Variables & Data types

- Immutable (can't be changed) → Default
- Mutable (can be changed).

let var = 100; * → Immutable variable

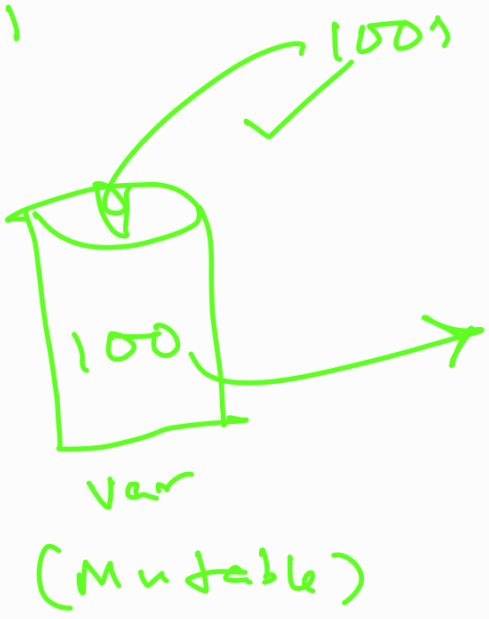
let var;
var = 100;

let var = 100; (Immutable)

X Var = 1001;

let mut var = 100;

var = 1001



(Mutable)

Constants :-

```
const A_CONSTANT : data_type = 100;
```

✓
const PI : float = 3.14; ✗ reassigned.

immutable var ✗ reassigned

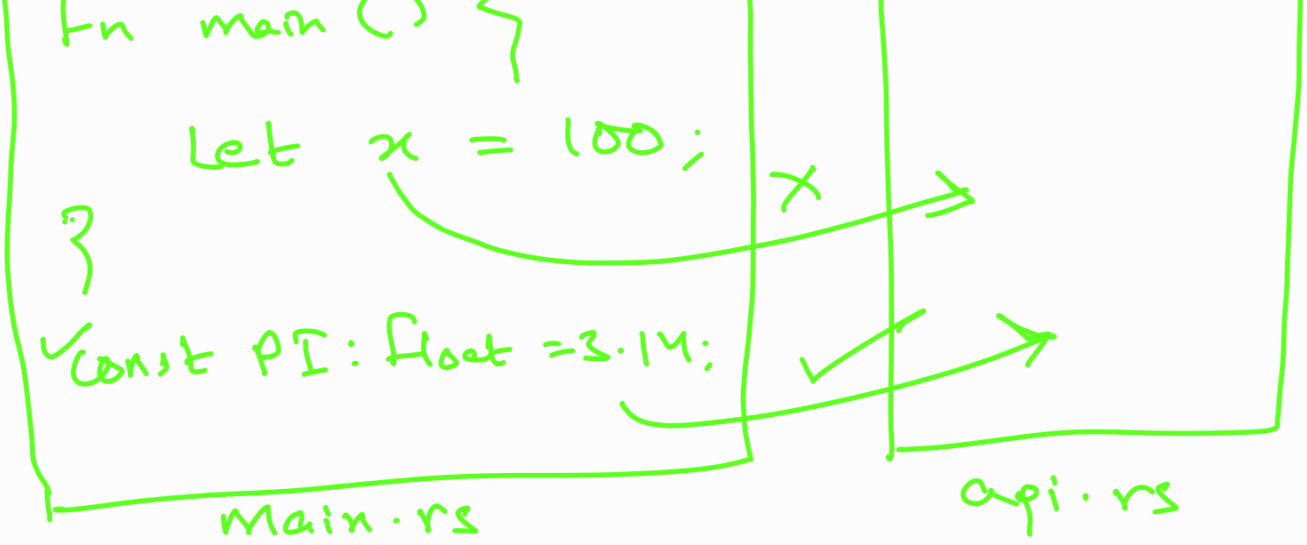
Scope — Globally (const) ✓

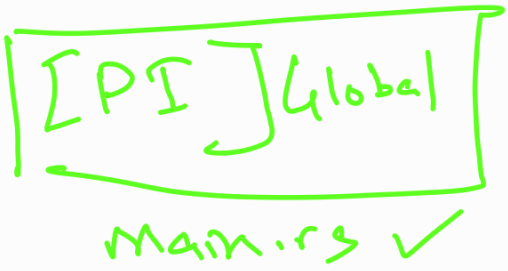
Locally (variables)

