



Better TypeScript Types



Rob Howard
@damncabbage
<http://robhoward.id.au>





Audience Interaction Time



0) Why

**Write down my assumptions in a
way the computer understands,
then have it tell me when I'm
wrong.**

Dear Kate,

Here's to the crazy ones. The mad ones. The troublemakers. The round pegs in square holes. The ones who see things others don't and who don't care. The ones who are not fond of rules. And they have no respect for the status quo. You can quote them, disagree with them, glorify or vilify them. About the only thing you can't do is ignore them. Because they change things.

They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.

Take care.
John Appleseed

Dear Kate,

Here's to the crazy ones. The mad ones. The troublemakers. The round pegs in square holes. The ones who see things others don't and who don't care. The ones who are not fond of rules. And they have no respect for the status quo. You can quote them, disagree with them, glorify or vilify them. About the only thing you can't do is ignore them. Because they change things.

They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.

Take care.
John Appleseed

Dear Kate,

Here's to the crazy ones. The mad ones. The troublemakers. The round pegs in square holes. The ones who see things others don't and who don't care. The ones who are not fond of rules. And they have no respect for the status quo. You can quote them, disagree with them, glorify or vilify them. About the only thing you can't do is ignore them. Because they change things.

They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.

Take care.
John Appleseed

Dear Kate,

Here's to the crazy ones. The mad ones. The troublemakers. The round pegs in square holes. The ones who see things others don't and who don't care. The ones who are not fond of rules. And they have no respect for the status quo. You can quote them, disagree with them, glorify or vilify them. About the only thing you can't do is ignore them. Because they change things.

They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.

Take care.
John Appleseed

Dear Kate,

Here's to the crazy ones. The mad ones. The troublemakers. The round pegs in square holes. The ones who see things others don't and who don't care. The ones who are not fond of rules. And they have no respect for the status quo. You can quote them, disagree with them, glorify or vilify them. About the only thing you can't do is ignore them. Because they change things.

They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.

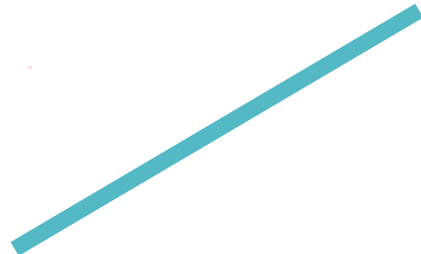
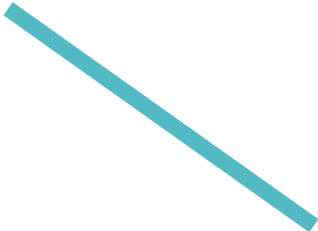
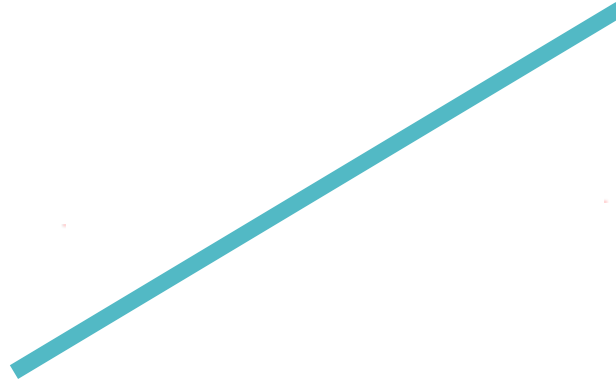
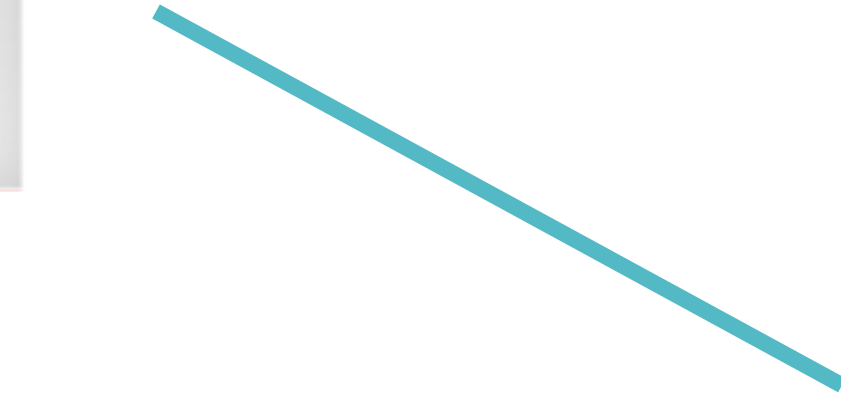
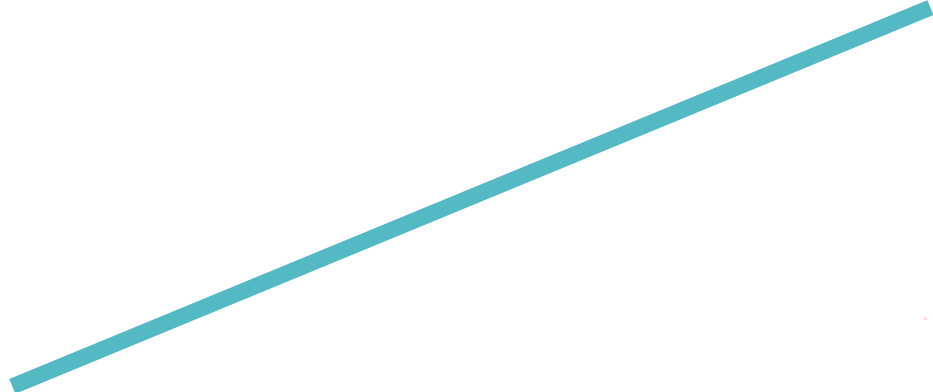
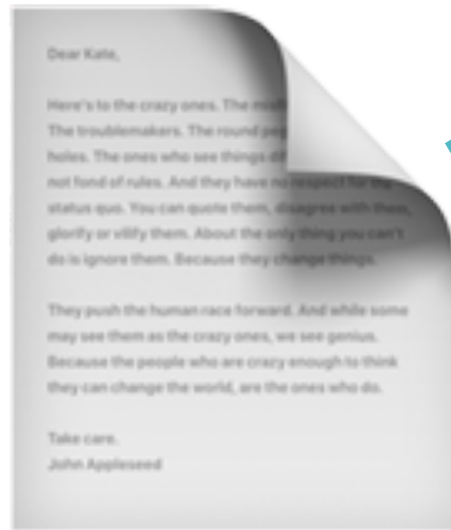
Take care.
John Appleseed

