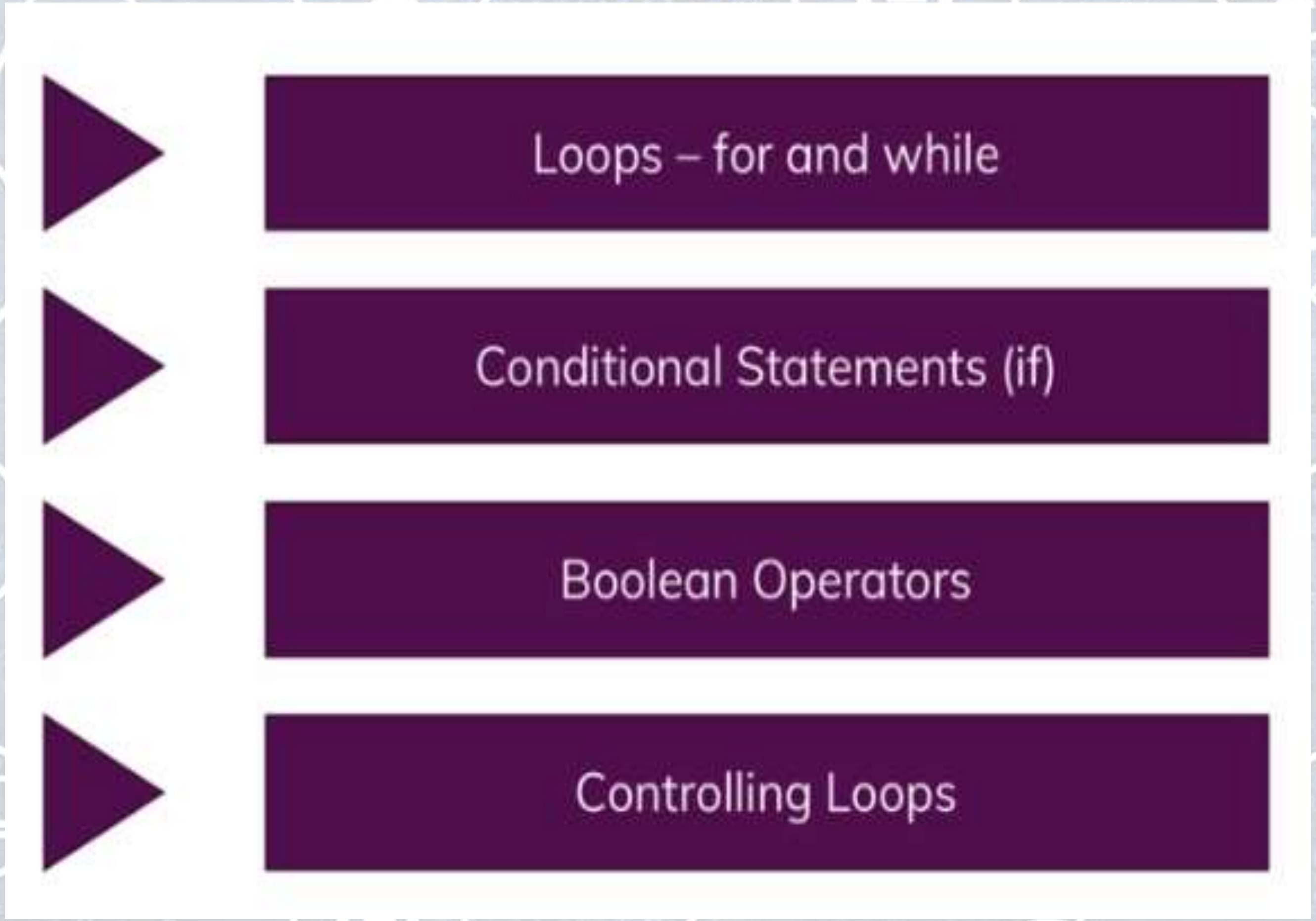
Welcome

# :Today’s Topic:[Python]



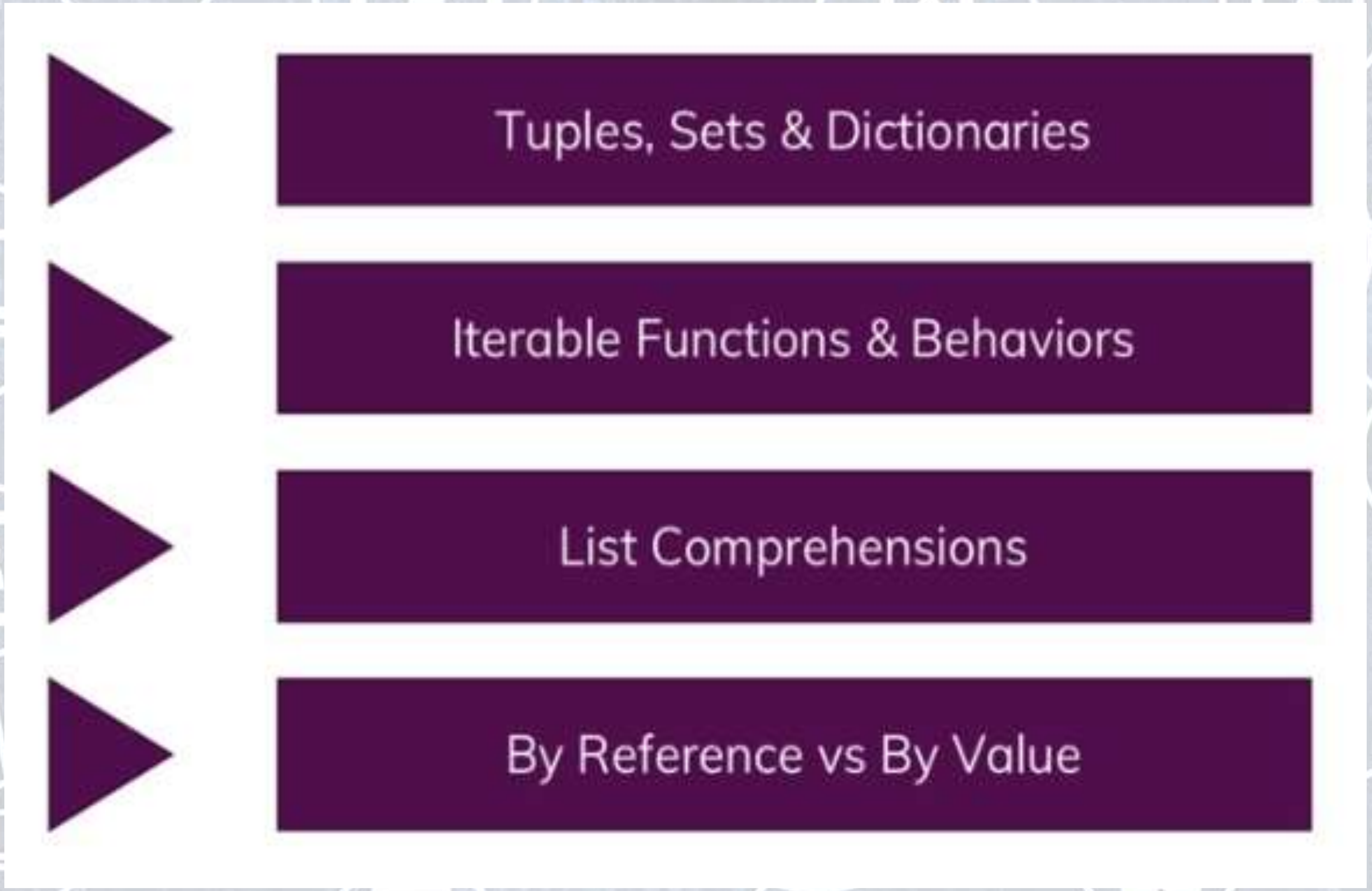
Loops – for and while

Conditional Statements (if)

