◆ Question 1: Sum of Two Integers

Problem

Complete the function to compute the sum of two integers.

Function Description

Complete the function `solveMeFirst(a, b)` with the following parameters:

- `a`: integer
- `b`: integer

Returns: sum of `a` and `b`.

Constraints

```
1 ≤ a, b ≤ 1000
```

Sample Input

```
2
3
```

Sample Output

```
5
```

Solution

```
def solveMeFirst(a, b):
    return a + b   # Just add and return

# Input
num1 = int(input())
num2 = int(input())
res = solveMeFirst(num1, num2)
print(res)
```

Explanation

- The problem is **basic function definition**.
- Just return `a + b`.
- No need for extra if-else (constraints guarantee valid input).

Tip: Always start by identifying if constraints eliminate corner cases. Here, they do.

◆ Question 2: Success or Fail (Happy Number Problem)

Problem

Given a number, repeatedly apply the transformation:

- Replace the number with the **sum of squares of its digits**.
- If it becomes `1`, print `success`.
- If it enters a cycle, print `fail`.

Input Format

One integer `n`.

Output Format

`success` or `fail`.

Sample Input 0

```
49
```

Sample Output 0

```
success
```

(49 → 97 → 130 → 10 → 1 ✅)

Sample Input 1

```
11
```

Sample Output 1

```
fail
```

(11 → 2 → 4 → 16 → 37 → 58 → 89 → 145 → 42 → 20 → 4 → … cycle ❌)

Solution

```
def is_successful(num):
    seen = set()
    while num != 1 and num not in seen:
        seen.add(num)
        num = sum(int(digit) ** 2 for digit in str(num))
    return "success" if num == 1 else "fail"

# Input
n = int(input())
print(is_successful(n))
```

Explanation

- Use a **set** to detect cycles.
- If number repeats → cycle → fail.
- If reaches 1 → success.

Tip: When problems mention "repeating cycle," think of **hash sets** or **visited states**.

◆ Question 3: Festival Date (Easter-like Calculation)

Problem

Festival can fall between **March 22 and April 25**. Formula is based on given year and constants `M` and `N`.

Follow the **computus algorithm** (Gauss' Easter algorithm).

Input Format

One integer: year (1583–2299).

Output Format

"In {year} the Festival is on {Month} {Day}{Suffix}"

Sample Input

```
2011
```

Sample Output

```
In 2011 the Festival is on April 24th
```

Solution

```
def festival_date(year):
    # Table values for M and N
    if 1583 <= year <= 1699:
        M, N = 22, 2
    elif 1700 <= year <= 1799:
        M, N = 23, 3
    elif 1800 <= year <= 1899:
        M, N = 23, 4
    elif 1900 <= year <= 2099:
        M, N = 24, 5
    elif 2100 <= year <= 2199:
        M, N = 24, 6
    elif 2200 <= year <= 2299:
        M, N = 25, 0

    a = year % 19
    b = year % 4
    c = year % 7
    d = (19 * a + M) % 30
    e = (2 * b + 4 * c + 6 * d + N) % 7

    day = d + e
    if day < 10:
        month = "March"
        date = 22 + day
    else:
        month = "April"
        date = day - 9

    # Exceptions
    if date == 26 and month == "April":
        date = 19
    elif date == 25 and month == "April" and d == 28 and e == 6 and a > 10:
        date = 18

    # Ordinal suffix
    if 11 <= date <= 13:
        suffix = "th"
    else:
        suffix = {1:"st",2:"nd",3:"rd"}.get(date % 10, "th")

    print(f"In {year} the Festival is on {month} {date}{suffix}")

# Input
year = int(input())
festival_date(year)
```

Explanation

- Formula uses modular arithmetic to determine date.

- Special cases adjust April 25/26.
- Add ordinal suffix for natural English output.

Tip: Break formula into parts (`a, b, c, d, e`) to avoid mistakes. Always test exceptions.

◆ Question 1: Reverse a String

Problem

Given a string, print it in reverse.

Input Format

One string `s`.

Output Format

Reversed string.

Sample Input

```
hello
```

Sample Output

```
olleh
```

Solution

```
def reverse_string(s):
    return s[::-1]

s = input().strip()
print(reverse_string(s))
```

Explanation

- Python slicing `[::-1]` gives reverse.
- Very common in interviews.

Tip: Know string slicing tricks.

◆ Question 2: Palindrome Check

Problem

Given a string, check if it is a palindrome (reads the same forwards and backwards).

Input Format

One string `s`.

Output Format

`YES` if palindrome, `NO` otherwise.

Sample Input

```
madam
```

Sample Output

```
YES
```

Solution

```
def is_palindrome(s):
    return s == s[::-1]

s = input().strip()
print("YES" if is_palindrome(s) else "NO")
```

Explanation

- Palindrome check = compare string to its reverse.