Dear Kate,

Here's to the crazy ones. The mad ones. The troublemakers. The round pegs in square holes. The ones who see things others don't and who don't care. The ones who are not fond of rules. And they have no respect for the status quo. You can quote them, disagree with them, glorify or vilify them. About the only thing you can't do is ignore them. Because they change things.

They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.

Take care.
John Appleseed





Dear Kate,

Here's to the crazy ones. The mad ones. The troublemakers. The round pegs in square holes. The ones who see things others don't and they have not respect for the status quo. You can quote them, disagree with them, glorify or vilify them. About the only thing you can't do is ignore them. Because they change things.

They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.

Take care,
John Appleseed

Dear Kate,

Here's to the crazy ones. The mad ones. The troublemakers. The round pegs in square holes. The ones who see things others don't and they have not respect for the status quo. You can quote them, disagree with them, glorify or vilify them. About the only thing you can't do is ignore them. Because they change things.

They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.

Take care,
John Appleseed

Dear Kate,

Here's to the crazy ones. The mad ones. The troublemakers. The round pegs in square holes. The ones who see things others don't and they have not respect for the status quo. You can quote them, disagree with them, glorify or vilify them. About the only thing you can't do is ignore them. Because they change things.

They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.

Take care,
John Appleseed

Dear Kate,

Here's to the crazy ones. The mad ones. The troublemakers. The round pegs in square holes. The ones who see things others don't and they have not respect for the status quo. You can quote them, disagree with them, glorify or vilify them. About the only thing you can't do is ignore them. Because they change things.

They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.

Take care,
John Appleseed

Dear Kate,

Here's to the crazy ones. The mad ones. The troublemakers. The round pegs in square holes. The ones who see things others don't and they have not respect for the status quo. You can quote them, disagree with them, glorify or vilify them. About the only thing you can't do is ignore them. Because they change things.

They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.

Take care,
John Appleseed

Dear Kate,

Here's to the crazy ones. The mad ones. The troublemakers. The round pegs in square holes. The ones who see things others don't and they have not respect for the status quo. You can quote them, disagree with them, glorify or vilify them. About the only thing you can't do is ignore them. Because they change things.

They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.

Take care,
John Appleseed



1) Possibilities



1.1) Keep only what's needed

```
function icon(type: string): string {  
    ...  
}
```



```
// CSS class name for icon  
function icon(type: string): string {  
    return `icon icon-${type}`;  
}
```

```
// CSS class name for icon  
function icon(type: string): string {  
    return `icon icon-${type}`;  
}
```

```
type Icon = 'spinner' | 'save';
```

```
function icon(type: Icon): string {  
    ...  
}
```



```
type Icon = 'spinner' | 'save';
```

```
function icon(type: Icon): string {  
    ...  
}
```

string

string

" "

string

" "

"spinner"

string

" "

"spinner"

"Hello, world!"

string

" "

"spinner"

"Hello, world!"

"不忍人之心"

string

" "

"spinner"

"Hello, world!"

"不忍人之心"

"ACT I
SCENE I. Athens. The palace of
THESEUS.

Enter THESEUS, HIPPOLYTA,
PHILOSTRATE, and Attendants

..."

