Data Structures

in TypeScript

Michael Dowden *(he/him)*
@mrdowden

What will we cover?

JavaScript

- " Record "
- Array
- Set
- Map

Constructed

- List
- Stack
- Queue
- Tree

Andromeda Galactic Solutions


Architecting CSS
The Programmer's Guide to Effective Stylesheets
Martine Dowden
Michael Dowden
Approachable Accessibility
Planning for Success
Martine Dowden
Michael Dowden
Programming Languages
ABC++
inedo
Andromeda
Galactic Solutions
ignition
exePark
LORE LINK
Michael Dowden
he / him
Google Developer Expert
2025 Web
MVP
Microsoft
Most Valuable Professional

Building without Structure

Photo by Alexandra Vasilyeva on Unsplash

Building with Structure

Photo by Alphacolor on Unsplash

Benefits of Structure

Performance

Photo by serjan midili on Unsplash

Memory Management

Photo by Robert Anasch on Unsplash

Maintenance

Photo by Toon Lambrechts on Unsplash

```
let user_1_first_name = 'Karel'
let user_1_last_name = 'Sanchis'
let user_1_email = 'ksanchis@networkadvertising.org'
let user_1_address_street_1 = '12 Roxbury Crossing'
let user_1_address_street_2 = 'PO Box 97501'
let user_1_address_city = 'Otacilio Costa'
let user_1_address_postal_code = '88540-000'
let user_1_address_country = 'Brazil'
let user_2_first_name = 'Tim'
let user_2_last_name = 'Pandey'
let user_2_email = 'tpandey1@smugmug.com'
let user_2_address_street_1 = '6 Elmside Avenue'
let user_2_address_street_2 = ''
let user_2_address_city = 'Pathum Thani'
let user_2_address_postal_code = '90110'
let user_2_address_country = 'Thailand'
```

Typing

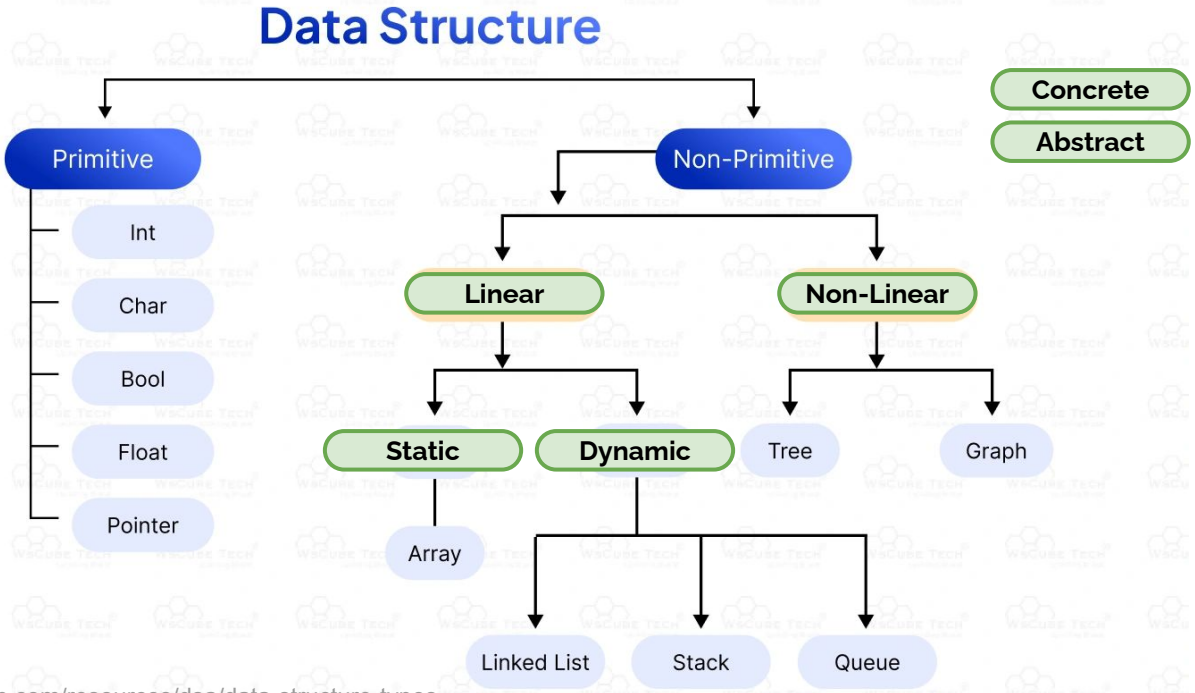
```
function join(a, b) {
  return a + b
}
```

```
function join(
  a: string, b: string
): string { return a + b }
```

```
function join(
  a: number, b: number
): number { return a + b }
```


join('x
s any,
)