Boolean Operators

Controlling Loops

## :Today’s Topic:[Python] [Cont.]



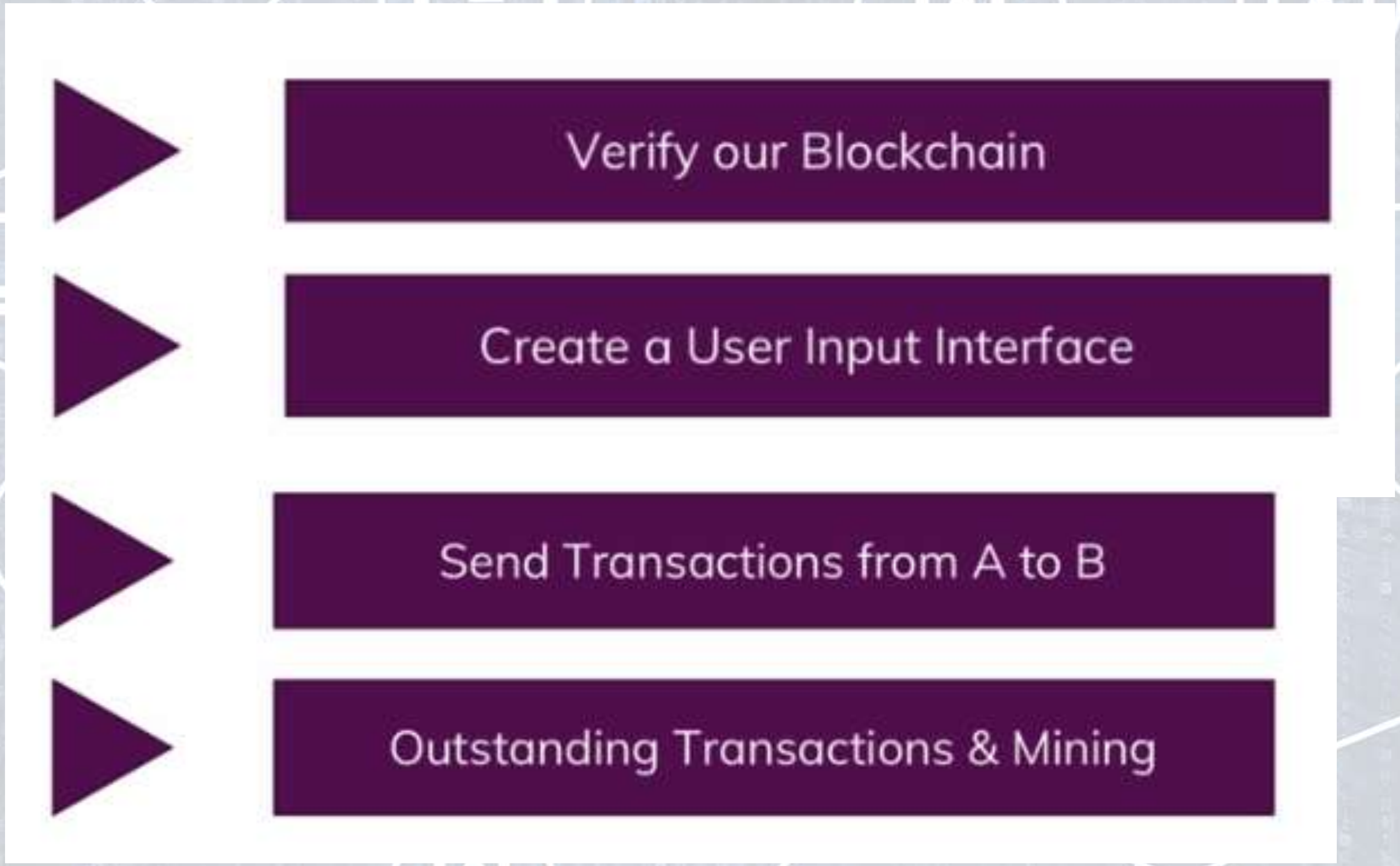
Tuples, Sets & Dictionaries

Iterable Functions & Behaviors

List Comprehensions

By Reference vs By Value

# :Today’s Topic:[Block Chain Perspective]



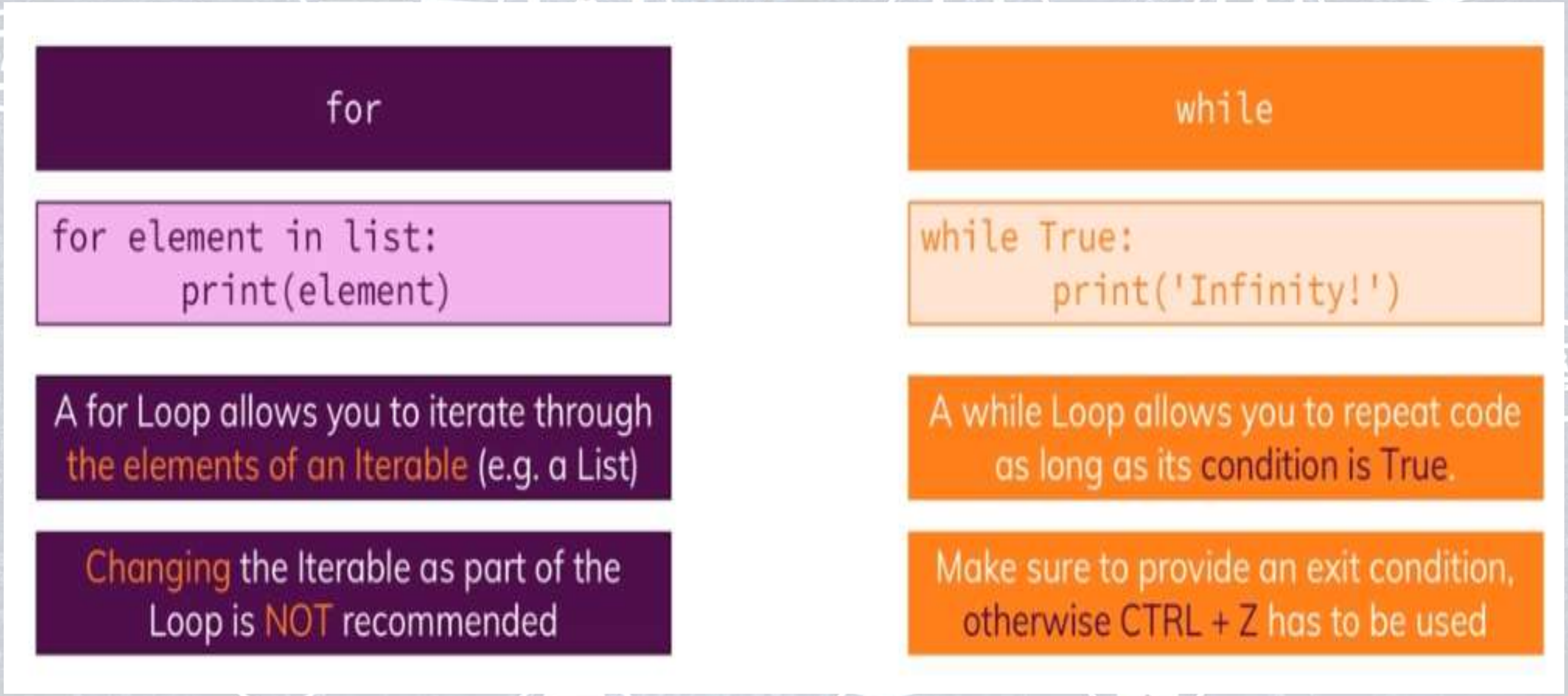
Verify our Blockchain

Create a User Input Interface

Send Transactions from A to B

Outstanding Transactions & Mining

# Loop



for

```
for element in list:
        print(element)
```

A for Loop allows you to iterate through the elements of an Iterable (e.g. a List)