"spinner" | "save"

"spinner" | "save"

"spinner"

"spinner" | "save"

"spinner"

"save"

Possibilities:

$\lim n \rightarrow \infty \dots?$

Possibilities:

2



**1.2) Don't spread a
choice across
multiple values.**

```
type Props = {  
  success?: boolean,  
  warning?: boolean,  
  error?:   boolean,  
  ...  
};
```

```
function Notification(props: Props): {  
  ...  
}
```

```
<Notification  
  error={true}  
>  
  This is bad.  
</Notification>
```



```
Notification({  
  error: true,  
  children: [  
    "This is bad.",  
  ]  
})
```

```
<Notification  
  error={true}  
>  
  This is bad.  
</Notification>
```

```
<Notification  
  error={true}  
>  
  This is bad.  
</Notification>
```

```
<Notification  
  success={true}  
>
```

This is good.

```
</Notification>
```

```
type Props = {  
  success?: boolean,  
  warning?: boolean,  
  error?:   boolean,  
  ...  
};
```

```
function Notification(props: Props): {  
  ...  
}
```

```
<Notification  
  success={true}  
  error={true}  
>
```

This is ... ???

```
</Notification>
```



```
<Notification  
  success={true}  
  error={true}  
>
```

This is ... ???

```
</Notification>
```

```
type Props = {  
  success?: boolean,  
  warning?: boolean,  
  error?:   boolean,  
  ...  
};
```

```
function Notification(props: Props): {  
  ...  
}
```

```
type Props = {  
  success?: boolean,  
  warning?: boolean,  
  error?:   boolean,  
  ...  
};
```

```
function Notification(props: Props): {  
  ...  
}
```

```
type Props = {  
  type: 'success' | 'warning' | 'error',  
  ...  
};
```

```
function Notification(props: Props): {  
  ...  
}
```

```
<Notification  
  type="success"  
>
```

This is good.

```
</Notification>
```



```
<Notification  
  type="success"  
>  
  This is good.  
</Notification>
```

Possibilities:

**(true,true,true + true,true,false +
true,false,false + ...)**

8

Possibilities:

3



1.3) Things that change separately

```
type Props = {  
  type: 'success' | 'warning' | 'error',  
  dismissable: boolean,  
};
```

```
function Notification(props: Props): {  
  ...  
}
```

```
type Props = {  
  type: 'success' | 'warning' | 'error',  
  dismissable: boolean,  
};
```

```
function Notification(props: Props): {  
  ...  
}
```


The background of the slide features a low-angle, upward-looking view of a modern building's facade. The building has a grid-like pattern of windows and architectural elements. A semi-transparent teal overlay covers the entire image, creating a uniform color scheme. The text is centered in the upper half of the image.

**1.4) Convenient types
are tempting but can
leave gaps.**

```
type Props = {  
  items: Array<Item>,  
};
```

```
function Carousel(props: Props): {  
  ...  
}
```

```
<Carousel  
  items=[...]  
>
```

```
<Carousel  
  items=[]  
>
```

```
type Props = {  
  items: Array<Item>,  
};
```

```
function Carousel(props: Props): {  
  ...  
}
```

```
type Props = {  
  itemsBefore: Array<Item>  
  currentItem: Item,  
  itemsAfter: Array<Item>,  
};
```

```
function Carousel(props: Props): {  
  ...  
}
```



```
type Props = {  
  itemsBefore: Array<Item>  
  currentItem: Item,  
  itemsAfter: Array<Item>,  
};
```

```
function Carousel(props: Props): {  
  ...  
}
```

```
<Carousel  
  itemsBefore=[]  
  item=...  
  itemsAfter=[]  
>
```

```
<Carousel  
  itemsBefore=[]  
  item=...  
  itemsAfter=[]  
>
```



1.5) Combining Unions and values that change separately

```
function Example(props: Props): {  
  const [isLoading, setLoading] =  
    useState(false);  
  const [isError, setError] =  
    useState(false);  
  const [data, setData] =  
    useState<string | null>(null);  
  
  ...  
}
```

```
type State = {  
  isLoading: boolean,  
  isError: boolean,  
  data: null | string,  
}
```

```
type State =  
  | { type: 'loading' }  
  | { type: 'error' }  
  | { type: 'success',  
      data: null | string  
    }
```


Possibilities:

$$(1 + 1 + (1 \times 2))$$

Possibilities:

4

The background of the slide features a photograph of a modern, multi-story building with a grid-like facade of windows and balconies. The entire image is covered with a semi-transparent blue overlay, which serves as a backdrop for the white text.

Sums and Products

Possibilities

A single possibility is a
'1'
(eg. true, 5, "hi", undefined)

Unions are Sums (addition)

Union types represent a group of 'either-or' possibilities.

Total states: add all the possibilities together.

```
type Icon =  
  | "spinner"  
  | "save"
```

```
type Icon =  
  | "spinner"  
  | "save"
```




```
type Icon =  
  | "spinner"  
  | "save"
```

← 1

← 1

```
type Icon =  
  | "spinner"      ← 1  
  | "save"          ← 1
```

Possible states for Icon

```
type Icon =  
  | "spinner"    ← 1  
  | "save"       ← 1
```

Possible states for Icon

= 1 + 1

```
type Icon =  
  | "spinner"    ← 1  
  | "save"       ← 1
```

Possible states for Icon

= 1 + 1

= 2

```
type boolean =  
  | true      ← 1  
  | false     ← 1
```

Possible states for boolean

= 1 + 1

= 2

type undefined =
| undefined ← 1

Possible states for undefined

= 1

```
type Icon = 'spinner' | 'save';
```

```
function icon(type: Icon): string {  
    ...  
}
```