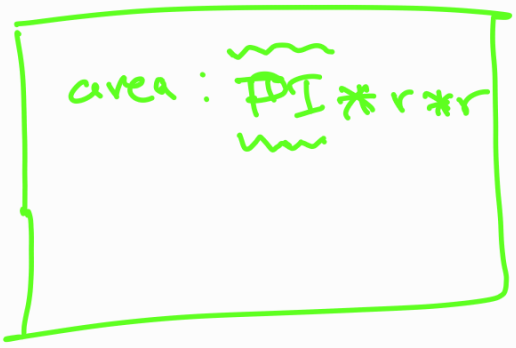
main.rs — const PI (Global)

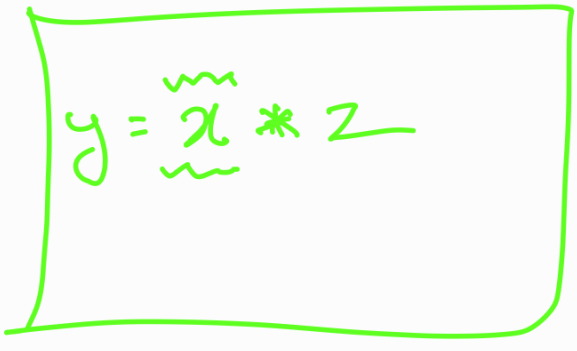
api.rs ✓

blockchain.rs ✓









Shadowing:

1 let x : integer = 30; — L1

x → 30

print(x) → 30

99

100 let x : float = 30.98; — L2

x → 30.98

print(x) → 30.98.

* Data Type
* Value

} variable.

let mut x = 30; (integer)

x = 31; ✓ (integer)

✗ x = 30.98; (float)

✗ x = true; (boolean)

x = 948;