Changing the Iterable as part of the Loop is NOT recommended

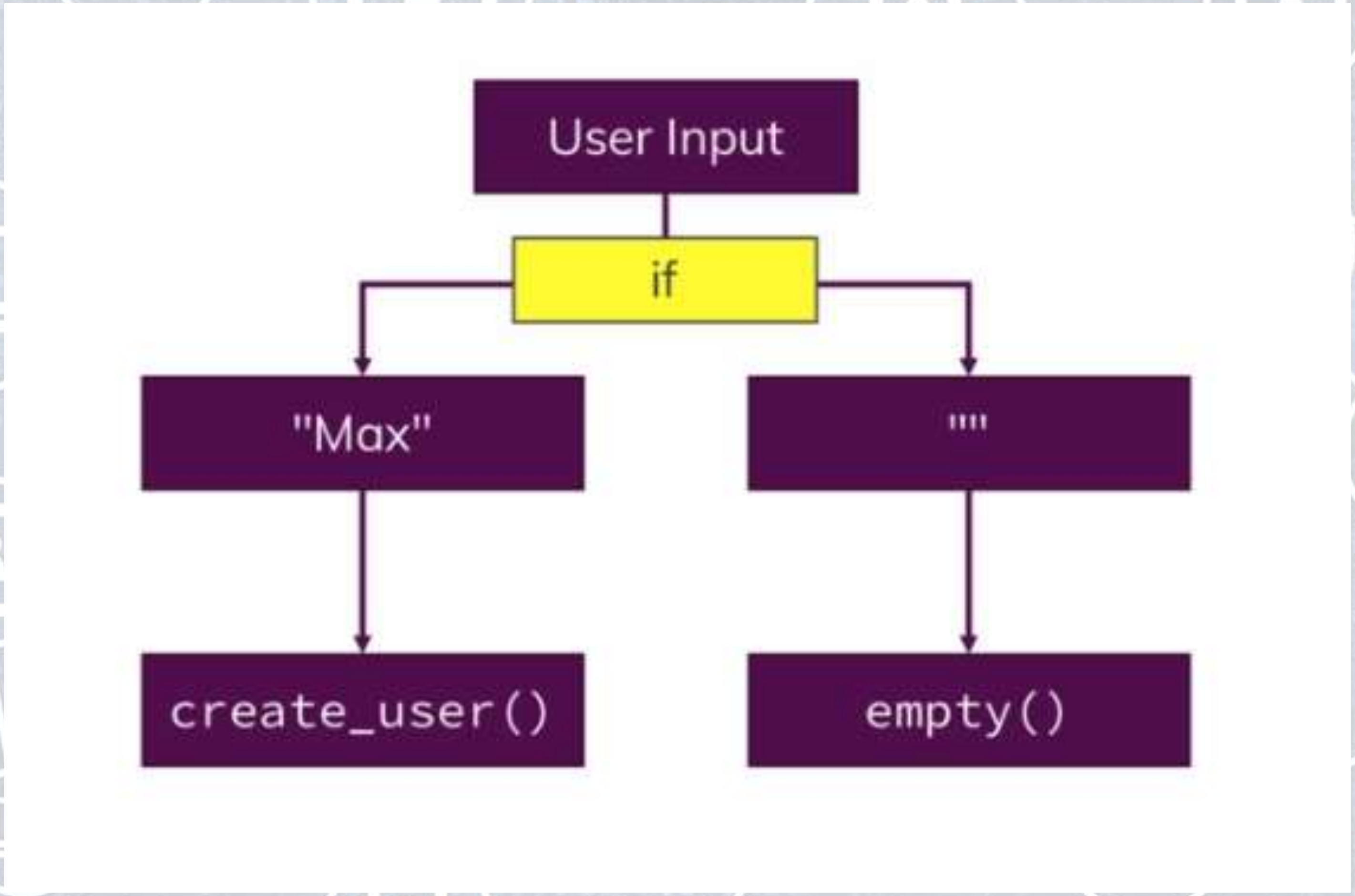
while

```
while True:
        print('Infinity!')
```

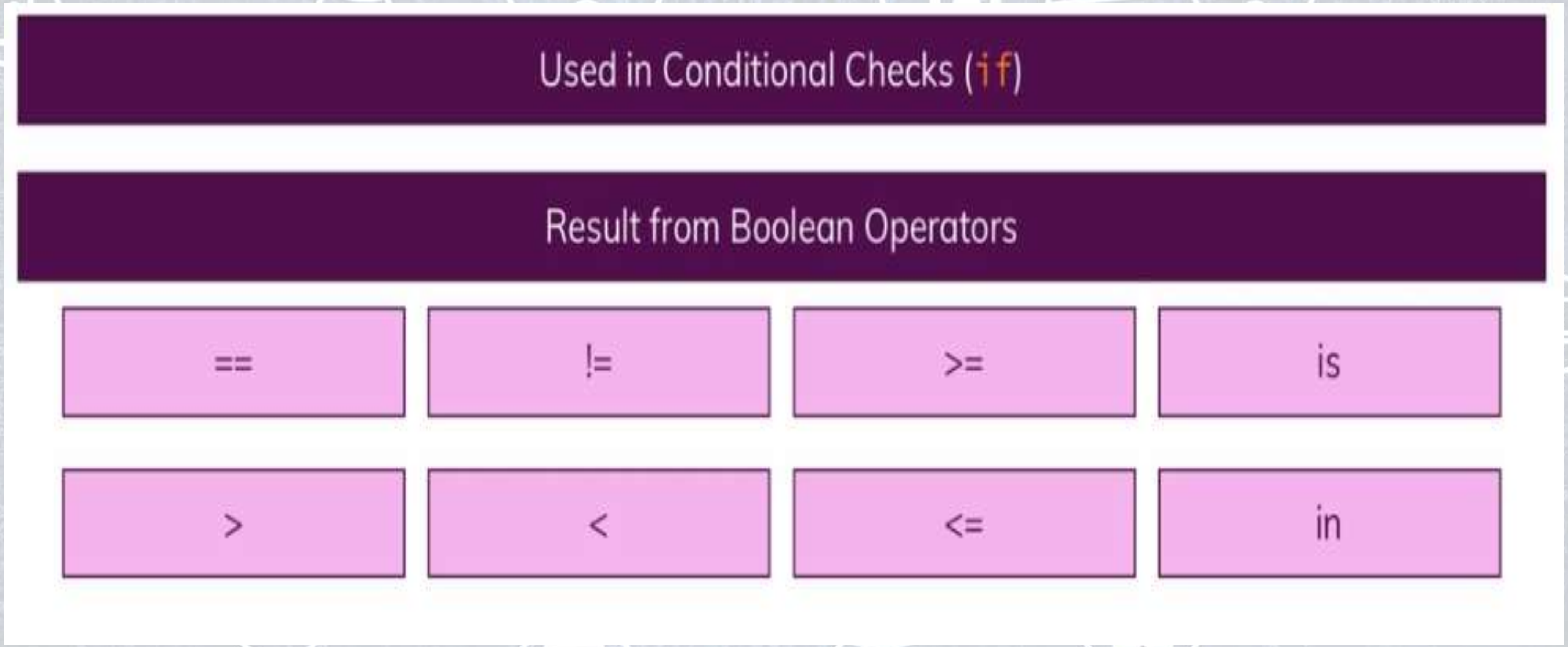
A while Loop allows you to repeat code as long as its condition is True.

