Pattern matching in TS

gillchristian @ HousingAnywhere

What's pattern matching?

"Checking a value against a pattern"

```
switch("hola") {
| "hola" => "HOLA"
| "chau" => "bye!"
| _      => "..."
}; /* "HOLA" */
```

```
switch("other") {
| "hola" => "HOLA"
| "chau" => "bye!"
| _      => "..."
}; /* "..." */
```

```scala
import scala.util.Random

val x: Int = Random.nextInt(10)

x match {
  case 0 => "zero"
  case 1 => "one"
  case 2 => "two"
  case _ => "many"
}
```

```scala
sealed abstract class Furniture
case class Couch() extends Furniture
case class Chair() extends Furniture

def findPlaceToSit(piece: Furniture): String = piece match {
  case a: Couch => "Lie on the couch"
  case b: Chair => "Sit on the chair"
}
```

```csharp
public static double ComputeAreaModernSwitch(object shape)
{
    switch (shape)
    {
        case Square s:
            return s.Side * s.Side;
        case Circle c:
            return c.Radius * c.Radius * Math.PI;
        case Rectangle r:
            return r.Height * r.Length;
        default:
            throw new ArgumentException(
                message: "shape is not a recognized shape",
                paramName: nameof(shape));
    }
}
```

```go
func do(i interface{}) {
    switch v := i.(type) {
    case int:
        fmt.Printf("Twice %v is %v\n", v, v*2)
    case string:
        fmt.Printf("%q is %v bytes long\n", v, len(v))
    default:
        fmt.Printf("I don't know about type %T!\n", v)
    }
}
```

```rust
let x = Some(5);
let y = 10;

match x {
    Some(50) => println!("Got 50"),
    Some(y)  => println!("Matched, y = {:?}", y),
    _        => println!("Default case, x = {:?}", x),
}
```

```
num : Int
num = 1

result : String
result =
  case num of
    1 -> "one"
    2 -> "two"
    _ -> "other"
```

```
map :: (a -> b) -> [a] -> [b]
map _ []     = []
map f (x:xs) = f x : map f xs
```

Source

```
type alias State = Int

type Action = Inc | Dec

reducer : State -> Action -> State
reducer state action =
  case action of
    Inc -> state + 1
    Dec -> state - 1


reducer 1 Inc -- 2
reducer 1 Dec -- 0
```

```elm
type alias State = Int

type Action
  = Inc
  | Dec
  | Add Int


reducer : State -> Action -> State
reducer state action =
  case action of
    Inc   -> state + 1
    Dec   -> state - 1
    Add x -> state + x
```

```elm
reducer 1 Inc         -- 2
reducer 1 Dec         -- 0
reducer 1 (Add 10)    -- 11
reducer 1 (Add -10)   -- -9
```
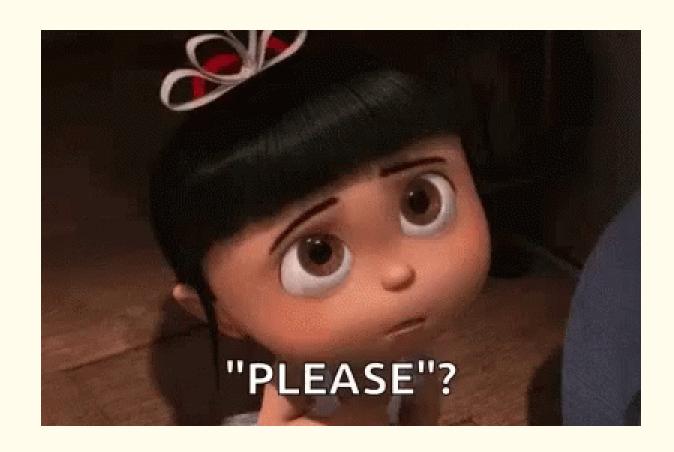
What about JavaScript?

