```
type Success<T> = {
	data: T
	error: null
}

type Failure<E> = {
	data: null
	error: E
}

export class Result<T, E = Error> {
	private inner: Success<T> | Failure<E>

	private constructor(inner: Success<T> | Failure<E>) {
		this.inner = inner
	}

	static ok<T, E = Error>(value: T) {
		return new Result<T, E>({ data: value, error: null })
	}

	static err<T, E = Error>(value: E) {
		return new Result<T, E>({ data: null, error: value })
	}

	is_err() {
		return this.inner.error !== null
	}
	unwrap() {
		if (this.inner.error !== null) {
			throw this.inner.error
		}

		return this.inner.data
	}
}

export async function try_catch<
	T,
	F extends (...params: any[]) => Promise<T>,
	E = Error,
>(f: F, ...args: Parameters<F>): Promise<Result<T, E>> {
	try {
		return Result.ok(await f(...args))
	} catch (error) {
		return Result.err(error as E)
	}
}
```