# Happy Number

Floyd's Tortoise and Hare cycle detection algorithm is used to detect whether a number, when repeatedly replaced by the sum of the squares of its digits, eventually falls into a cycle. For checking whether a number is a **Happy Number**, this is important because the sequence of sums of squares can either converge to `1` (indicating the number is happy), or it can start repeating in a cycle without reaching `1` (indicating the number is sad).

Here's why we need it:

1. **Happy Number Concept**:
    - If a number is happy, after repeatedly squaring its digits and summing them, we'll eventually reach the number `1`.
    - If a number is **not happy**, this process will eventually form a cycle where the sums will start repeating.

3. **Why Floyd's Cycle Detection**?
    - Floyd's Tortoise and Hare algorithm uses two pointers (`slow` and `fast`) that move through the sequence at different speeds:
        - `slow` moves one step at a time.
        - `fast` moves two steps at a time.
    - If there’s a cycle, the two pointers will eventually meet inside the cycle (because of their different speeds).
    - If no cycle exists (i.e., we eventually reach `1`), the `slow` and `fast` pointers will converge at `1`.

Without cycle detection, the program could incorrectly enter into an infinite loop, checking the same numbers over and over without realizing that it's in a cycle.

## Example Walkthrough for a Happy Number (`19`)

### Steps in the Sequence (Sum of Squares of Digits):

```
19 → 82 → 68 → 100 → 1
```

| Iteration | `slow` Value | `fast` Value | Explanation |
|---|---|---|---|
| 1 | 19 | 19 | Both pointers start at the same point. |
| 2 | 82 | 68 | `slow` moves by 1 step, `fast` by 2. |
| 3 | 68 | 1 | `fast` reaches `1` faster. |
| 4 | 100 | 1 | `slow` now moves closer to `1`. |
| 5 | 1 | 1 | Pointers meet at `1`, indicating success. |

### Steps in the Sequence for Sum of Squares of Digits:

For the number `20`, repeatedly replacing the number with the sum of the squares of its digits leads to a cycle:

`20 → 4 → 16 → 37 → 58 → 89 → 145 → 42 → 20`

#### Step-by-Step Computations

- $20 \to 2^2 + 0^2 = 4$
- $4 \to 4^2 = 16$
- $16 \to 1^2 + 6^2 = 37$
- $37 \to 3^2 + 7^2 = 58$
- $58 \to 5^2 + 8^2 = 89$
- $89 \to 8^2 + 9^2 = 145$
- $145 \to 1^2 + 4^2 + 5^2 = 42$
- $42 \to 4^2 + 2^2 = 20$

A cycle is detected, showing that the sequence does not converge to `1`.

## Floyd's Algorithm Walkthrough

### Starting Values

- `slow = 20`
- `fast = 20`

`20 → 4 → 16 → 37 → 58 → 89 → 145 → 42 → 20`

| Iteration | `slow` Value | `fast` Value | Explanation |
|---|---|---|---|
| 1 | 4 | 16 | `slow` moves 1 step: `20 → 4`. `fast` moves 2 steps: `20 → 4 → 16`. |
| 2 | 16 | 37 | `slow` moves 1 step: `4 → 16`. `fast` moves 2 steps: `16 → 37 → 58`. |
| 3 | 37 | 58 | `slow` moves 1 step: `16 → 37`. `fast` moves 2 steps: `37 → 58 → 89`. |
| 4 | 58 | 145 | `slow` moves 1 step: `37 → 58`. `fast` moves 2 steps: `58 → 89 → 145`. |
| 5 | 89 | 42 | `slow` moves 1 step: `58 → 89`. `fast` moves 2 steps: `145 → 42 → 20`. |
| 6 | 145 | 20 | `slow` moves 1 step: `89 → 145`. `fast` moves 2 steps: `20 → 4 → 16`. |
| 7 | 42 | 42 | `slow` and `fast` pointers meet at `42`, indicating a cycle. |

```c
// Function to check if a number is a Happy Number
int isHappyNumber(int num){

    int slow = num , fast = num;

    // cycle detection algorithm

    do {
        slow = Calculation(slow);                  // Move slow by one step
        fast = Calculation(Calculation(fast)); // Move fast by two steps
    } while (slow != fast)

    // The loop terminates when either:
    // 1. The pointers converge at 1
    // 2. The pointers converge at some other number (cycle)



    // After exiting the loop, the function checks if the value of slow is 1.
    return (slow==1);
}
```

```
// Function to calclulate the sum of squares of digits of a number

int Calculation(int num){

    int sum = 0;
    while (num != 0){
        int last_digit = num % 10;          // Find   the last digit
        sum + =  last_digit*last_digit;     // Square the last digit
        num = num /10;                      // Remove the last digit
    }
    return sum;
}
```

- last_digit = 456 % 10 = 6
- Sum = last_digit * last_digit = 36
- num = num / 10 = 45
- last_digit = 45 % 10 = 5
- Sum = last_digit * last_digit = 36 + 25 = 61
- num = num / 10 = 4
- last_digit = 4 % 10 = 4
- Sum = last_digit * last_digit = 36 + 25 + 16 = 77
- num = num / 10 = 0 → Termination condition

## Example Walkthrough

### Input:

`num = 123`

### Step-by-Step Execution:

| Iteration | `num` | `last_digit` | `last_digit^2` | `sum` |
|---|---|---|---|---|
| Initial | 123 | - | - | 0 |
| 1 | 123 | 3 | 9 | 9 |
| 2 | 12 | 2 | 4 | 13 |
| 3 | 1 | 1 | 1 | 14 |
| End | 0 | - | - | 14 |