Classification



https://www.wscubetech.com/resources/dsa/data-structure-types

Struct / Record

Non-Linear | Static | Concrete

Photo by Edgar Chaparro on Unsplash

Object as Record

```
type Product = {
  sku: string
  upc: string
  name: string
  price: number
  description: string
  weight: number
  dimensions: {
    depth: number
    height: number
    width: number
  }
}
```

```
const product: Product = {
  sku: '',
  upc: '',
  name: '',
  price: 13.99,
  description: '',
  weight: 12.3,
  dimensions: {
    depth: 12.0,
    height: 6.25,
    width: 4.55
  }
}
```

TypeScript Record Type

```ts
type CatName = "miffy" | "boris" | "mordred";

interface CatInfo {
  age: number;
  breed: string;
}

const cats: Record<CatName, CatInfo> = {
  miffy: { age: 10, breed: "Persian" },
  boris: { age: 5, breed: "Maine Coon" },
  mordred: { age: 16, breed: "British Shorthair" },
};
```

Generics

```
type Square = {
  width: number
  height: number
}

type Circle = {
  radius: number
}

type Border<T> = {
  weight: 'light' | 'normal' | 'bold'
  color: string
  shape: T
}
```

```
let b: Border<Square> = {
  weight: 'bold',
  color: '#000000',
  shape: {
    width: 24,
    height: 36
  }
}
```

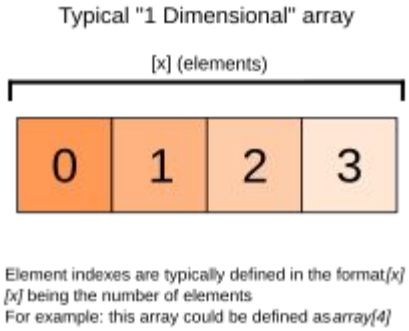
🧰 Array

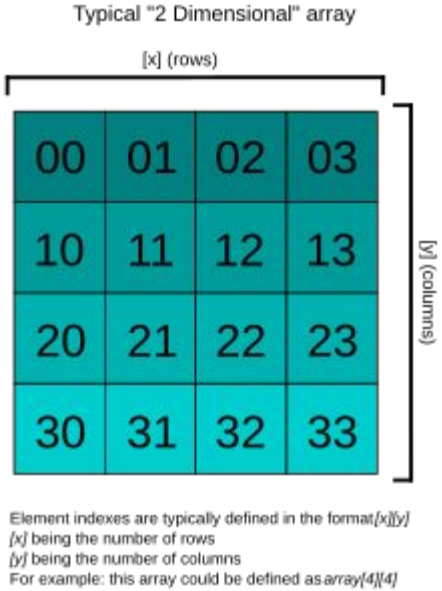
Linear | Static | Concrete

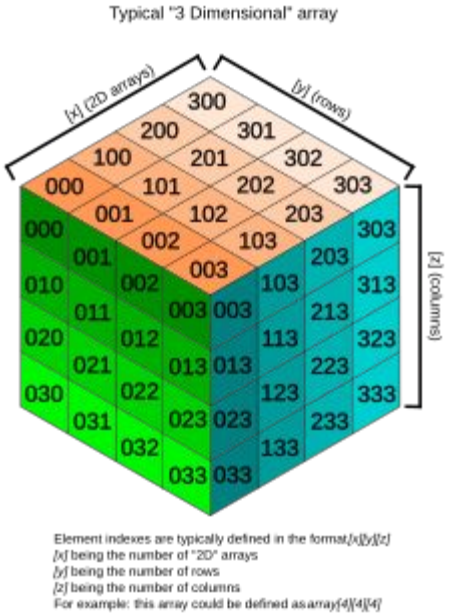
32	false	"London"	3.14159	{ id: 123 }
0	1	2	3	4

