* Check prime :

Eg: n = 17

op: true

Eg: n = 25

op: false

Eg: 25

[2, 24]

25 % 2 == 0

25 % 3 == 0

25 % 4 == 0

25 % 5 == 0

25 % 6

:

25 % 24 == 0

not a prime

1. In general in real world, how will you solve this.

what is a prime number ?

→ a number is prime if it is only divisible by 1 and itself, (not divisible by any other number apart from 1, itself)

Eg: 103

[2, 102]

103 % 2 == 0

103 % 3 == 0

103 % 4 == 0

. . .

. . .

. . .

. . .

. . .

103 % 102 == 0

* factors of number ⇒ number % x == 0

⇒ then x is a factor of that number

Q: Is 103 a prime number ?

a) true

b) false

c) none

2. How to convert the previous approach to code ?

variables,
conditionals,
loops,
type conversions etc...

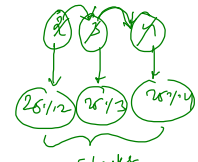
N = 25 → we are checking from (2 → 24) (2 → N-1)

① for loop helps us to take from 2 to 24

num = 2 3 4 5 6 ..... 24 25 ✗ num < N
num < 25
(or) num <= 24
num <= N-1

2 → init, +1 → operation, 25 → condition

```
for (let num = 2; num < N ; num ++) {
    if (N % num == 0) {
        console.log ("Not a prime");
    }
    else {
        console.log ("a prime");
    }
}
```



* we found a number apart from 1 & N which divides N → not a prime

eg: N = 25

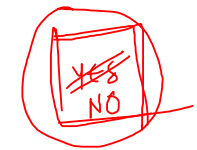
num = 2 → 3 → 4 → 5

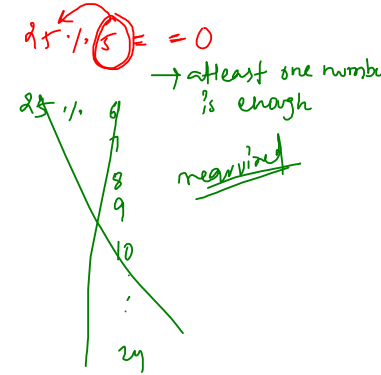
num	25 % num == 0	
2	25 % 2 == 0 (false)	"a prime"
3	25 % 3 == 0 (false)	"a prime"
4	25 % 4 == 0 (false)	"a prime"
5	25 % 5 == 0 (true)	"not prime"

→ The 1st version of code did not work, because we are deciding by looking at a single number every time, we need to check with all [2, N-1] and then only decide.

```
let isPrime = true;   (assume it is prime)
for (let num = 2; num < N; num++) {
    if (N % num == = 0) {
        isPrime = false;
        break;
    }
}
if (isPrime) console.log("prime")
else console.log("not prime")
```

* check/flag variable





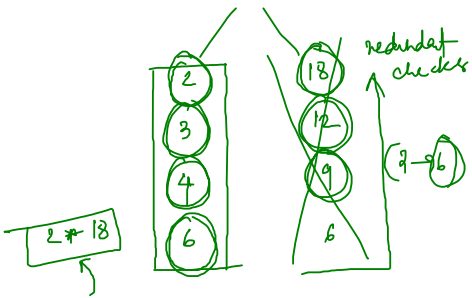
✱ Further Improving check prime :

(9 → 25)

→ earlier approach takes N iterations / N checks

(can we decrease this checks / iterations) ⇒ $\sqrt{n}$ checks / iterations

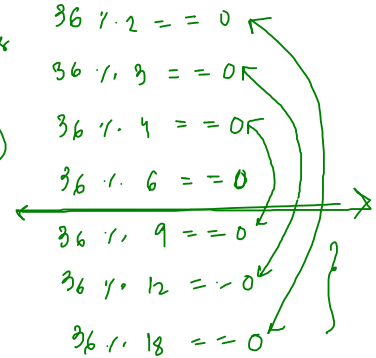
eg : 36 (factors of 36)

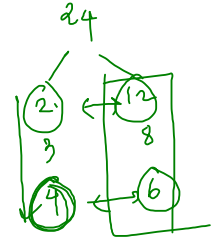


36 % 2 == 0 ↖ Both are same

36 % 18 == 0 ↙

18 * 2





(9 → $\sqrt{n}$)

9 → $\sqrt{36}$ ⇒ 9 → 6

9 → $\sqrt{24}$ ⇒ 9 → 4

36

1. 2, 3, 4, 5, 6, 7, 8, 9, ... 18 ... 35 (2 → N-1)
2. 2, 3, 4, 5, 6 (2 → $\sqrt{n}$) → why?

