Intro to "Advanced" TypeScript

`TypeScript` = `static types` + `JavaScript`

`static types` = `TypeScript` - `JavaScript` 🤔

```
const me = {
  nickname:  "Adi",
  github:    "9at8",
  instagram: "9at8",
  discord:   "9at8#8019",
  twitter:   "9at8_",
}
```

Types as Sets

```
boolean = {true, false}
number = {-1, 2, -3, 3.14, 42, ...}
string = {"oof", "ooff", "oofff", "ooffff", ...}

interface Person {
  firstName: string
  lastName: string
}

Person  = Forall fname in string ->
            Forall lname in string ->
              ({ firstName: fname, lastName: lname })
```

```ts
interface Box {
  id: number
  item: any
}

const box: Box = { id: 42, item: "i am a string" }

box.item
//  ^^^^ oh no! the compiler no longer knows that it clearly is a string!
```

Generics

```
interface Box<Item> {
  id: number
  item: Item
}

const stringBox: Box<string> = { id: 42, item: "i am a string" }
const numberBox: Box<number> = { id: 24, item: 1 }

stringBox.item  // <- the compiler knows that item is a string
numberBox.item  // <- the compiler knows that item is a number
```

```ts
interface Box<Item extends string | number> {
  id: number
  item: Item
}

const stringBox: Box<string> = { id: 42, item: "i am a string" }
const numberBox: Box<number> = { id: 24, item: 1 }
const boolBox: Box<boolean> = { id: -42, item: true }
//                  ^^^^^^^
// Type 'boolean' does not satisfy the constraint 'string | number'.
```

```ts
type Unbox<TBox> = TBox extends Box<infer Item>
  ? Item extends Box<any>
    ? Unbox<Item>
    : Item
  : never

const strBox = { id: 42, item: "i am a string" }
const strBoxBox = { id: 24, item: strBox }

type InnerType1 = Unbox<typeof strBox>
//   ^^^^^^^^^^ string
type InnerType2 = Unbox<typeof strBoxBox>
//   ^^^^^^^^^^ string
```

`meta-typing` package

https://github.com/ronami/meta-typing

Incrementing and decrementing numbers

```
type IncTable = { 0: 1; 1: 2; 2: 3; 3: 4; 4: 5; 5: 6; 6: 7; 7: 8; 8: 9; 9: 10 };

type DecTable = { 10: 9; 9: 8; 8: 7; 7: 6; 6: 5; 5: 4; 4: 3; 3: 2; 2: 1; 1: 0 };

type Inc<T extends number> = T extends keyof IncTable
  ? IncTable[T]
  : never;

type Dec<T extends number> = T extends keyof DecTable
  ? DecTable[T]
  : never;

type Two = Inc<1>
type One = Dec<Inc<1>>
```

Head and tail of a list

```
type Head<T extends Array<any>> =
  T extends [any, ...Array<any>]
    ? T['0']
    : never;

type Tail<T extends Array<any>> =
  T extends [any, ...infer Rest]
    ? Rest
    : never;
```

Less than or equal to

```
// `true` if `A` is smaller than or equals to `B`, `false` otherwise
type Lte<A extends number, B extends number> =
  IsNever<A> extends true
  ? IsNever<B> extends true
    ? true
    : false
  : Lte<Inc<A>, Inc<B>>;

/*
  Lte<8, 9> = Lte<9, 10> = Lte<10, never> = Lte<never, never> = true
  Lte<9, 8> = Lte<10, 9> = Lte<never, 10> = false
 */
```

THE BIG REVEAL

Insertion sort

```
type Insert<
  N extends number,
  R extends number[]
> = R extends []
  ? [N]
  : Lte<N, Head<R>> extends true
    ? [N, ...R]
    : [
        Head<R>,
        ...Insert<N, Cast<Tail<R>, number[]>>
      ]
```

```
type InsertionSort<
  T extends number[],
  R extends number[] = []
> = T extends []
  ? R
  : InsertionSort<
      Cast<Tail<T>, number[]>,
      Insert<Head<T>, R>
    >
```

Playground Link

- Sorting
  - Quick-sort
  - Merge-sort
  - Insertion-sort
- Puzzles
  - N-Queens
  - Maze-solving
  - Binary trees
  - Square Matrix Rotation
  - Towers of Hanoi

(Probably a good idea to not use this stuff during interviews)

Slides