**Values that change
independently are
Products (multiplication)**

Total states: multiply the groups
of possibilities together.


```
type Props = {  
  success?: boolean,  
  warning?: boolean,  
  error?:   boolean,  
  ...  
};
```

```
function Notification(props: Props): {  
  ...  
}
```

```
type Props = {  
  success?: boolean,  
  warning?: boolean,  
  error?:   boolean,  
  ...  
};
```

```
type Props = {  
  success?: boolean,  
  warning?: boolean,  
  error?:    boolean,  
  ...  
};
```

```
success: true,  
warning: false,  
error:   false,
```

```
type Props = {  
  success?: boolean,  
  warning?: boolean,  
  error?:    boolean,  
  ...  
};
```

```
success: false,  
warning: true,  
error:   true,
```

```
type Props = {  
  success?: boolean,  
  warning?: boolean,  
  error?:   boolean,  
  ...  
};
```

```
success: false,  
warning: false,  
error:   false,
```

```
type Props = {  
  success?: boolean,  
  warning?: boolean,  
  error?:   boolean,  
  ...  
};
```

```
type Props = {  
  success?: boolean,  
  warning?: boolean,  
  error?:    boolean,  
  ...  
};
```

← **true + false**

```
type Props = {  
  success?: boolean,  
  warning?: boolean,  
  error?:   boolean,  
  ...  
};
```

← 1 + 1


```
type Props = {  
  success?: boolean,  
  warning?: boolean,  
  error?:   boolean,  
  ...  
};
```

← 1 + 1
← 1 + 1

```
type Props = {  
  success?: boolean,  
  warning?: boolean,  
  error?:   boolean,  
  ...  
};
```

← 1 + 1

← 1 + 1

← 1 + 1

```
type Props = {  
  success?: boolean,    ← 1 + 1  
  warning?: boolean,    ← 1 + 1  
  error?:    boolean,   ← 1 + 1  
  ...  
};
```

Possible states for Icon

```
type Props = {  
  success?: boolean,    ← 1 + 1  
  warning?: boolean,    ← 1 + 1  
  error?:    boolean,    ← 1 + 1  
  ...  
};
```

Possible states for Icon

$$= (1 + 1) \times (1 + 1) \times (1 + 1)$$

```
type Props = {  
  success?: boolean,    ← 1 + 1  
  warning?: boolean,    ← 1 + 1  
  error?:    boolean,    ← 1 + 1  
  ...  
};
```

Possible states for Icon

$$= (1 + 1) \times (1 + 1) \times (1 + 1)$$

$$= 8$$



2) Exhaustivity

```
type Icon = 'spinner' | 'save';  
  
function icon(type: Icon): string {  
    ...  
}
```

```
type Icon = 'spinner' | 'save';
```

```
function icon(type: Icon): string {  
  switch (type) {  
    case 'spinner': ...  
    case 'save': ...  
  }  
}
```



```
type Icon = 'spinner' | 'save' | 'ok';
```

```
function icon(type: Icon): string {  
  switch (type) {  
    case 'spinner': ...  
    case 'save': ...  
  }  
}
```

```
type Icon = 'spinner' | 'save' | 'ok';
```

```
function icon(type: Icon): string {  
  switch (type) {  
    case 'spinner': ...  
    case 'save': ...  
  }  
}
```

```
type Icon = 'spinner' | 'save' | 'ok';
```

```
function icon(type: Icon): string {  
  switch (type) {  
    case 'spinner': ...  
    case 'save': ...  
  }  
}
```

Function lacks ending
return statement and
return type does not
include 'undefined'.

```
type Icon = 'spinner' | 'save' | 'ok';
```

```
function icon(type: Icon): string {  
  switch (type) {  
    case 'spinner': ...  
    case 'save': ...  
    case 'ok': ...  
  }  
}
```

```
type Icon = 'spinner' | 'save' | 'ok';
```

```
function icon(type: Icon): string {  
  switch (type) {  
    case 'spinner': ...  
    case 'save': ...  
    case 'ok': ...  
  }  
}
```





Dear Kate,

Here's to the crazy ones. The mad ones. The troublemakers. The round pegs in square holes. The ones who see things others don't and they have not respect for the status quo. You can quote them, disagree with them, glorify or vilify them. About the only thing you can't do is ignore them. Because they change things.

They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.

Take care,
John Appleseed

Dear Kate,

Here's to the crazy ones. The mad ones. The troublemakers. The round pegs in square holes. The ones who see things others don't and they have not respect for the status quo. You can quote them, disagree with them, glorify or vilify them. About the only thing you can't do is ignore them. Because they change things.

They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.

Take care,
John Appleseed

Dear Kate,

Here's to the crazy ones. The mad ones. The troublemakers. The round pegs in square holes. The ones who see things others don't and they have not respect for the status quo. You can quote them, disagree with them, glorify or vilify them. About the only thing you can't do is ignore them. Because they change things.

They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.

Take care,
John Appleseed

Dear Kate,

Here's to the crazy ones. The mad ones. The troublemakers. The round pegs in square holes. The ones who see things others don't and they have not respect for the status quo. You can quote them, disagree with them, glorify or vilify them. About the only thing you can't do is ignore them. Because they change things.