Make sure to provide an exit condition, otherwise CTRL + Z has to be used

# If-else



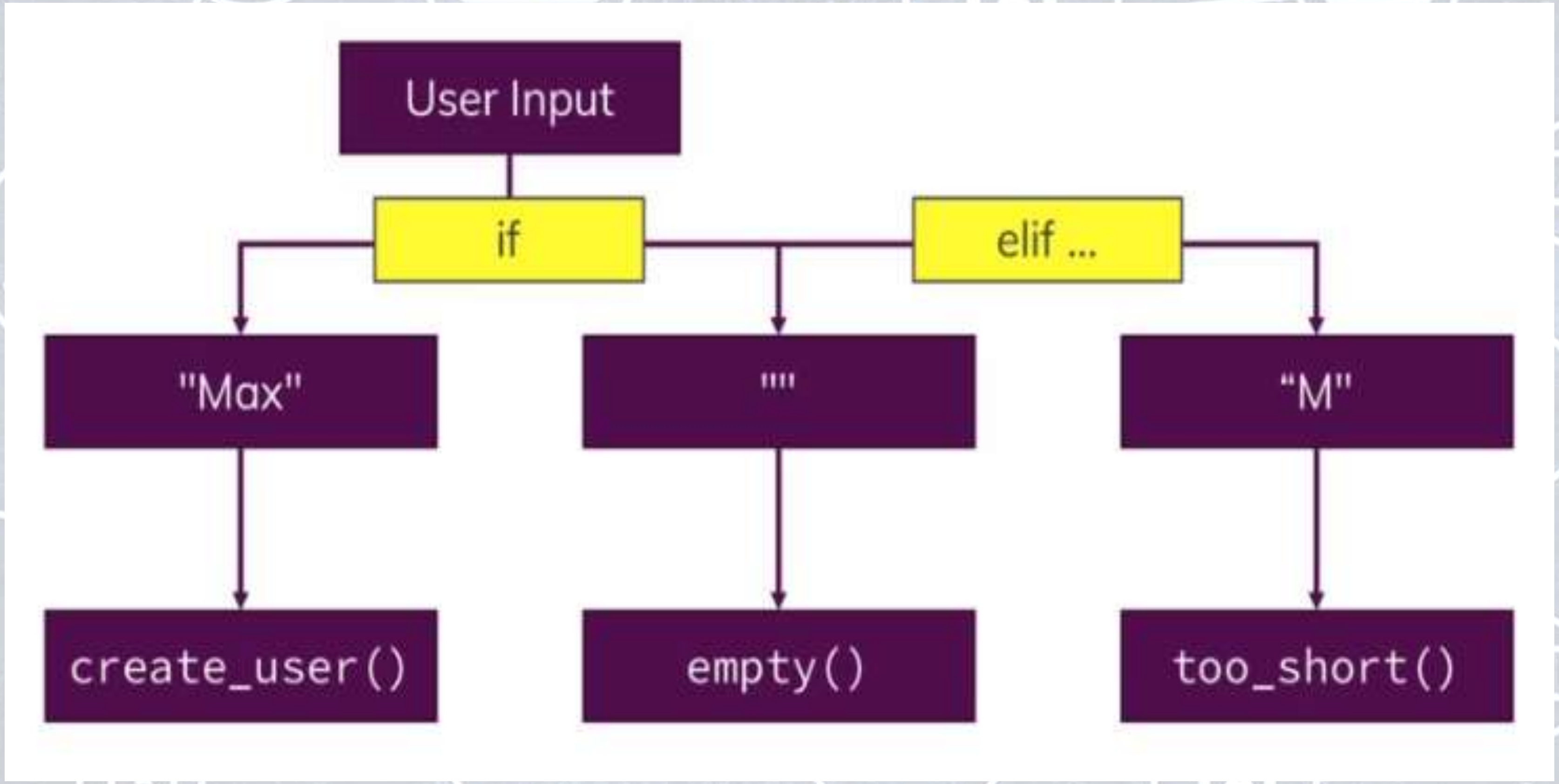
# :Conditional Operators:


Used in Conditional Checks (if)