Tip: Think symmetry problems → reverse or two-pointer technique.

◆ Question 3: Fibonacci Numbers

Problem

Print the first `n` Fibonacci numbers.

Input Format

Integer `n`.

Output Format

Fibonacci sequence separated by spaces.

Sample Input

```
5
```

Sample Output

```
0 1 1 2 3
```

Solution

```
def fibonacci(n):
    a, b = 0, 1
    result = []
    for _ in range(n):
        result.append(a)
        a, b = b, a + b
    return result

n = int(input())
print(*fibonacci(n))
```

Explanation

- Iterative approach avoids recursion overhead.
- Use tuple swap `a, b = b, a+b`.

Tip: Fibonacci shows up often → know iterative & recursive.

◆ Question 4: Prime Number Check

Problem

Given a number `n`, check if it is prime.

Input Format

One integer `n`.

Output Format

`Prime` or `Not Prime`.

Sample Input

```
7
```

Sample Output

```
Prime
```

Solution

```
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

n = int(input())
print("Prime" if is_prime(n) else "Not Prime")
```

Explanation

- Only check divisors up to `√n`.
- Common optimization.

Tip: Prime problems always → trial division up to √n.

◆ Question 5: Count Vowels in a String

Problem

Count how many vowels (`a, e, i, o, u`) are in the string.

Input Format

One string `s`.

Output Format

Integer count.

Sample Input

```
education
```

Sample Output

```
5
```

Solution

```
def count_vowels(s):
    vowels = "aeiouAEIOU"
    return sum(1 for ch in s if ch in vowels)

s = input().strip()
print(count_vowels(s))
```

Explanation

- Use membership test `in`.
- Case-insensitive by including uppercase.

Tip: For counting characters, **generator + sum()** is Pythonic.

Tips for Solving Longer Coding Questions

1️⃣ Read the story → Extract the core problem

- Many coding challenges wrap the real task in a *story*.
- Your first step: **ignore the story** and underline the *mathematical/computational task*.
- Example (Festival Problem):

> Story talks about hunters & tradition → actual task = implement Gauss' formula with conditions.

☞ **Tip:** Write the core requirement in 1 line for yourself.

2️⃣ Identify Inputs & Outputs clearly

- Almost always in competitive coding, the story ends with *Input Format* and *Output Format*.
- Example:
- `Input: year (1583–2299)`
- `Output: Date in Month-Day format with suffix`

☞ **Tip:** Write a sample input/output mapping to understand transformation.

3️⃣ Find the Pattern / Formula

- Some problems require formulas (like Easter date).
- Others require **repeated transformations** (like sum of squares until 1).
- Others need **simulation** (like matrix rotations).

☞ **Tip:** If you see numbers changing step by step → think "loop + transformation function".
☞ If you see conditions (if date = April 26 → shift) → think "edge case handling".

4️⃣ Break Problem into Functions

Instead of one giant code, **split into helper functions**:

- `def transform(num):` → for hunters problem
- `def suffix(day):` → for festival problem

☞ This makes debugging easier.

5️⃣ Handle Edge Cases

- Look for "exceptions" or "special rules" in the problem.
- Example: Festival date cannot be April 26 → handle that separately.

☞ **Tip:** Always test on given sample test cases — they usually cover edge cases.

6️⃣ Use Sets, Dictionaries, and Math Tricks

- **Cycle detection?** → use a `set` to store seen numbers.
- **Suffixes?** → use a dictionary `{1:"st",2:"nd",3:"rd"}`.
- **Big formulas?** → break down into variables `a, b, c, d, e`.

7⃣ Start with Sample Test Case → Dry Run

- Don’t rush into coding.
- Take the example input, work it out on paper, see how transformations work.
- Then match your steps with what the program should do.

◆ Mindset Tips

✅ Don’t panic when you see long “story problems” → 70% is *storytelling fluff*.
✅ Translate it into: “Given X, apply formula Y, output Z”.
✅ Always **solve small parts first** (like just calculating `d+e`, then later adding suffix).
✅ Debug using print statements when stuck.

◆ Example Walkthrough (Festival Problem)

1. Story → Festival date changes each year.
2. Input/Output → Input = year, Output = festival date.
3. Formula → Provided (M, N, etc.).
4. Plan → Implement step by step: calculate `a, b, c, d, e` → find day.
5. Edge Cases → handle April 25 & 26 rules.
6. Build functions → one for suffix, one for formula.
7. Dry run with 2011 → check it gives April 24th ✅.

☞ So the golden rule:
Story → Core Math → Formula/Loop → Edge Cases → Output Formatting