- Sequential, iterable collection of values
- Can contain any values
- Homogeneous / Complex

Array Dimensions







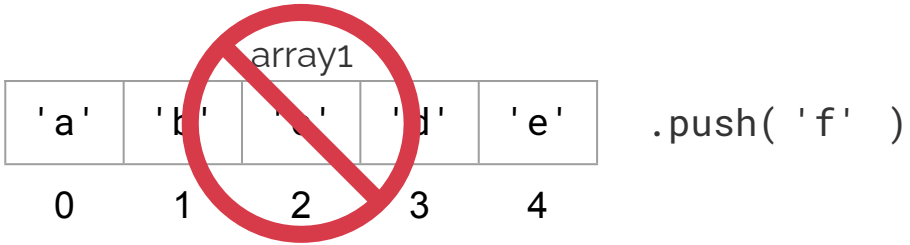
https://en.wikipedia.org/wiki/Array_(data_structure)

Andromeda Galactic Solutions

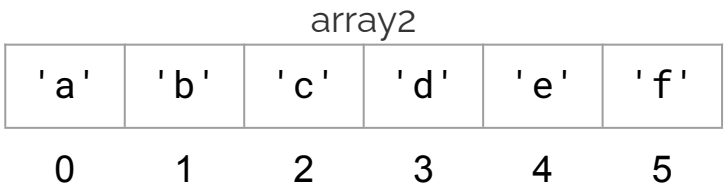
Arrays in TypeScript

```
let array: string[] = ['a', 'b', 'c']

let complexArray: Array<String | Number> = ['a', 2, 'x', 2.3]

let array2d = [['x','o','x'],['x','x','o'],['o','o','x']]


let list: number[] = []

list.push(3.14)
```

Array Memory

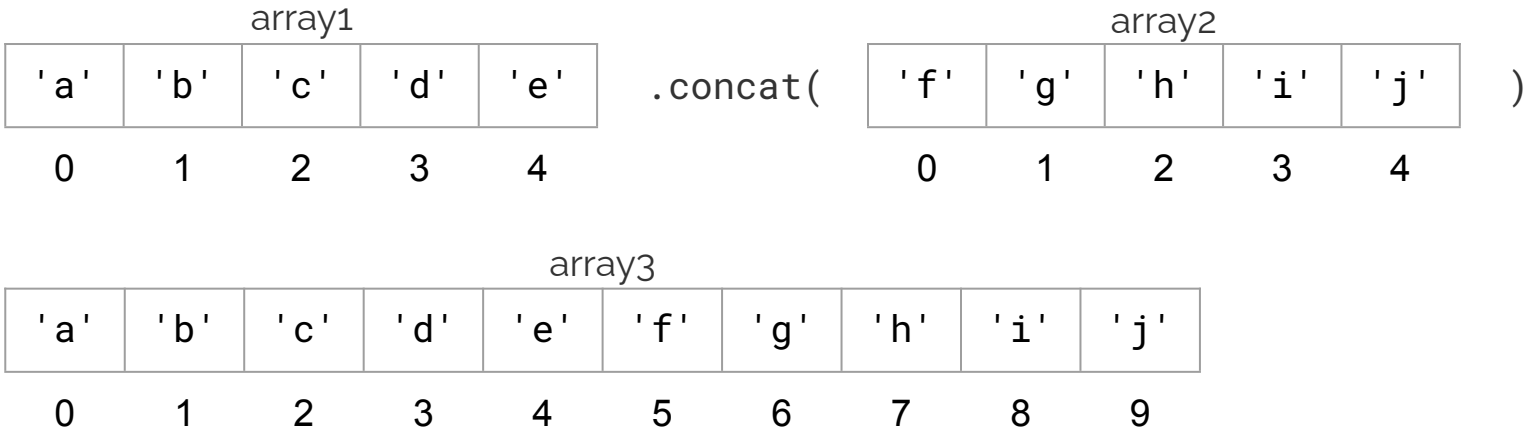



```
.push( 'f' )
```



Andromeda Galactic Solutions

Array Memory



Andromeda Galactic Solutions

Iterable

```
// for loop
for (let i = 0; i < a.length; i++) { console.log(a[i]) }

// for-in
for (const elem in a) { console.log(elem) }

// for-of
for (const elem of a) { console.log(elem) }

// foreach
a.forEach(elem => { console.log(elem) })
```