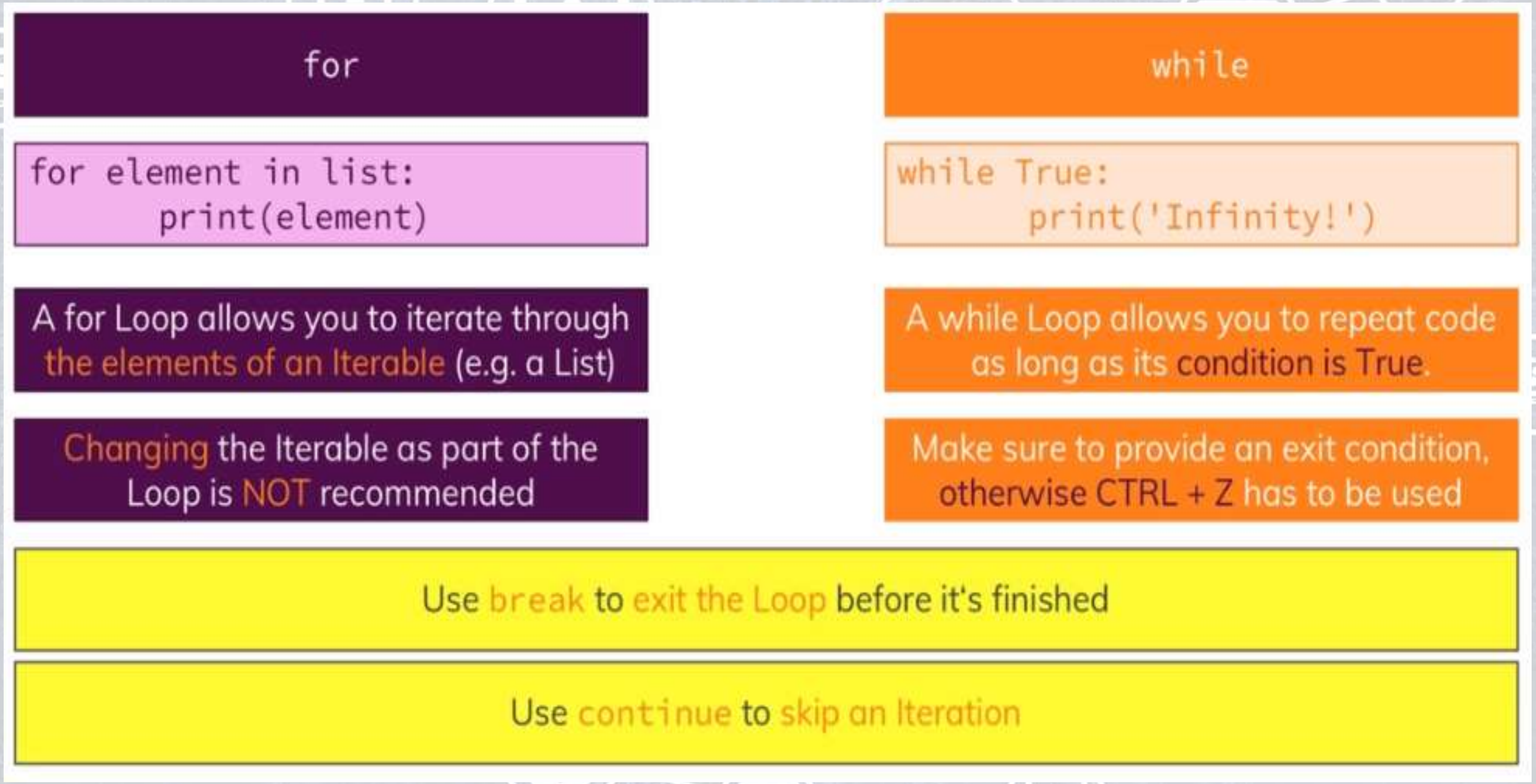
Result from Boolean Operators

| | | | |
|---|---|---|---|
| == | != | >= | is |
| > | < | <= | in |

## If-else [Cont.]



# Loop [Conti.]

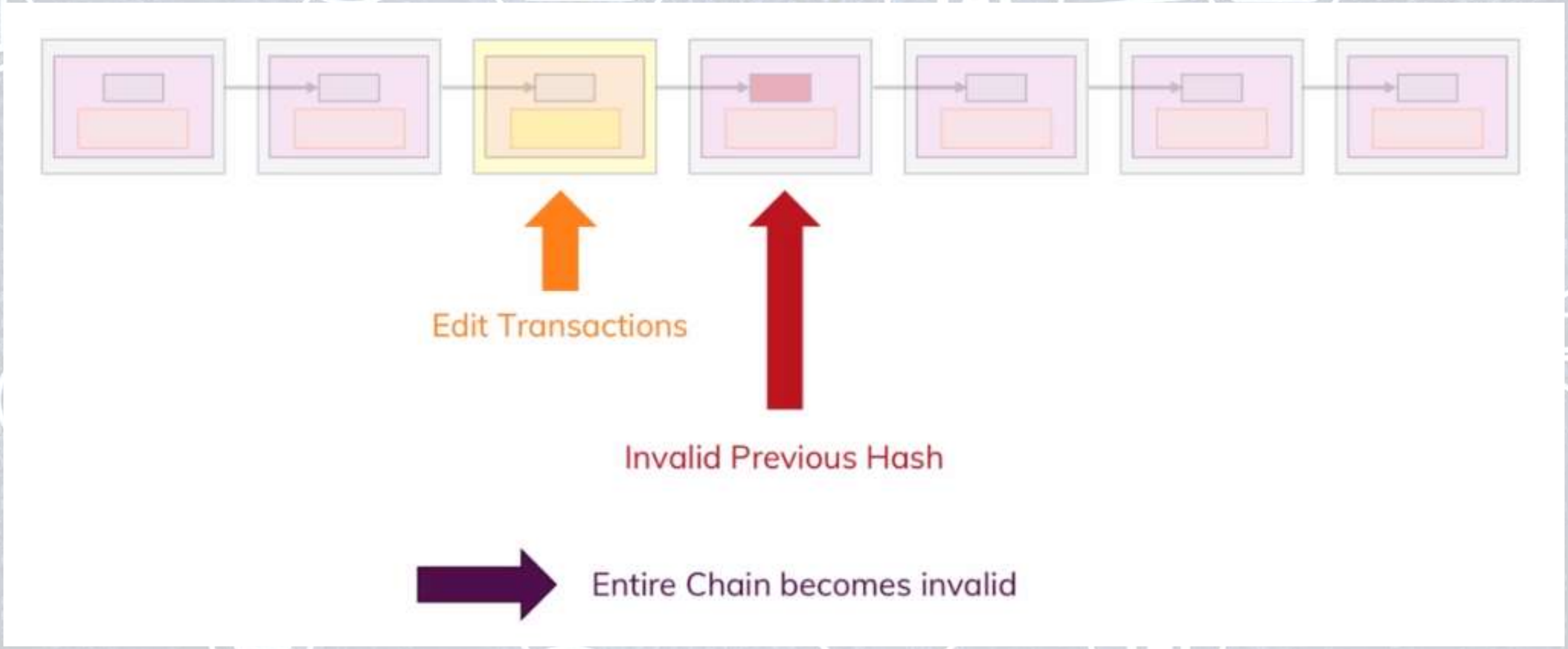


| for | while |
|---|---|
| `for element in list:`<br>`    print(element)` | `while True:`<br>`    print('Infinity!')` |
| A for Loop allows you to iterate through the elements of an Iterable (e.g. a List) | A while Loop allows you to repeat code as long as its condition is True. |
| Changing the Iterable as part of the Loop is NOT recommended | Make sure to provide an exit condition, otherwise CTRL + Z has to be used |

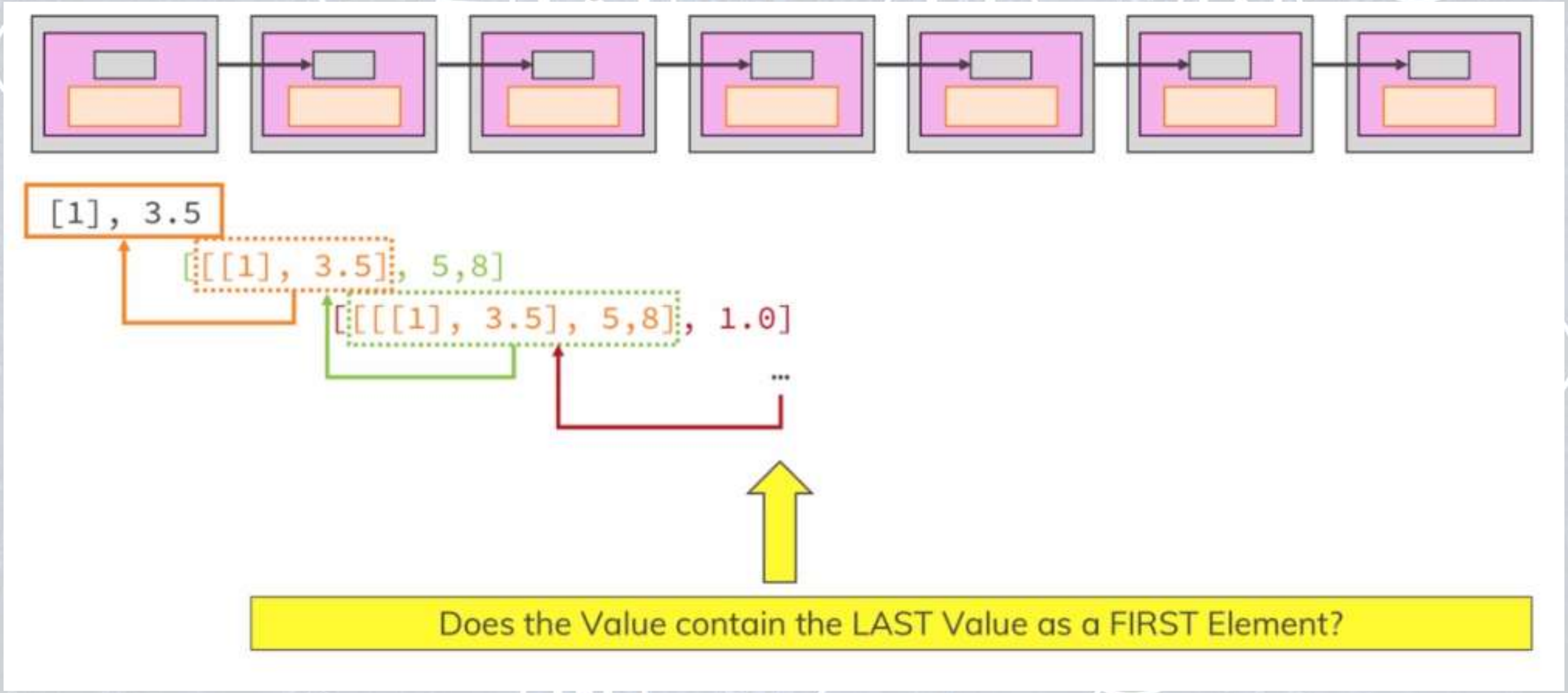
Use break to exit the Loop before it's finished