let x = 30; x

let x = 30; ✗

let x = 30.98; ✓

let x = true; ✓

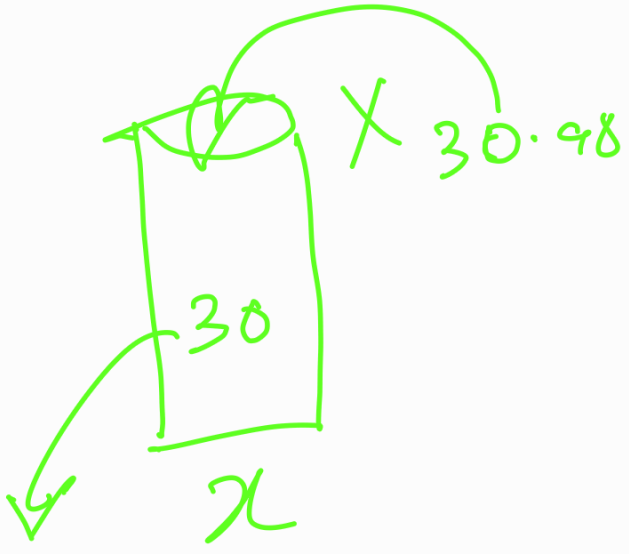
⋮

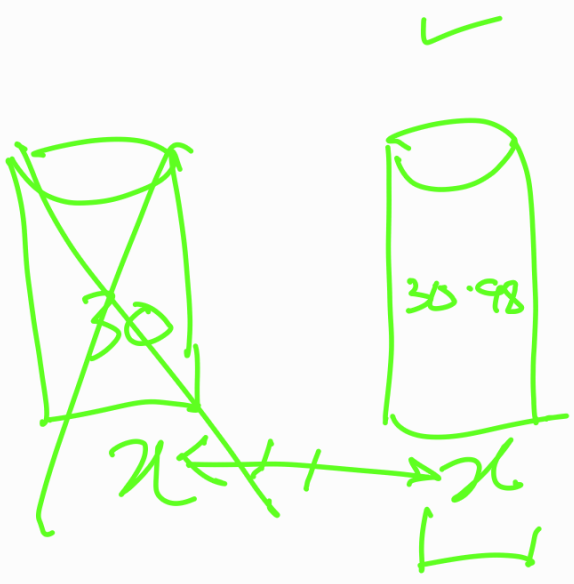
let x = 30;

x = 30.98;

let x = 30;

let x = 30.98;





⋮

x — 1 mar

y - 1 new

mut

x = 10

x = 11

x = 12

Data Types

- Scalar (A single value)
- Compound (Composed of multiple single values)

Integers
- Signed (i8 --- i128)
- Unsigned (u8 --- u128)

Floats — f64, f32

Booleans — true, false.

Characters - 'a', 'b', 'c'

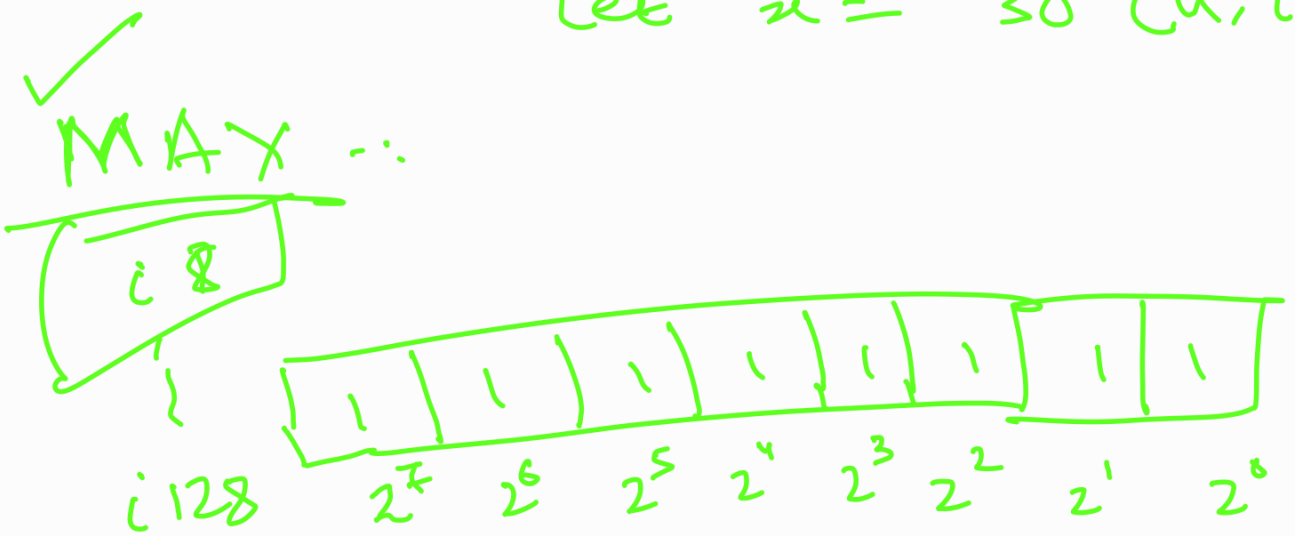
64 bits
32 bits

\+

let x = -30 (i...)