Map & Filter

```
const a = [1, 32, 7, 16, 9, 11]

a.every(elem => elem > 0) // true

a.some(elem => elem > 30) // true

a.filter(elem => elem % 2 == 0) // [32, 16]

a.map((n, i) => ({ i, n, even: n % 2 == 0 }))
// [{ i:0, n:1, even:false}, { i:1, n:32, even:true }, …]
```

Reduce

```
const employees = [{id:23, name:'Frank'}, {id:73, name:'Sally'}]
const byId = employees.reduce((acc, emp) => {
  acc[emp.id] = emp
  return acc
}, {})
byId[23].name === 'Frank' // VS
employees.find(emp => emp.id == 23)?.name === 'Frank'
```

Linear | Static | Abstract

🧰 Tuple

Ordered and finite list of elements, similar to an array

Notated: `(1, 2, 3)` (instead of the array which is `[1, 2, 3]`)

- Monad (1-tuple) = `(1)`
- Pair (2-tuple) = `(1, 2)`

Multi-type such as `('pi', 3.14, 'e', 2.718)`

Order matters, so `(1, 2, 3) !== (2, 1, 3)`

Tuples in TypeScript

```
let response = [404, 'Not Found']
let tuple: [string, number, string, number] = ['pi', 3.14, 'e', 2.718]

type Phone = [number, number, number, string?]
let phones: Phone[] = [
  [213, 213, 1242],
  [222, 555, 1259, 'x123'],
]
```

🧰 Set

Linear | Static | Concrete

- Easily convert between Set and Array
- Distinct/unique entries
- Iterate over entries
- Mathematical Set operations like difference, intersection, and union

```
const s1 = new Set([1, 2, 2, 3])
const a = [...s1]
console.log( JSON.stringify(a) )
// 1, 2, 3

let s2 = new Set()
s2.add(1)
s2.add(3)
```

🧰 Map

Linear | Dynamic | Concrete

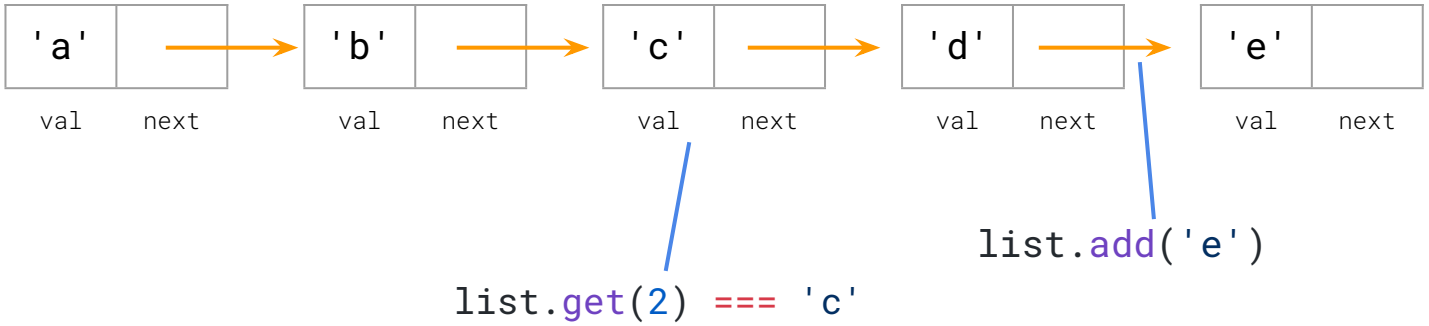
- Key-value store
- Remembers insertion order
- Contains no default keys
- No prototype
- Keys of any type
- Iterable
- Tracks size

```
let contacts = new Map()

contacts.set('Mike',
  'michael@email.org')

contacts.set('Jill',
  'jill@email.org')

contacts.size // 2

contacts.get('Mike')

//'michael@email.org'
```

Andromeda Galactic Solutions

Linear | Dynamic | Concrete

🧰 Linked List



- Arrays and Linked Lists are concrete implementations of Lists
- Memory doesn't need to be contiguous
- Methods: add, insert, delete, get, etc.