Use continue to skip an Iteration

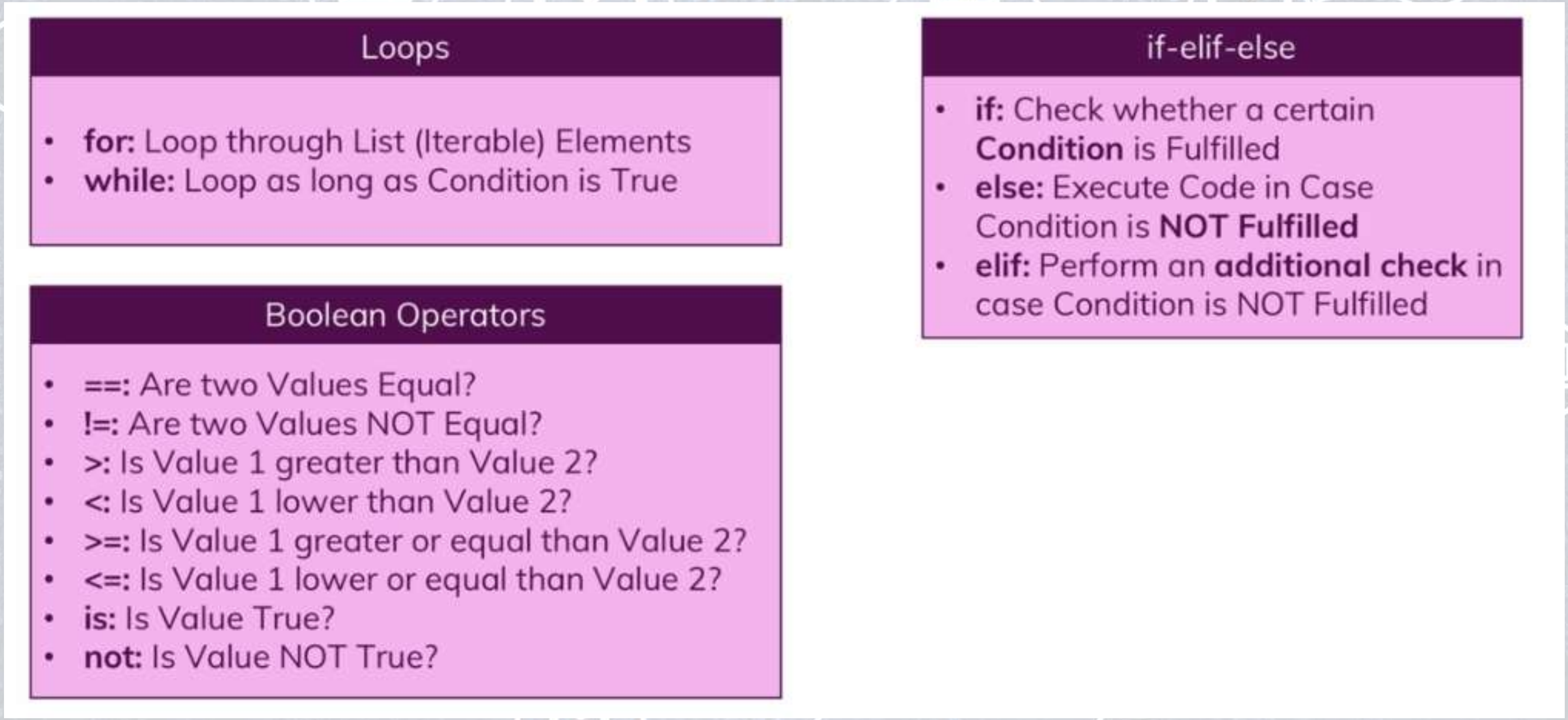
# Manipulating Block Chain



# Verifying Our Block Chain



# :Till Now:



### Loops

- **for:** Loop through List (Iterable) Elements
- **while:** Loop as long as Condition is True

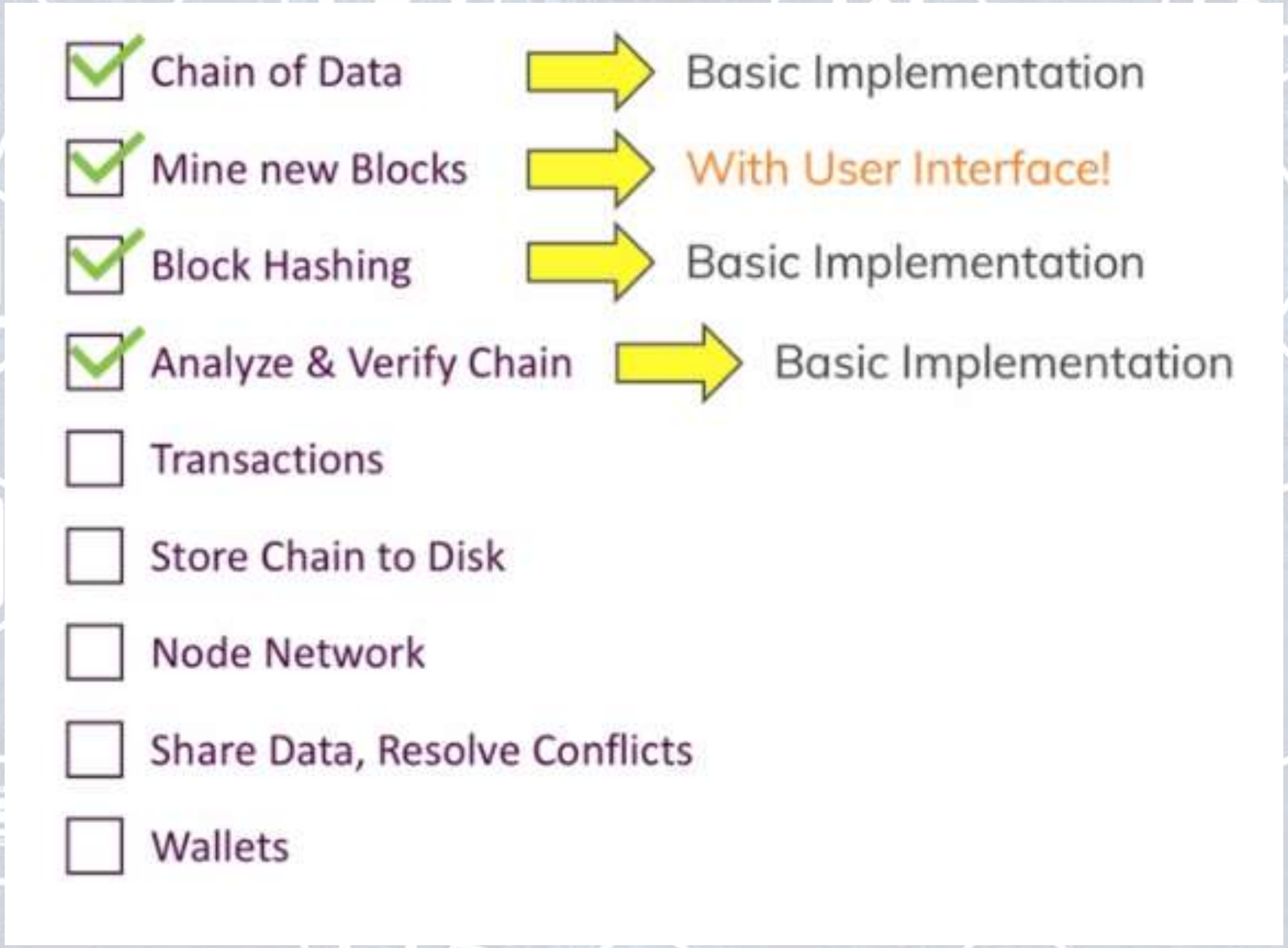
### Boolean Operators

- **==:** Are two Values Equal?
- **!=:** Are two Values NOT Equal?
- **>:** Is Value 1 greater than Value 2?
- **<:** Is Value 1 lower than Value 2?
- **>=:** Is Value 1 greater or equal than Value 2?
- **<=:** Is Value 1 lower or equal than Value 2?
- **is:** Is Value True?
- **not:** Is Value NOT True?