\-

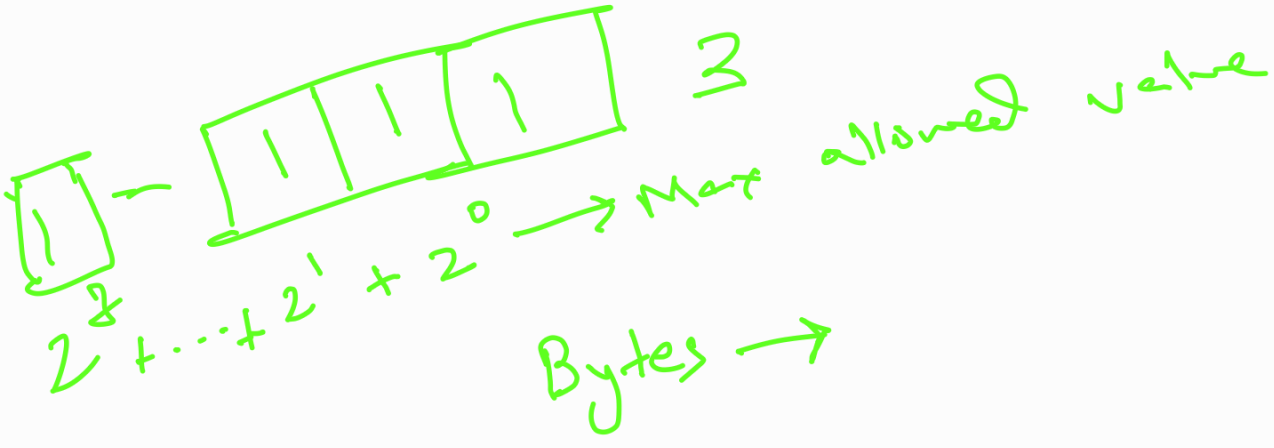
Let x = 30 (u, i)