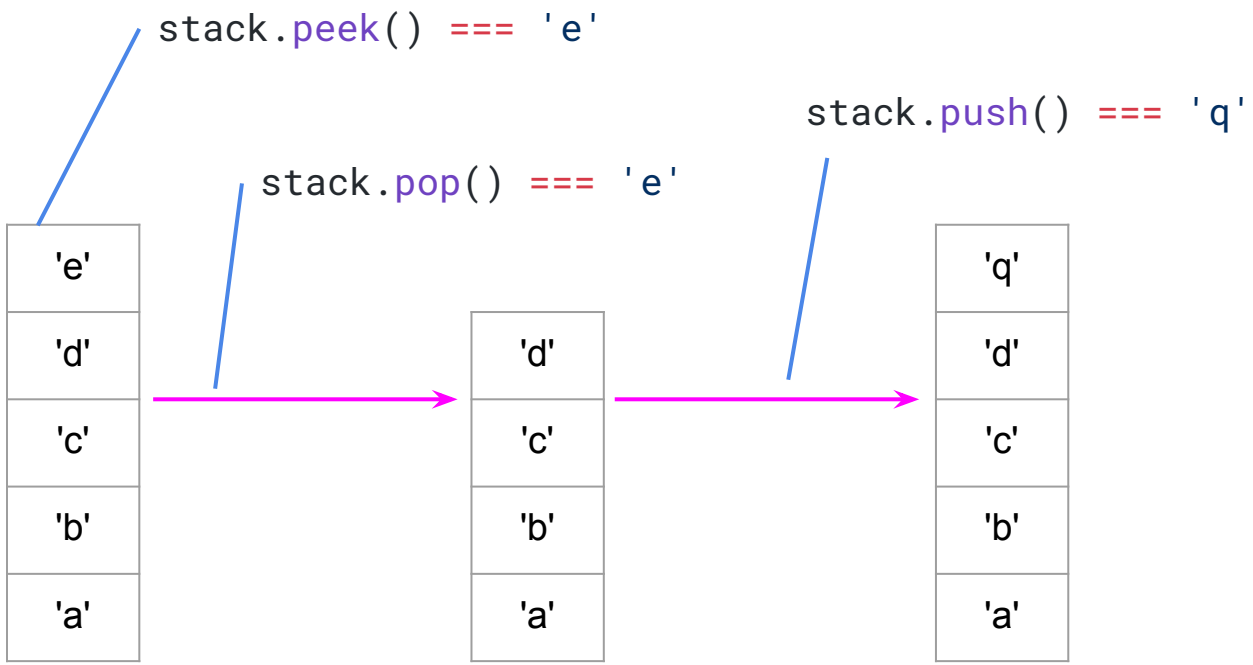
Linked Lists in TypeScript

```ts
type Node<T> = { val: T, next: Node<T> }
class LinkedList<T> {
  size = 0
  first: Node<T>
  add(val: T) {
    const node = { val, next: null }
    if(!this.first) this.first = node
    else {
      let x = this.first
      while(x.next != null) x = x.next
      x.next = node
    }
    this.size++
  }
  get(n: number): T {
    let node = this.first
    for(let i = 0; i < n; i++) {
      node = node?.next
    }
    return node?.val
  }
}
```

```ts
const list = new
LinkedList<string>()
list.add('Mark')
list.add('Mary')
list.add('Sally')
console.log( list.get(1) ) // Mary
```

🧰 Stack

Linear | Dynamic | Abstract



Andromeda Galactic Solutions

Array-backed Stack in TypeScript

```
const stack: string[] = []

stack.push('Mark')

stack.push('Mary')

console.log(stack) // Mark, Mary

stack.pop() === 'Mary'
```