They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.

Take care,
John Appleseed

Dear Kate,

Here's to the crazy ones. The mad ones. The troublemakers. The round pegs in square holes. The ones who see things others don't and they have not respect for the status quo. You can quote them, disagree with them, glorify or vilify them. About the only thing you can't do is ignore them. Because they change things.

They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.

Take care,
John Appleseed

Dear Kate,

Here's to the crazy ones. The mad ones. The troublemakers. The round pegs in square holes. The ones who see things others don't and they have not respect for the status quo. You can quote them, disagree with them, glorify or vilify them. About the only thing you can't do is ignore them. Because they change things.

They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.

Take care,
John Appleseed



```
type Example = boolean | string;
```

```
function example(x: Example): number {  
  switch (typeof x) {  
    case 'boolean':  
      ...  
    case 'string':  
      ...  
  }  
}
```

```
type Example = boolean | string;
```

```
function example(x: Example): number {  
  switch (typeof x) {  
    case 'boolean':  
      ...  
    case 'string':  
      ...  
  }  
}
```



```
type Example = boolean | string;
```

```
function example(x: Example): number {  
  switch (typeof x) {  
    case 'boolean':  
      ...  
    case 'string':  
      ...  
  }  
}
```

← x is a boolean in here

```
type Example = boolean | string;
```

```
function example(x: Example): number {  
  switch (typeof x) {  
    case 'boolean':  
      ...  
    case 'string':  
      ...  
  }  
}
```

← x is a boolean in here

← x is a string in here

```
type Example = boolean | string;
```

```
function example(x: Example): number {  
  switch (typeof x) {  
    case 'boolean':  
      ...  
    case 'string':  
      ...  
  }  
}
```

← switch() satisfies
TS's exhaustivity checker.

← x is a boolean in here

← x is a string in here

```
type Example = boolean | string;
```

```
function example(x: Example): number {  
  switch (typeof x) {  
    case 'boolean':  
      ...  
    case 'string':  
      ...  
  }  
}
```

← switch() satisfies
TS's exhaustivity checker.

← x is a boolean in here

← x is a string in here



```
type Example = Array<string> | string;
```

```
function example(x: Example): number {  
  if (Array.isArray(x)) {  
    ...  
  } else if (typeof x === 'string') {  
    ...  
  }  
}
```

```
type Example = Array<string> | string;
```

```
function example(x: Example): number {  
  if (Array.isArray(x)) {  
    ...  
  } else if (typeof x === 'string') {  
    ...  
  }  
}
```

```
type Example = Array<string> | string;
```

```
function example(x: Example): number {  
  if (Array.isArray(x)) {  
    ...  
  } else if (typeof x === 'string') {  
    ...  
  }  
}
```

Function lacks ending
return statement and
return type does not
include 'undefined'.

```
type Example = Array<string> | string;
```

```
function example(x: Example): number {  
    if (Array.isArray(x)) {  
        ...  
    }  
    switch (typeof x) {  
        case 'string':  
            ...  
    }  
}
```



```
type Example = Array<string> | string;
```

```
function example(x: Example): number {  
    if (Array.isArray(x)) {  
        ...  
    }  
    switch (typeof x) {  
        case 'string':  
            ...  
        }  
    }  
}
```

```
type Example = Array<string> | string;
```

```
function example(x: Example): number {  
    if (Array.isArray(x)) {  
        ... ← x is an array in here  
    }  
    switch (typeof x) {  
        case 'string':  
            ...  
        }  
    }  
}
```

```
type Example = Array<string> | string;
```

```
function example(x: Example): number {  
  if (Array.isArray(x)) {  
    ...  
  }  
  switch (typeof x) {  
    case 'string':  
      ...  
  }  
}
```

```
type Example = Array<string> | string;
```

```
function example(x: Example): number {  
  if (Array.isArray(x)) {
```

```
    ...
```

```
  }
```

```
  switch (typeof x) {
```

```
    case 'string':
```

```
      ...
```

```
    }
```

```
  }
```

← x is a string in here.

```
type Example = Array<string> | string;
```

```
function example(x: Example): number {  
    if (Array.isArray(x)) {  
        ...  
    }  
    switch (typeof x) {  
        case 'string':  
            ...  
        }  
    }  
}
```