$(2^2 \times 1) + (2^1 \times 1) + (2^0 \times 1)$

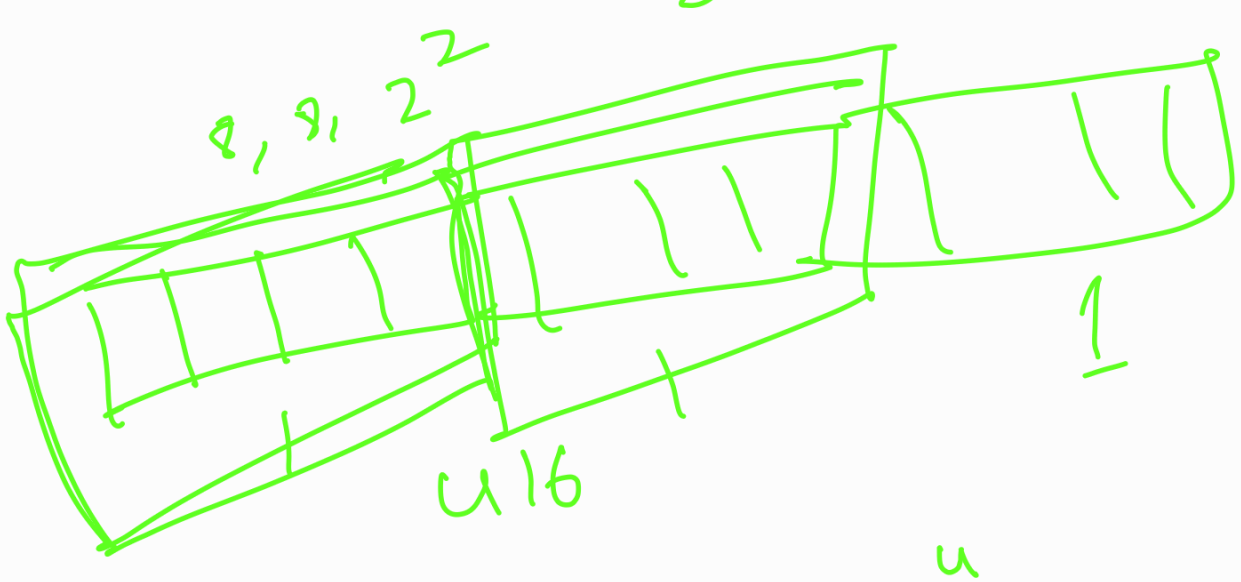
i8 :- A signed integer of 8 bits

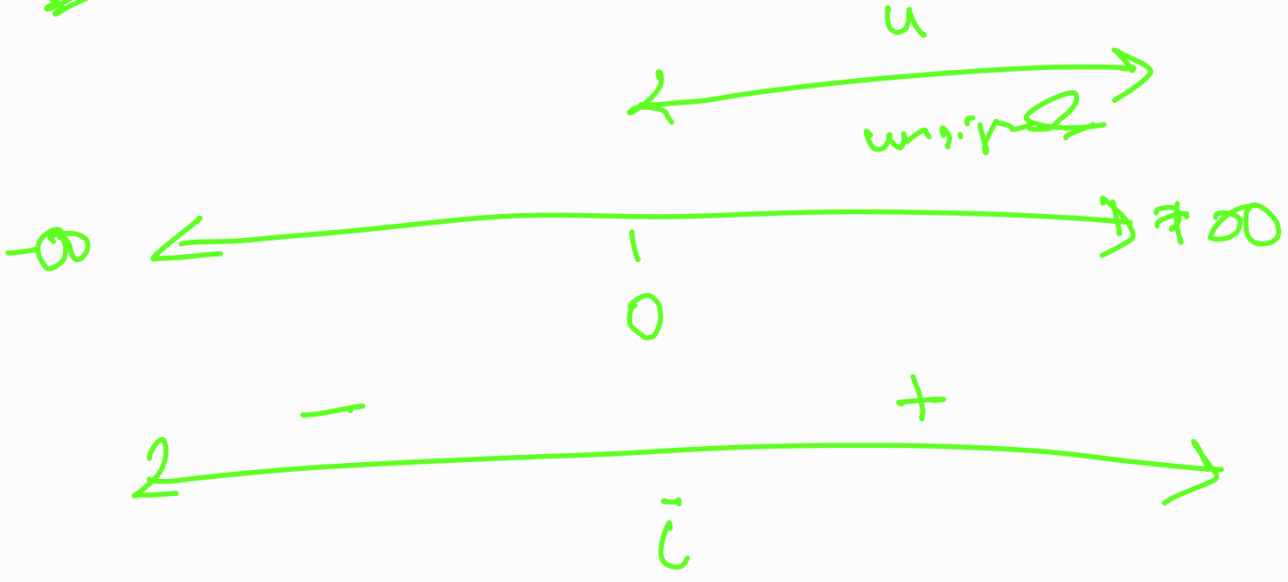
i128 :- A " " " 128 bits.



u8 100 — u128 = 100

8 bits = 1 byte





27/06/24

Compound Data Types :-

- Tuple
- Array [*** Not your usual array]

Tuple :- (...) (5 integers, 4 floats) → One value.

let tup = (, , , ,);

integers

u8, i32, i32, u64

f32

bool

A tuple is a single compound element.

() → Unit Type

0 1 2 3 4

5 − 1 = 4

len − 1

0

let x = (1, 2, 3, 4, 5);

let first = x.0;

" second = x.1;

tup.len()

tup.len

let last_ele = len − 1;

⋮

= x.4 / x.last_index;

Allocated on the "stack" and not on "heap."

Length of tuple = b

b − 1

0 ..... b − 1 ⇒ b elements

1 ---- b ⇒ b elements.

Size of the tuple remains constant.

```
let tup = (1, 2, 3, 4, 5);
```

X Add

X Remove

* A tuple cannot grow / shrink in size.

Array::

```
let arr : [i32; 5] = [0, 1, 2, 3, 4];

let var_name : [Type; Size] = Value;


// When my array has Different elements.
let arr : [i32; 2] = [1, 2];

let arr = [0; 100];.

let first = arr[0]; // typ. 0
let second = arr[1];
```

```
: arr[len-1];
```

Array is allocated on stack.

No growth

No shrinkage