Stacks in TypeScript

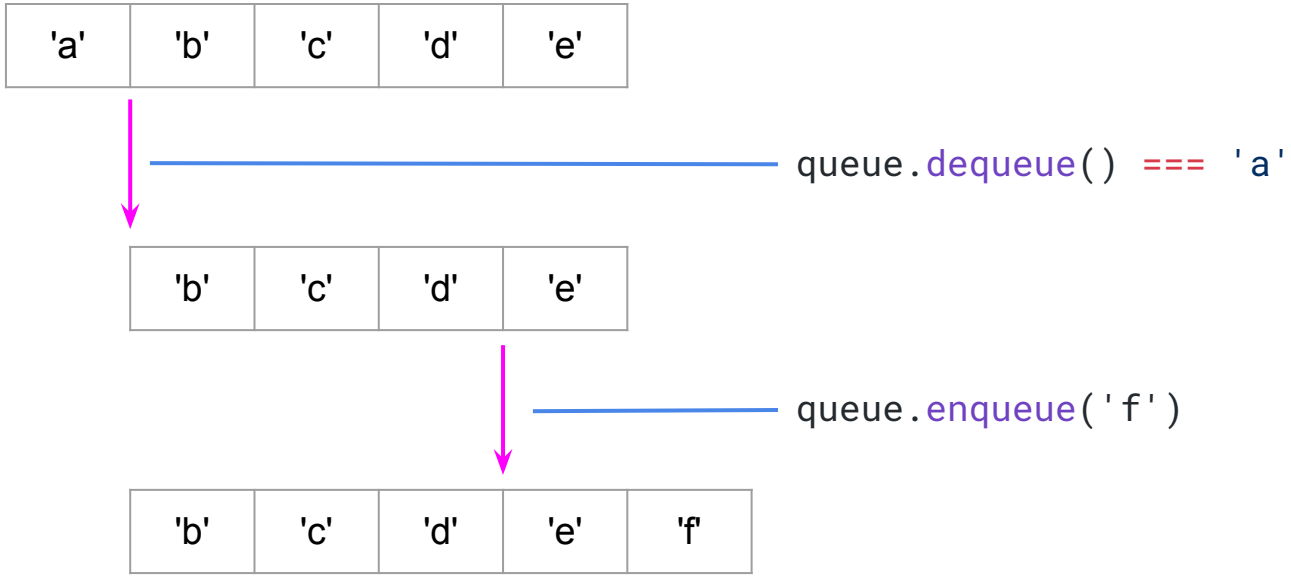
```
type Node<T> = {
  val: T, next: Node<T>
}
class Stack<T> {
  top: Node<T>
  push(val: T) {
    this.top = {val, next:this.top}
  }
  pop(): T {
    const node = this.top
    this.top = node?.next
    return node?.val
  }
  peek(): T { return this.top?.val }
}
```

```
const stack = new Stack<string>()
stack.push('Mark')
stack.push('Mary')
console.log( stack.peek() ) // Mary
stack.pop() === 'Mary'
console.log( stack.peek() ) // Mark
```

Andromeda Galactic Solutions

🧰 Queue

Linear | Dynamic | Abstract



Array-backed Queue in TypeScript

```
const queue: string[] = []
queue.push('Mark') // enqueue
queue.push('Mary') // enqueue
console.log(queue) // Mark, Mary
queue.shift() === 'Mark' // dequeue
console.log(queue) // Mary
```