← switch() satisfies
TS's exhaustivity checker.
← x is a string in here.

```
type Example = Array<string> | string;
```

```
function example(x: Example): number {  
    if (Array.isArray(x)) {  
        ...  
    }  
    switch (typeof x) {  
        case 'string':  
            ...  
        }  
    }  
}
```

```
type Example = Array<string> | string;
```

```
function example(x: Example): number {  
    if (Array.isArray(x)) {  
        ...  
    }  
    switch (typeof x) {  
        case 'string':  
            ...  
    }  
}
```



```
type Example = Array<string> | string;
```

```
function example(x: Example): number {  
  if (Array.isArray(x)) {  
    ...  
  }  
  switch (typeof x) {  
    case 'string':  
      ...  
  }  
}
```




```
class Foo { ... }; class Bar { ... };  
type Example = Foo | Bar;  
  
function example(x: Example): number {  
    if (x instanceof Foo) {  
        ...  
    } else if (x instanceof Bar) {  
        ...  
    }  
}
```

```
class Foo { ... }; class Bar { ... };  
type Example = Foo | Bar;
```

```
function example(x: Example): number {  
    if (x instanceof Foo) {                ← Can't use switch().  
        ...  
    } else if (x instanceof Bar) {  
        ...  
    }  
}
```

```
class Foo { ... }; class Bar { ... };  
type Example = Foo | Bar;
```

```
function example(x: Example): number {  
    if (x instanceof Foo) { ← Can't use switch().  
        ...  
    } else if (x instanceof Bar) { ←  
        ... Can't use switch().  
    }  
}
```

```
class Foo { ... }; class Bar { ... };  
type Example = Foo | Bar;
```

```
function example(x: Example): number {  
    if (x instanceof Foo) {  
        ...  
    } else if (x instanceof Bar) {  
        ...  
    }  
}
```

```
class Foo { ... }; class Bar { ... };  
type Example = Foo | Bar;
```

```
function example(x: Example): number {  
    if (x instanceof Foo) {  
        ...  
    } else if (x instanceof Bar) {  
        ...  
    }  
}
```

Function lacks ending
return statement and
return type does not
include 'undefined'.

```
class Foo { ... }; class Bar { ... };  
type Example = Foo | Bar;
```

```
function example(x: Example): number {  
  if (x instanceof Foo) {  
    ...  
  } else if (x instanceof Bar) {  
    ...  
  }  
  return notExhaustive();  
}
```

```
function notExhaustive(): never {  
    throw new Error('Not exhaustive');  
}  
function example(x: Example): number {  
    if (x instanceof Foo) {  
        ...  
    } else if (x instanceof Bar) {  
        ...  
    }  
    return notExhaustive();  
}
```

```
function notExhaustive(): never {  
    throw new Error('Not exhaustive');  
}  
function example(x: Example): number {  
    if (x instanceof Foo) {  
        ...  
    } else if (x instanceof Bar) {  
        ...  
    }  
    return notExhaustive();  
}
```




```
function notExhaustive(): never {  
    throw new Error('Not exhaustive');  
}  
function example(x: Example): number {  
    if (x instanceof Foo) {  
        ...  
    } else if (x instanceof Bar) {  
        ...  
    }  
    return notExhaustive();  
}
```



Type Guards & Exhaustivity

(It's a rabbit-hole; I'm still figuring this out myself.)

<https://basarat.gitbooks.io/typescript/docs/types/typeGuard.html>

<https://www.typescriptlang.org/docs/handbook/advanced-types.html#exhaustiveness-checking>

(Appears to be out of date / inaccurate.)

3) Intentionally-Different Types

**If you mean
something different,
use a different type.**

number

Row ID

Kilogram

Count

number

Length (cm)

Currency (AUD)

Currency (JP Yen)

```
async function saveUser(  
  id:    string,  
  email: string  
) : Promise<User> {  
  ...  
}  
// ...  
await saveUser(id, "a@b.c");
```

```
async function saveUser(  
  id:    string,  
  email: string  
) : Promise<User> {  
  ...  
}  
// ...  
await saveUser(id, "a@b.c");
```



```
async function saveUser(  
  id:    string,  
  email: string  
): Promise<User> {  
  ...  
}  
// ...  
await saveUser(id, "a@b.c");
```

```
async function saveUser(  
  id:    string,  
  email: string  
) : Promise<User> {  
  ...  
}  
// ...  
await saveUser("a@b.c", id);
```

```
async function saveUser(  
  id:    string,  
  email: string  
) : Promise<User> {  
  ...  
}  
// ...  
await saveUser(id, "a@b.c");
```



Method #1: Context

```
async function saveUser(  
  id:    string,  
  email: string  
) : Promise<User> {  
  ...  
}  
// ...  
await saveUser(id, "a@b.c");
```

```
async function saveUser({ id, email }: {  
  id:    string,  
  email: string  
}): Promise<User> {  
  ...  
}  
// ...  
await saveUser({ id, email: "a@b.c"});
```

```
async function saveUser({ id, email }: {  
  id:      string,  
  email: string  
}): Promise<User> {  
  ...  
}  
// ...  
await saveUser({ id, email: "a@b.c"});
```

```
type User = {  
  id:    string,  
  email: string,  
};
```

```
async function saveUser(  
  { id, email }: User  
): Promise<User> {  
  ...  
}
```



```
type User = {  
  id:      string,  
  email: string,  
};
```

```
async function saveUser(  
  { id, email }: User  
): Promise<User> {  
  ...  
}
```



```
async function saveUser(  
  { id, email }: User  
): Promise<User> {  
  ...  
}
```