### if-elif-else

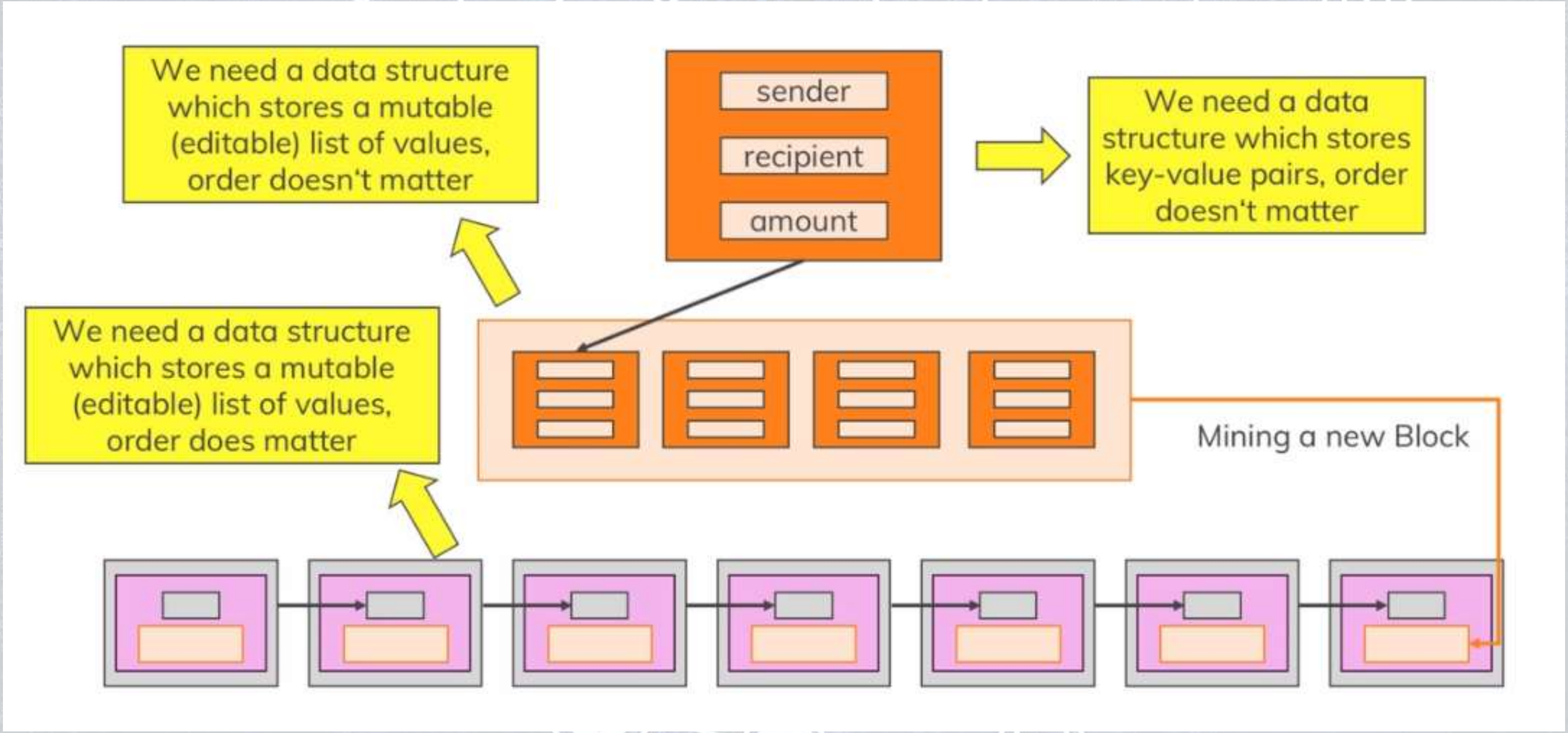
- **if:** Check whether a certain **Condition** is Fulfilled
- **else:** Execute Code in Case Condition is **NOT Fulfilled**
- **elif:** Perform an **additional check** in case Condition is NOT Fulfilled

# :Our Block Chain:

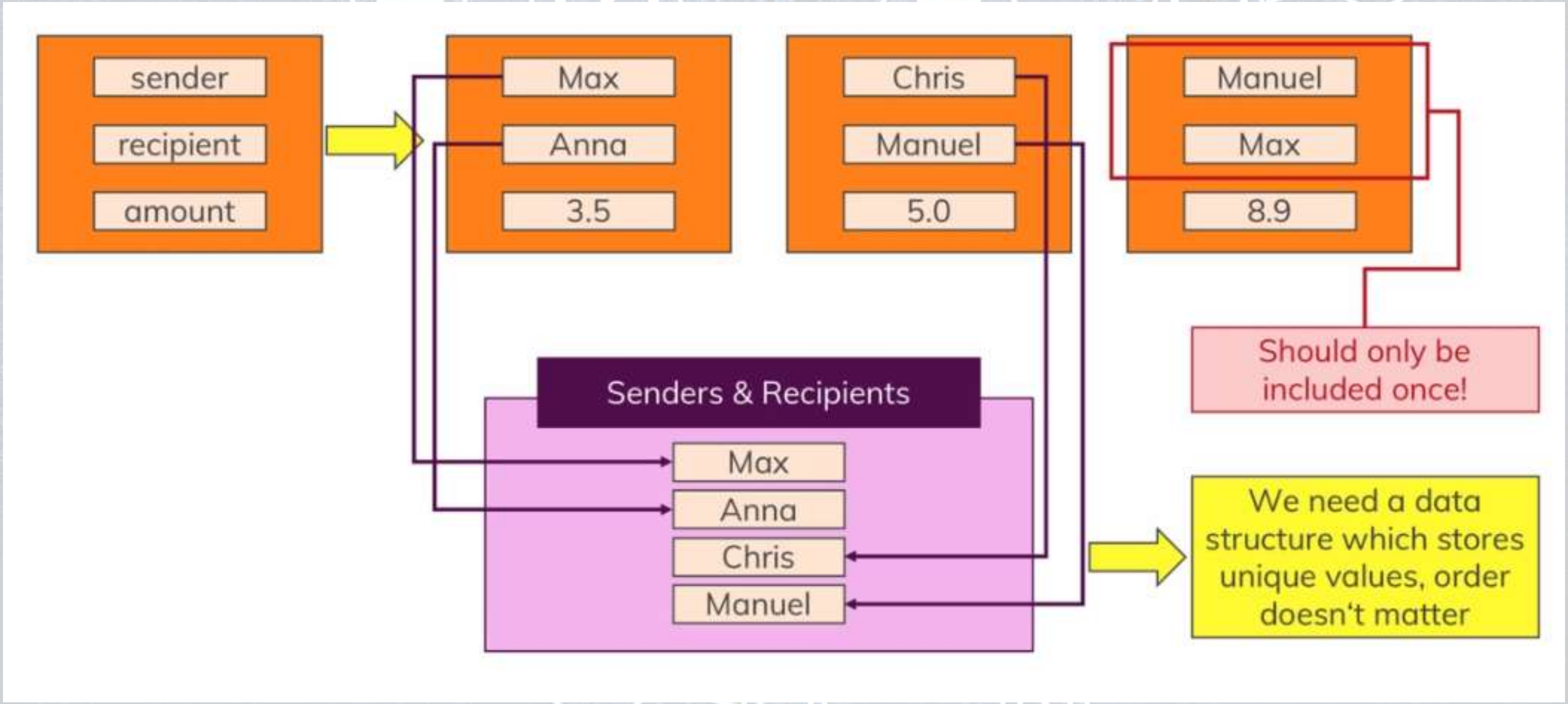


- ✓ Chain of Data → Basic Implementation
- ✓ Mine new Blocks → With User Interface!
- ✓ Block Hashing → Basic Implementation
- ✓ Analyze & Verify Chain → Basic Implementation
- ☐ Transactions
- ☐ Store Chain to Disk
- ☐ Node Network
- ☐ Share Data, Resolve Conflicts
- ☐ Wallets