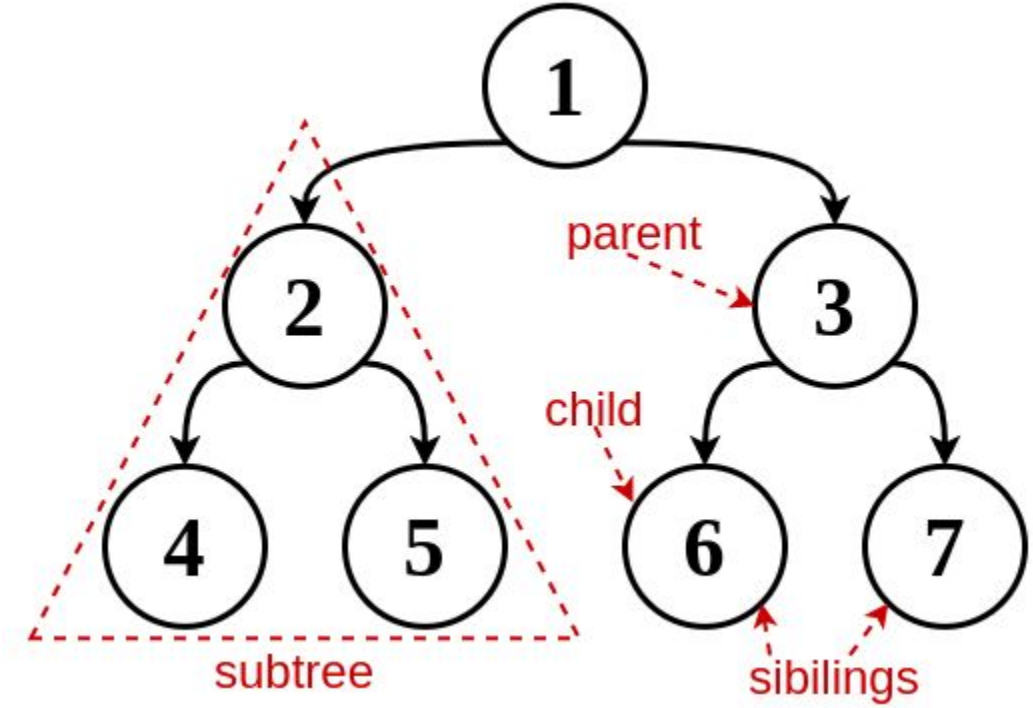
Queues in TypeScript

```ts
type Node<T> = { val: T, next: Node<T> }
class Queue<T> {
  private top: Node<T>
  private end: Node<T>
  size = 0
  enqueue(val: T) {
    const node = { val, next: null }
    if(!this.end) { this.top = node }
    else { this.end.next = node }
    this.end = node; this.size++
  }
  dequeue(): T {
    const node = this.top
    this.top = node?.next
    if(!this.top) this.end = null
    this.size--; return node?.val
  }
}
```

```ts
const queue = new Queue<string>()
queue.enqueue('Mark')
queue.enqueue('Mary')

console.log( queue.dequeue() )
// Mark

console.log( queue.size ) // 1
```

🧰 Tree

Non-Linear | Dynamic | Abstract



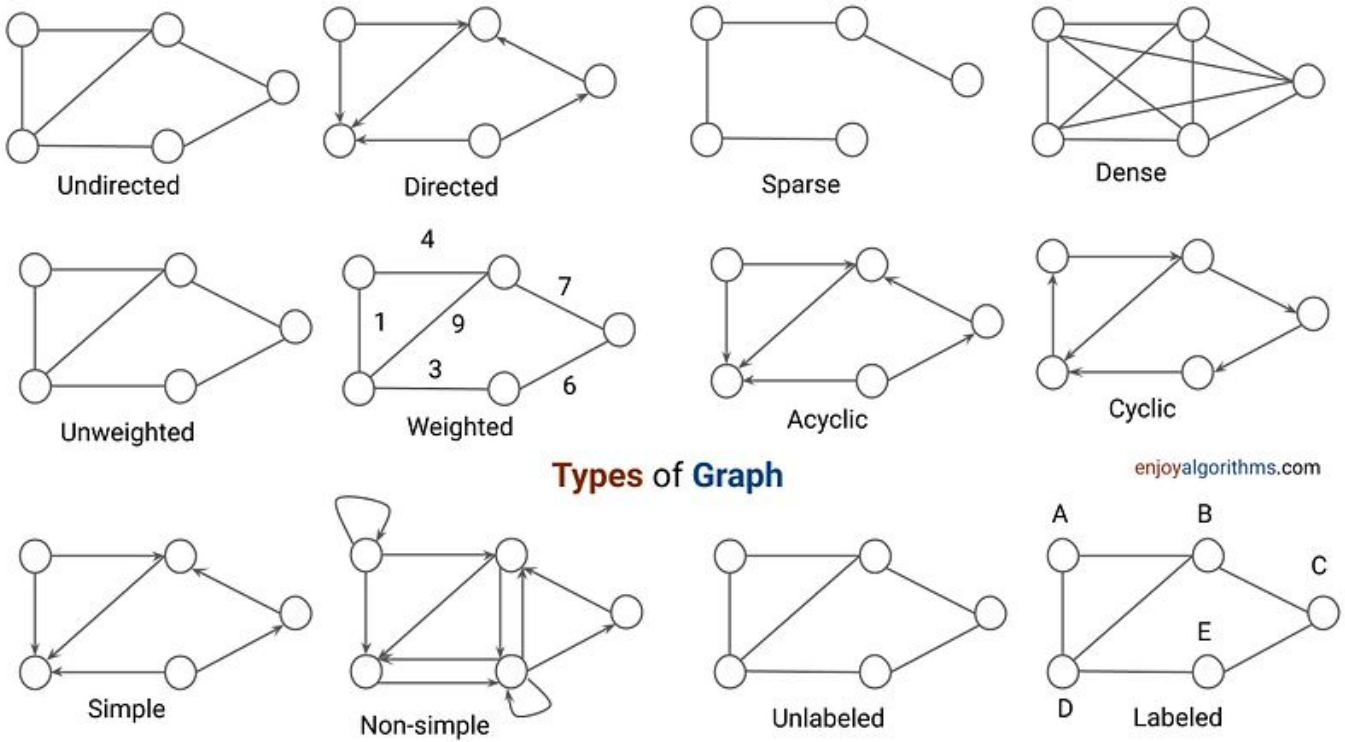
https://ricardoborges.dev/blog/data-structures-in-typescript-binary-search-tree