We can only hope ...

tc39/proposal-pattern-matching

```
const res = await fetch(jsonService)
case (res) {
  when {status: 200, headers: {'Content-Length': s}} -> {
    console.log(`size is ${s}`)
  }
  when {status: 404} -> {
    console.log('JSON not found')
  }
  when {status} if (status >= 400) -> {
    throw new RequestError(res)
  }
}
```

```
<Fetch url={API_URL}>
  {props => case (props) {
    when {loading} -> <Loading />
    when {error} -> <Error error={error} />
    when {data} -> <Page data={data} />
    when _ -> throw new Error('badmatch')
  }}
</Fetch>
```

On the mean time ...

```
const fn = R.cond([
  [R.equals(0),   R.always('water freezes at 0°C')],
  [R.equals(100), R.always('water boils at 100°C')],
  [R.T,           t => `nothing special happens at ${t}°C`],
])


fn(0)   // => 'water freezes at 0°C'
fn(50)  // => 'nothing special happens at 50°C'
fn(100) // => 'water boils at 100°C'
```

ramdajs.com/docs/#cond

```
// static/js/ui/Chip.js

const createIcon = R.cond([
  [React.isValidElement, R.identity],
  [R.is(String), name => <SvgIcon name={name} />],
  [R.is(Object), props => <SvgIcon {...props} />],
  [R.T, () => null],
])
```

Many approaches

(list them)

https://blog.carbonfive.com/2018/01/08/an-introduction-to-adts-and-structural-pattern-matching-in-typescripthttps://pattern-matching-with-typescript.alabor.mehttps://blog.logrocket.com/pattern-matching-and-type-safety-in-typescript-1da1231a2e34https://blog.parametricstudios.com/posts/pattern-matching-custom-data-types

```
import match from '@housinganywhere/match'

type Variant =
  | 'success'
  | 'danger'
  | 'warning'

const variantColor = match<Variant, string>({
  success: () => 'green',
  danger: () => 'red',
  warning: () => 'yellow',
})
```

@housinganywhere/match

```
import { wildMatch } from '@housinganywhere/match';

type Vowels = 'a' | 'e' | 'i' | 'o' | 'u';

const isA = wildMatch<Vowels, string>({
  a: () => 'Yay!',
  _: (v) => `Nope, "${v}" is not "a"`,
});
```

```
type Matcher<T extends string, R> = { [K in T]: (k: K) => R };

const match = <T extends string, R = void>(m: Matcher<T, R>) => (t: T) => m[t](t);
```

```
type PartialMatcher<T extends string, R> =
  { [K in T]?: (k: K) => R } & { _: (t: T) => R; };

const wildMatch = <T extends string, R = void>(m: PartialMatcher<T, R>) => (t: T) => {
  const f = m[t];
  if (f) {
    return f(t);
  }

  return m._(t);
};
```

PR #1

```
type PayoutTypes = 'iban' | 'bank' | 'paypal'

const PayoutMethod = ({ payoutMethod, payoutType }) =>
  <div>
    {match<PayoutTypes, React.ReactNode>({
      iban: () => (
        <IbanMethod method={payoutMethod} isNew={!payoutMethod} />
      ),
      bank: () => (
        <BankMethod method={payoutMethod} isNew={!payoutMethod} />
      ),
      paypal: () => (
        <PaypalMethod method={payoutMethod} isNew={!payoutMethod} />
      ),
    })(payoutType)}
  </div>
```

Code time !!!

So far we have ...

```
match<'foo' | 'bar'> // states

match<{ name } | { email }> // data
```

```
type RemoteData<D, E> =
  | NotAsked
  | Loading
  | Success<D>
  | Failure<E>
```

gillchristian/remote-data-ts

```
const renderArticle = cata<Article, string, React.ReactNode>({
  notAsked:          () => <Empty />,
  loading:           () => <Spinner />,
  success: (article) => <Article {...article} />,
  error:       (msg) => <Msg variant="danger">{msg}</Msg>,
})
```

Map (map, then)

```
// (A -> B) -> RD<A, E> -> RD<B, E>
// (A -> B) -> A[]        -> B[]

// Promise<A> -> (A -> B) -> Promise<B>

<A, E, B>(fn: (d: A) => B) => (rd: RemoteData<A, E>) => RemoteData<B, E>;
```

Chain (then, flatMap)

```
// (A -> B[])        -> A[]        -> B[]
// (A -> RD<B, E>) -> RD<A, E> -> RD<B, E>

// Promise<A> -> (A -> Promise<B>) -> Promise<B>

<D, E, R>(fn: (d: D) => RemoteData<R, E>) => (rd: RemoteData<D, E>) => RemoteData<R, E>;
```

Fold (withDefault)

```
<D, E, R>(fn: (d: D) => R) => (def: R) => (rd: RemoteData<D, E>) => R;
```

Example

Questions?

THE END