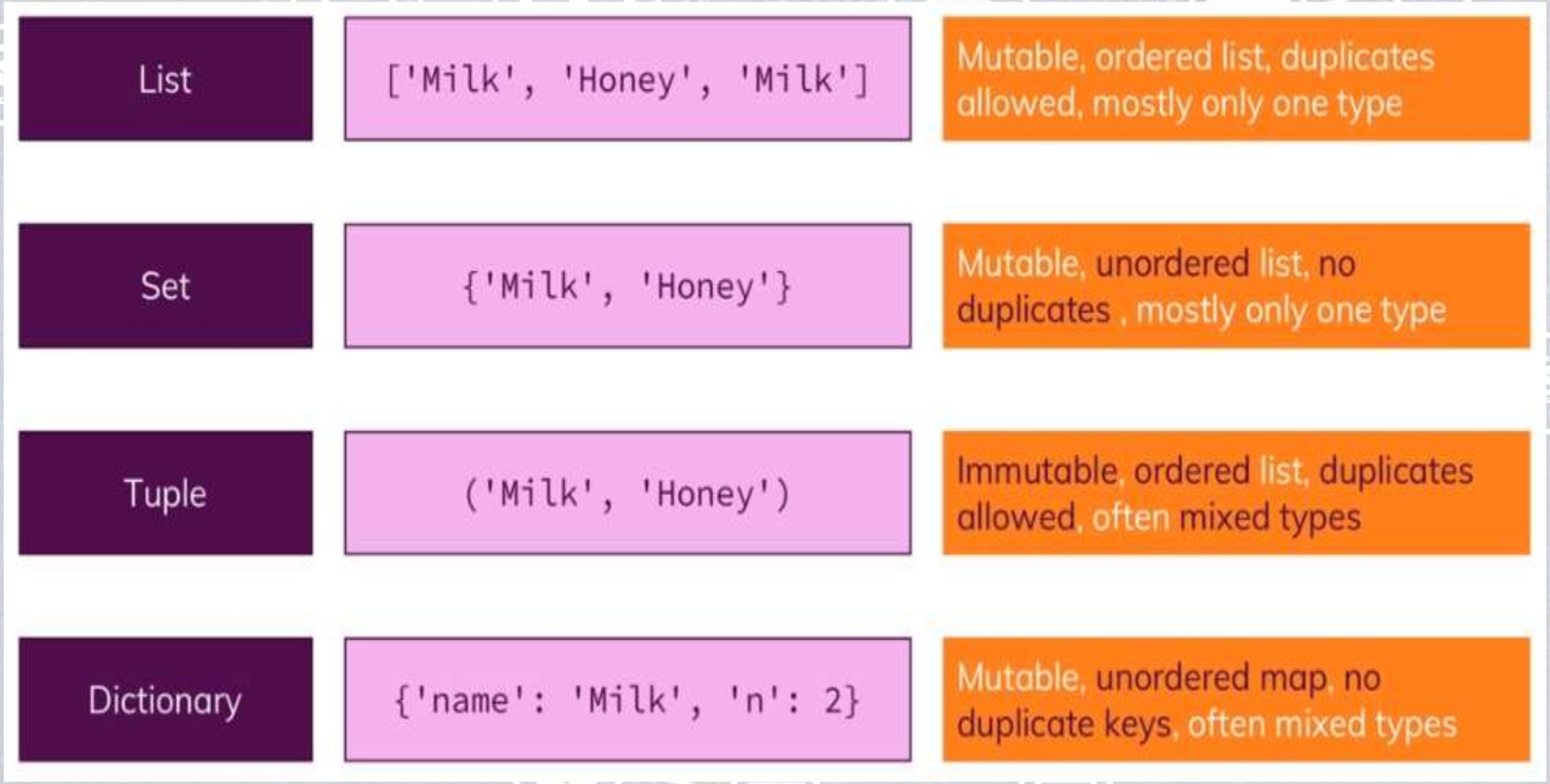
# :Block Chain With Detailed Transaction:


We need a data structure which stores a mutable (editable) list of values, order doesn't matter
sender
recipient
amount
We need a data structure which stores key-value pairs, order doesn't matter
We need a data structure which stores a mutable (editable) list of values, order does matter
Mining a new Block

# Block Chain [Managing Users]

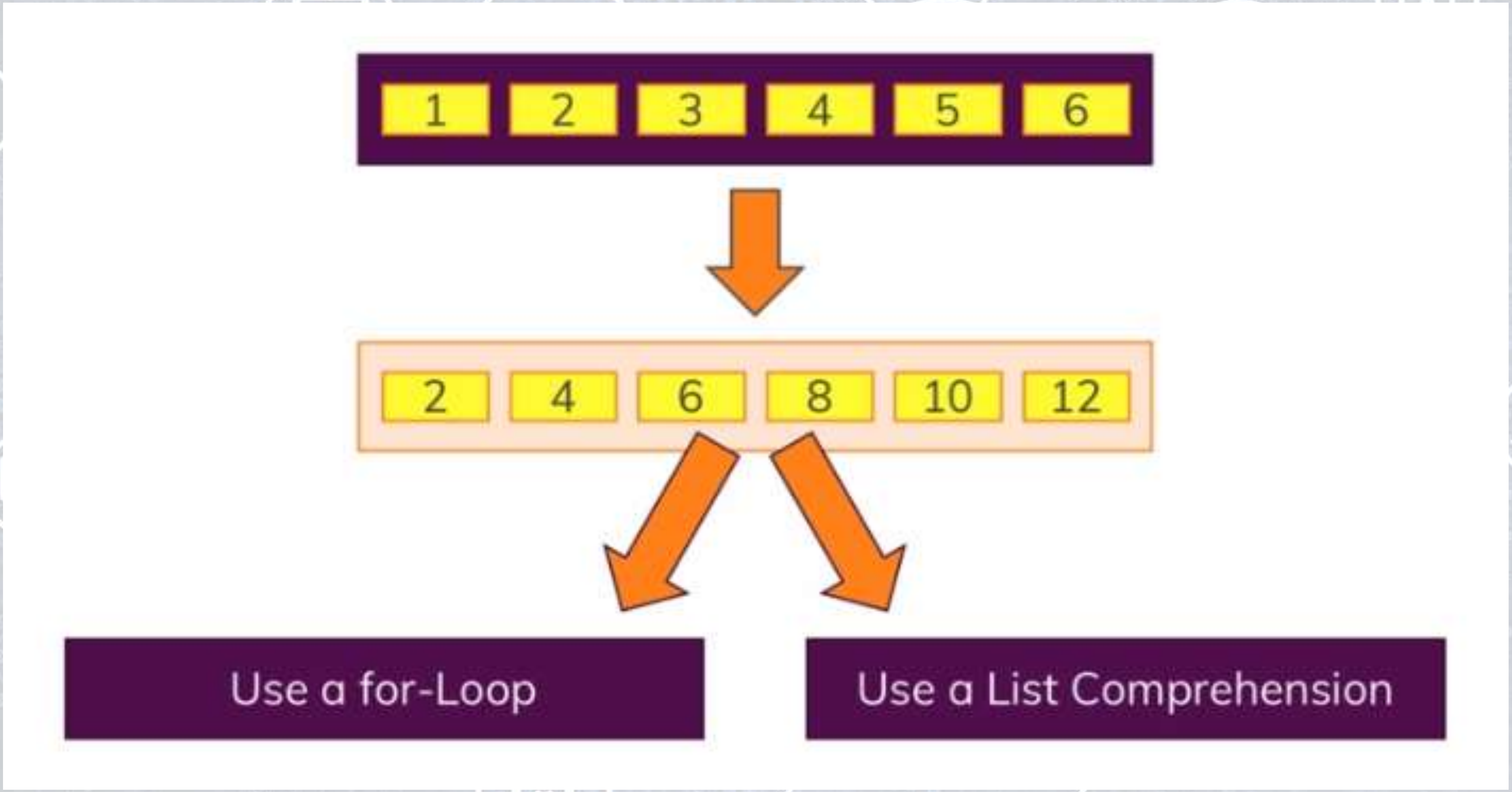

sender
recipient
amount
Max
Anna
3.5
Chris
Manuel
5.0
Manuel
Max
8.9
Senders & Recipients
Max
Anna
Chris
Manuel
Should only be included once!
We need a data structure which stores unique values, order doesn't matter

## :Data Structures:



| | | |
|---|---|---|
| List | `['Milk', 'Honey', 'Milk']` | Mutable, ordered list, duplicates allowed, mostly only one type |
| Set | `{'Milk', 'Honey'}` | Mutable, unordered list, no duplicates , mostly only one type |
| Tuple | `('Milk', 'Honey')` | Immutable, ordered list, duplicates allowed, often mixed types |
| Dictionary | `{'name': 'Milk', 'n': 2}` | Mutable, unordered map, no duplicate keys, often mixed types |

# List Comprehensions



## :Coming Up Next:

- Improving Block Chain Validation.
- Rewarding Miners.
- Verifying Transaction.
- Etc…

Thank you