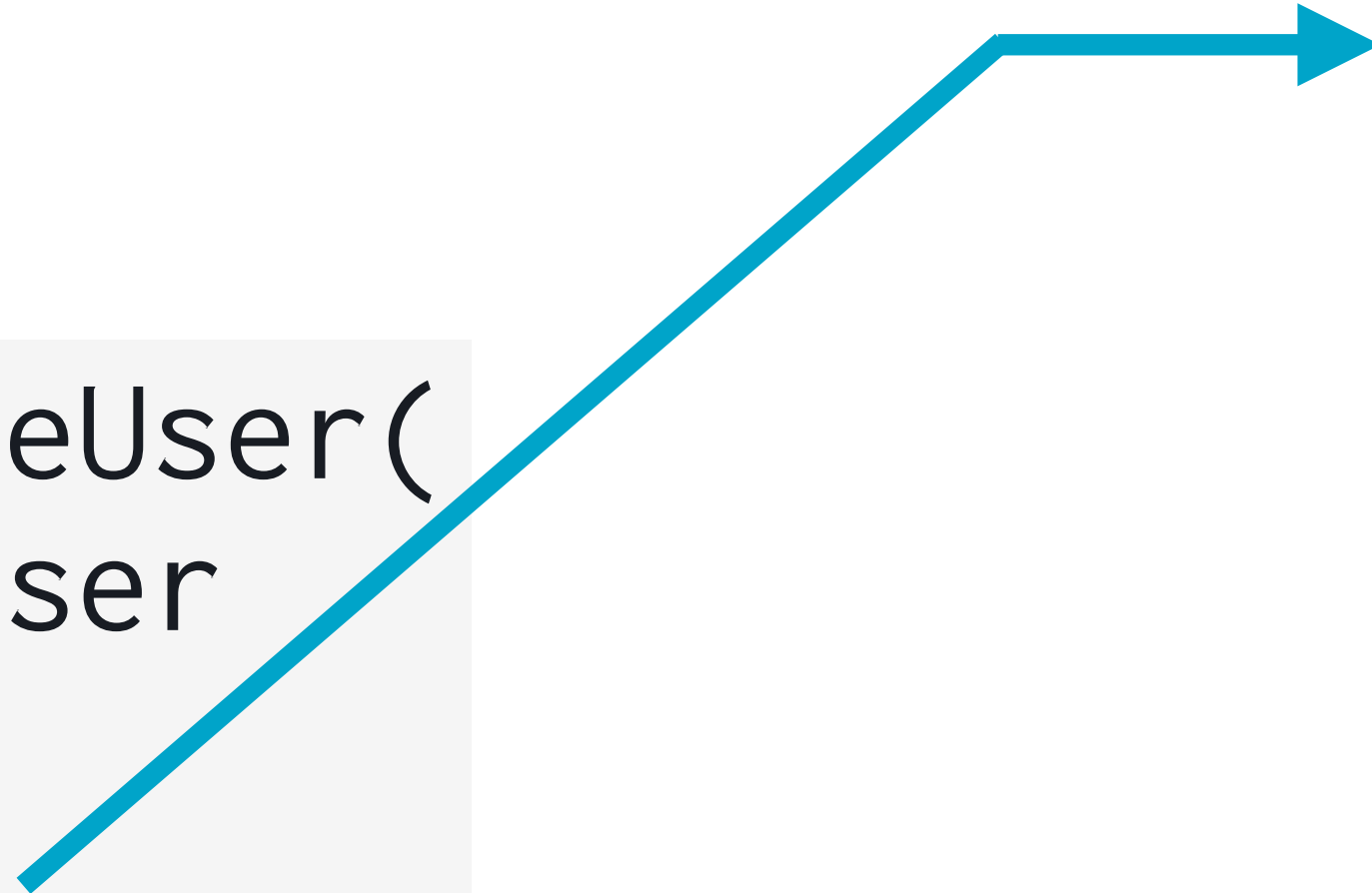
```
async function saveUser(  
  { id, email }: User  
): Promise<User> {  
  ...  
}
```

```
function checkId(string) {  
  ...  
}
```

```
function validEmail(string) {  
  ...  
}
```

```
async function saveUser(  
  { id, email }: User  
) : Promise<User> {  
  ...  
}
```

id: string



```
function checkId(string) {  
  ...  
}
```

```
function validEmail(string) {  
  ...  
}
```

```
async function saveUser(  
  { id, email }: User  
): Promise<User> {  
  ...  
}
```

id: string

```
function checkId(string) {  
  ...  
}
```

email: string

```
function validEmail(string) {  
  ...  
}
```



```
async function saveUser(  
  { id, email }: User  
): Promise<User> {  
  ...  
}
```