Trees in TypeScript

```
class TreeNode<T> {
  private data: T;
  private children: TreeNode<T>[];

  constructor(data: T) {
    this.data = data;
    this.children = [];
  }

  addChild(child: TreeNode<T>): void {
    this.children.push(child);
  }
}
```

Graph

Non-Linear | Dynamic | Abstract



https://www.enjoyalgorithms.com/blog/types-of-graphs-in-data-structures

Graphs in TypeScript

```
class GraphNode<T> {
  private data: T;
  private neighbors: GraphNode<T>[];

  constructor(data: T) {
    this.data = data;
    this.neighbors = [];
  }

  addNeighbor(child: GraphNode<T>): void {
    this.neighbors.push(child);
  }
}
```

Graphs in TypeScript

```
class Graph<T> {
  nodes: GraphNode<T>[];
  constructor() {
    this.nodes = [];
  }
  addNode(value: any) {
    const node = new GraphNode<T>(value);
    this.nodes.push(node);
  }
  addEdge(source: GraphNode<T>, destination: GraphNode<T>) {
    source.addNeighbor(destination);
    destination.addNeighbor(source);
  }
}
```

Summary

Record	Non-Linear	Static	Concrete
Array	Linear	Static	Concrete
Tuple	Linear	Static	Abstract
Set	Linear	Static	Concrete
Map	Linear	Dynamic	Concrete
Stack	Linear	Dynamic	Abstract
Queue	Linear	Dynamic	Abstract
Linked List	Linear	Dynamic	Concrete
Tree	Non-Linear	Dynamic	Abstract
Graph	Non-Linear	Dynamic	Abstract

Andromeda Galactic Solutions

We ask for
your feedback!

PLEASE VOTE NOW!

devmio

Michael Dowden *(he/him)*

@mrdowden

mrdowden.com

Google Developer Expert

2025 Web

MVP Microsoft® Most Valuable Professional

Architecting CSS

The Programmer's Guide to Effective Stylesheets

Martine Dowden
Michael Dowden

Apress®

Approachable Accessibility

Planning for Success

Martine Dowden
Michael Dowden

Apress®

Programming Languages

ABC++

References

- https://www.linkedin.com/pulse/mastering-arrays-comprehensive-guide-typescript-muhammad-rizwan-l2ybe/
- https://www.techtarget.com/whatis/definition/tuple
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Memory_management
- https://www.typescriptlang.org/docs/handbook/utility-types.html#recordkeys-type
- https://www.typescriptlang.org/docs/handbook/variable-declarations.html#re-declarations-and-shadowing
- https://en.wikipedia.org/wiki/Tuple
- https://developer.mozilla.org/en-US/docs/Web/JavaScript
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/Set
- https://stackfull.dev/graph-data-structure-in-typescript