If Statements
Most narrowing comes from expressions inside if statements, where different type operators narrow inside the new scope

```typescript
if (typeof input === "string") {
    input // string
}
if (input instanceof Array) {
    input // number[]
}
```

Expressions
Narrowing also occurs on the same line as code, when doing boolean operations

```typescript
const input = getUserInput()
input // string | number
const inputLength = (typeof input === "string" && input.length) || input
    // input: string
```

Assignment
Narrowing types using ‘as const’
Subfields in objects are treated as though they can be mutated, and during assignment the type will be ‘widened’ to a non-literal version. The prefix ‘as const’ locks all types to their literal versions.

```typescript
const data1 = {
    name: "Zagreus",
} as const
const data2 = {
    name: "Zagreus",
} as const
```

Type Guards
A function with a return type describing the CFA change for a new scope when it is true.

```typescript
function isErrorResponse(obj: Response): obj is APIErrorResponse {
    return obj instanceof APIErrorResponse
}
```

Assertion Functions
A function describing CFA changes affectting the current scope, because it throws instead of returning false.

```typescript
function assertResponse(obj: any): asserts obj is SuccessResponse {
    if (!obj instanceof SuccessResponse) {
        throw new Error("Not a success!")
    }
}
```

Discriminated Unions
Type Responses =
```
|
| { status: 200, data: any } |
| { status: 301, to: string } |
| { status: 400, error: Error } |
```

```
All members of the union have the same property name, CFA can discriminate on that.
```

Usage
```
const response = getResponse()
response // Responses
switch(response.status) {
    case 200: return response.data
    case 301: return redirect(response.to)
    case 400: return response.error
}
```

Guard Functions
```
if (obj instanceof APIErrorResponse) {
    return true
}
```

Usage
```
const response = getResponse()
response // APIErrorResponse
```

```
if (isErrorResponse(response)) {
    response // APIErrorResponse
}
```

Tracks through related variables
```
const response = getResponse()
const isSuccessResponse = res instanceof SuccessResponse
```

Assertion Functions change the current scope or throw
```
assertResponse(res)
res // SuccessResponse
```

Re-assignment updates types
```
let data: string | number = ...
data // string | number
data = "Hello"
data // string
```