36 % 2 == 0 → 36 % 18 == 0

are same checks (duplicates) — not necessary redundant

1) 36 % 4 == 0 → 36 / 9 == 0

4, 9

duplicates

36/4 = 9

36/9 = 4

num <= $\sqrt{n}$

(num)^2 <= ($\sqrt{n}$)^2

(num)^2 <= n

num * num <= n

(or)

num ** 2 <= n

Q: Sum of digits :

Eg: n = 1325

op: 1+3+2+5 = 11

Eg: n = 565423

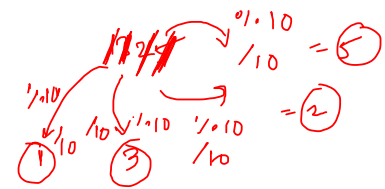
op: 5+6+5+4+2+3 = 25

1. How do I get Individual digits from a number ?

1325 % 10 = 5
↓ 1325/10
132 % 10 = 2
↓ 132/10
13 % 10 = 3
↓ 13/10
1 % 10 = 1
↓ 1/10
0 ✗

⇒ a + b + c + d

num % 10 (last dig)
→ remove last dig
→ num = num/10



```
function SumofDigits(n) {
  let sum = 0;
  n = Number(n);
  while (n > 0) {
    const digit = n % 10; // extract digit
    sum = sum + digit; // sum += digit;
    n = parseInt(n / 10); // remove the digit
  }

  console.log(sum);
}
```

n = 1325 sum = 0 5 7 10 (11)
132 13 1 0

① 1325 > 0
digit = 1325 % 10 = 5
sum = 0 + 5 = 5
n = pI (1325/10) = 132

② 132 > 0
digit = 132 % 10 = 2
sum = 5 + 2 = 7
n = pI (132/10) = 13

③ 13 > 0
digit = 13 % 10 = 3
sum = 7 + 3 = 10
n = p± (13/10) = 1

④ 1 > 0
digit = 1 % 10 = 1
sum = 10 + 1 = 11
n = pI (1/10) = 0

⑤ 0 > 0 (false)
terminate. while

★ In 1st iteration, digit variable is created 

and when the while loop or any other loop moves to the next iteration all the things that are newly created in previous iteration are deleted/removed. That means when it goes to 2nd iteration a one more new 

is created, the old digit is lost already.

Q: Reverse a numba :

eg: n = 1 3 2 5

op: 5 2 3 1

5	2	3	1
1000	100	10	1
1	3	2	5

⇒ 1*5 + 10*2 + 100*3 + 1000*1

⇒ 5 + 20 + 300 + 1000

= 1325

1325 % 10 = 5

1325/10 ↓

132 % 10 = 2

132/10 ↓

13 % 10 = 3

13/10 ↓

1 % 10 = 1

1/10 ↓

0 ✗

rev = 0

rev = rev * 10 + digit

① rev = 0 * 10 + 5
= 0 + 5 = 5

② rev = 5 * 10 + 2
= 50 + 2 = 52

③ rev = 52 * 10 + 3
= 520 + 3 = 523

④ rev = 523 * 10 + 1
= 5230 + 1 = 5231

1000 100 10 1 → 1000 100 10 1

1 3 2 5 → 5 2 3 1

new = new * 10 + d

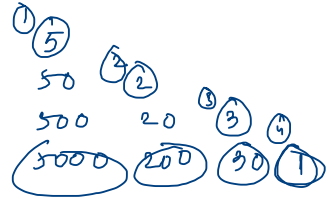
1325 % 10 = 5 * 1000

132 % 10 = 2 * 100 +

13 % 10 = 3 * 10 +

1 % 10 = 1 * 1

5000 + 200 + 30 + 1 = 5231



* solve all problems of loops-1

(palindrome, HCF, power)

→ (armstrong numbers in range)

(link in chat) (gym)