TypeScript
Cheat Sheet

Control Flow Analysis

Key points

CFA nearly always takes a union and reduces the number of types inside the union based on logic in your code.

Most of the time CFA works inside natural JavaScript boolean logic, but there are ways to define your own functions which affect how TypeScript narrows types.

If Statements

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

typeof (for primitives)

```
const input = getUserInput()
input // string | number

if (typeof input === "string") {
   input // string
}
```

"property" in object (for objects)

```
const input = getUserInput()
input // string | { error: ... }

if ("error" in input) {
   input // { error: ... }
}
```

instanceof (for classes)

```
const input = getUserInput()
input // number | number[]

if (input instanceof Array) {
   input // number[]
}
```

type-guard functions (for anything)

```
const input = getUserInput()
input // number | number[]

if (Array.isArray(input)) {
   input // number[]
}
```

Expressions

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

```
const input = getUserInput()
input // string | number

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

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
}
```

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.

```
const data1 = {              typeof data1 = {
   name: "Zagreus"   »          name: string
}                            }

const data2 = {              typeof data2 = {
   name: "Zagreus"   »          name: "Zagreus"
} as const                   }
```

Tracks through related variables

```
const response = getResponse()
const isSuccessResponse
     = res instanceof SuccessResponse

if (isSuccessResponse)
  res.data // SuccessResponse
```

Re-assignment updates types

```
let data: string | number = ...
data // string | number
data = "Hello"
data // string
```

Type Guards

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

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

Return type position describes what the assertion is

Usage

```
const response = getResponse()
response // Response | APIErrorResponse

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

Assertion Functions

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

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

Usage

```
const res = getResponse():
res // SuccessResponse | ErrorResponse

assertResponse(res)

res // SuccessResponse
```

Assertion functions change the *current* scope or throw