000 and 2400 are leap years, while 1800, 1900, 2100, 2200, 2300 and 2500 are NOT leap years

Task

Given a year, determine whether it is a leap year. If it is a leap year, return the Boolean True, otherwise return False.

It is only necessary to complete the `is_leap` function.

Input Format

year, the year to test.

Constraints

$1900 <= year <= 10^5$

Output Format

The function must return a Boolean value (True/False).

Example

```
is_leap(1990) == False # should be true
```

Explanation

1990 is not a multiple of 4 hence it's not a leap year.

Task 2

Intro

Consider a list (list = []). You can perform the following commands:

1. insert i e: Insert integer `e` at position `i`.
2. remove e: Delete the first occurrence of integer `e`.
3. append e: Insert integer `e` at the end of the list.
4. sort: Sort the list.
5. pop: Pop the last element from the list.
6. reverse: Reverse the list.

Initialize your list and perform set of commands where each command will be of the 6 types listed above. Iterate through each command in order and perform the corresponding operation on your list.

It is only necessary to complete the `process_list` function.

Example 1

```
process_list(["append 1", "append 2", "insert 3 1"]) == [1, 3, 2] # should be true
```

Explanation:

1. initialize empty list: arr = []
2. “append 1”: Append 1 to the list, arr = [1].
3. “append 2”: Append 2 to the list, arr = [1, 2].
4. “insert 1 3”: Insert 3 at index 1, arr = [1, 3, 2].

Input Format

process_list accepts list of strings. Each string contains one of the commands described above.

Constraints

The elements added to the list must be integers.

Output Format

List of integers.

Example 2

```
process_list([
  "insert 0 5",
  "insert 1 10",
```

```
  "insert 0 6",
  "remove 6",
  "append 9",
  "append 1",
  "sort",
  "pop",
  "reverse",
]) == [9, 5, 1]
```

Task 3

Task

Change the character at a given index and then return the modified string.

It is only necessary to complete the `mutate_string` function.

`mutate_string` has the following parameters:

1. string string: the string to change
2. int position: the index to insert the character at
3. string character: the character to insert

`mutate_string` returns the altered string

Example

```
mutate_string("abracadabra", 5, "X") == "abracXdabra" # should be true
```

Task 4

Intro

Given an integer, `n`, build a list of strings.

For each integer `i` from 1 to `n`, string with index `i` should contains the following representations of number `i`:

1. Decimal
2. Octal
3. Hexadecimal (capitalized)
4. Binary

It is only necessary to complete the `print_formatted` function.

Input Format

`print_formatted` has the following parameters:

1. int number: the maximum value

Output

For each line the four values must be included in the order specified above. Each value should be space-padded to match the width of the binary value of `number` and the values should be separated by a single space.

Constraints

$1 <= n <= 99$

Example

```
print_formatted(17) == [
    "    1     1     1     1",
    "    2     2     2    10",
    "    3     3     3    11",
    "    4     4     4   100",
    "    5     5     5   101",
    "    6     6     6   110",
    "    7     7     7   111",
    "    8    10     8  1000",
    "    9    11     9  1001",
    "   10    12     A  1010",
    "   11    13     B  1011",
    "   12    14     C  1100",
    "   13    15     D  1101",
    "   14    16     E  1110",
    "   15    17     F  1111",
    "   16    20    10 10000",
    "   17    21    11 10001",
]
```

Hints

1. There are built-in functions such as `bin(n)`, `hex(n)` and `oct(n)`. Run python interpreter and check how they work.
2. https://docs.python.org/3/library/string.html#formatstrings

Task 5

Task

You have a non-empty set `s`, and you have to execute given commands on it.

The commands will be `pop`, `remove n` and `discard n`. Use corresponding set methods (e.g. `your_set.pop()`). Figure out what they do and what's the difference.

It is only necessary to complete the `process_set` function.

Input Format

`process_set` has the following parameters:

1. List of numbers to create set from. All of the elements are non-negative integers, less than or equal to 9.
2. List of strings. Each string is either pop, remove or discard command followed by their associated value.

Output Format

`process_set` should return the sum of the elements of set on a single line.

Example

```
process_set(
    [1, 2, 3, 4, 5, 6, 7, 8, 9],
    [
        "pop",
        "remove 9",
        "discard 9",
        "discard 8",
        "remove 7",
        "pop",
        "discard 6",
        "remove 5",
        "pop",
        "discard 5",
    ]
) == 4
```

Explanation

After completing all of these operations on the set, we get set = (4). Hence, the sum of elements is 4.

Note: Convert the elements of set `s` to integers while you are assigning them.

Task 6

Task

You will be given two lists (A and B) of strings. For each string `sb` from list B check whether it has appeared in list A or not. Create list of the positions (index + 1) of each occurrence of `sb` in list A. If it does not appear, add -1 to the list. Return list of such list (length equals to the length of the list B)

It is only necessary to complete the `lists_intersections` function.

Example

List A contains 'a', 'b', 'a'. List B contains 'a', 'c'

For the first string in list B, 'a', it appears at positions 1 and 3 in list A. The second word, 'c', does not appear in group A, so corresponding positions list would be [-1].

```
lists_intersections(
    ['a', 'b', 'a'],    # list A
    ['a', 'c'],         # list B
) == [
    [1, 3],             # positions of the first string of list B
    [-1],               # positions of the second string of list B
]
```

Input Format

lists_intersections accepts two lists of strings.

Constraints

```
1 <= length of each string in the list A or B <= 100
```

Output Format

List of lists of integers.

Example 2

```
lists_intersections(
    ["a", "a", "b", "a", "b"]
    ["a", "b", "c"]
) == [
    [1, 2, 4],                  # 'a' appeared 3 times in positions 1, 2 and 4.
    [3, 5],                     # 'b' appeared 2 times in positions 3 and 5.
```

```
    [-1],                           # there is no `c` in list A => -1.
]
```

Hint: Use `from collections import defaultdict` to import defaultdict structure

Task 7

Task

You are given a string `S`. Your task is to find out whether `S` is a valid regex or not.

It is only necessary to complete the `is_valid_regex` function.

Example

`is_valid_regex` accepts string and should return bool.

```
is_valid_regex(r".*\+") == True
is_valid_regex(r".*+") == False
```

`.*\+`: Valid regex. `.*+`: Has the error multiple repeat. Hence, it is invalid.

Hint

Use `import re` to get regexp module `re`. Check docs for `re.compile` and figure out how to validate regexp using this function: https://docs.python.org/3/library/re.html#re.compile

Task 8

Intro

Integers in Python can be as big as the bytes in your machine's memory. There is no limit in size as there is: 2^31-1 (C++ int) or 2^63-1 (C++ long long int).

As we know, the result of a^b grows really fast with increasing b.

Task

Read four numbers `a`, `b`, `c` and `d`, and calculate the result of $a^b + c^d$.

It is only necessary to complete the `big_calc` function.

Example

```
big_calc(9, 29, 7, 27) == 4710194409608608369201743232
```

Constraints

$1 <= a, b, c, d <= 1000$

Note: This result is bigger than $2^63 - 1$. Hence, it won't fit in the long long int of C++ or a 64-bit integer.