id: **string**

```
function checkId(string) {  
  ...  
}
```

email: **string**

```
function validEmail(string) {  
  ...  
}
```





Method #2: Brands

(Opaque/nominal types)


```
type Brand<WrappedType, TypeName> =  
    WrappedType & { __brand: TypeName };
```

```
type Brand<WrappedType, TypeName> =  
    WrappedType & { __brand: TypeName };  
  
type UserId = Brand<string, 'UserId'>;
```

```
type Brand<WrappedType, TypeName> =  
    WrappedType & { __brand: TypeName };  
  
type UserId = Brand<string, 'UserId'>;  
  
const id = '1234-abcd' as UserId;
```

```
type Brand<WrappedType, TypeName> =  
    WrappedType & { __brand: TypeName };  
  
type UserId = Brand<string, 'UserId'>;  
  
const id = '1234-abcd' as UserId;  
const email = 'a@b.c';
```

```
type Brand<WrappedType, TypeName> =  
    WrappedType & { __brand: TypeName };  
  
type UserId = Brand<string, 'UserId'>;  
  
const id = '1234-abcd' as UserId;  
const email = 'a@b.c';  
  
expectsUserId(id);
```

```
type Brand<WrappedType, TypeName> =  
  WrappedType & { __brand: TypeName };
```

```
type UserId = Brand<string, 'UserId'>;
```

```
const id = '1234-abcd' as UserId;
```

```
const email = 'a@b.c';
```

```
expectsUserId(id);
```

```
expectsUserId(email);
```

```
type Brand<WrappedType, TypeName> =  
  WrappedType & { __brand: TypeName };
```

```
type UserId = Brand<string, 'UserId'>;
```

```
const id = '1234-abcd' as UserId;
```

```
const email = 'a@b.c';
```

```
expectsUserId(id); // 
```

```
expectsUserId(email);
```



```
type Brand<WrappedType, TypeName> =  
  WrappedType & { __brand: TypeName };
```

```
type UserId = Brand<string, 'UserId'>;
```

```
const id = '1234-abcd' as UserId;
```

```
const email = 'a@b.c';
```

```
expectsUserId(id); // 
```

```
expectsUserId(email); // 
```

```
stringToUserId(id: string): UserId
```

```
userIdToString(id: UserId): string
```

```
type User = {  
  id:      UserId,  
  email: string,  
};
```

```
async function saveUser(  
  { id, email }: User  
): Promise<User> {  
  ...  
}
```

id: UserId

```
function checkId(UserId) {  
  ...  
}
```

```
async function saveUser(  
  { id, email }: User  
): Promise<User> {  
  ...  
}
```

```
function validEmail(string) {  
  ...  
}
```

email: string



4) Proving things to the type-checker

unknown \Rightarrow known

(implicitly: ... or throw an error)

`(Array<Thing>) => NonEmptyArray<Thing>`

```
function messageFromApi(  
  raw: unknown  
): { message: string } {  
  if (typeof raw == 'object' && raw != ...) {  
    return ...;  
  }  
}
```



```
import * as t from 'io-ts';

const User = t.type({
  id: t.string,
  email: t.string,
});

// Validation succeeded
User.decode(JSON.parse('{ "id": 1, "email": "g@z.com" }'));
// => Right({ id: 1, name: "g@z.com" })

// Validation failed
User.decode(JSON.parse('{ "email": "g@z.com" }'));
// => Left(...)
```

5) Lightning Round of Odd Tips

--strict

**Avoid numeric
enum**

```
enum MyEnum {  
    a = 1,  
    b = 2,  
}  
function fun(en: MyEnum) {  
    // ...  
}  
  
fun(666); // no error
```

<https://twitter.com/GiulioCanti/status/1105873882882412545>

(String enums are
fortunately unaffected by this;
thanks, @nhardy96!)

**readonly &
ReadOnly<...>**

<https://www.typescriptlang.org/docs/handbook/utility-types.html>

```
type User = {...};
```

```
// NewUser has everything except  
// the 'id' field from User:
```

```
type NewUser =  
    Exclude<User, 'id'>;
```

<https://www.typescriptlang.org/docs/handbook/utility-types.html>

type vs interface

https://medium.com/@martin_hotell/interface-vs-type-alias-in-typescript-2-7-2a8f1777af4c

interface



type



interface



type



interface



type



any

vs

mixed / unknown

vs

type variables (generics)



Bringing it all back.

- Shrink the possibilities for your input and output to as closely match what's going on in your head as you can muster.

- Shrink the possibilities for your input and output to as closely match what's going on in your head as you can muster.
- Use exhaustivity to give yourself a to-do list when you change code.

- Shrink the possibilities for your input and output to as closely match what's going on in your head as you can muster.
- Use exhaustivity to give yourself a to-do list when you change code.
- Use different types when you mean different things.

- Shrink the possibilities for your input and output to as closely match what's going on in your head as you can muster.
- Use exhaustivity to give yourself a to-do list when you change code.
- Use different types when you mean different things.
- 'Prove' things to the type checker so that it can do work for you.

- Shrink the possibilities for your input and output to as closely match what's going on in your head as you can muster.
- Use exhaustivity to give yourself a to-do list when you change code.
- Use different types when you mean different things.
- 'Prove' things to the type checker so that it can do work for you.
- don't use numeric enums, please

Better TypeScript Types



Rob Howard
@damncabbage
<http://robhoward.id.au>

- <https://michalzalecki.com/nominal-typing-in-typescript/>
- <https://www.typescriptlang.org/docs/handbook/utility-types.html>
- <https://dev.to/busypeoples/notes-on-typescript-pick-exclude-and-higher-order-components-40cp>
- https://medium.com/@martin_hotell/interface-vs-type-alias-in-typescript-2-7-2a8f1777af4c
- <https://mariusschulz.com/blog/typescript-3-0-the-unknown-type>
- <https://gcanti.github.io/io-ts/>
- <https://basarat.gitbooks.io/typescript/>