

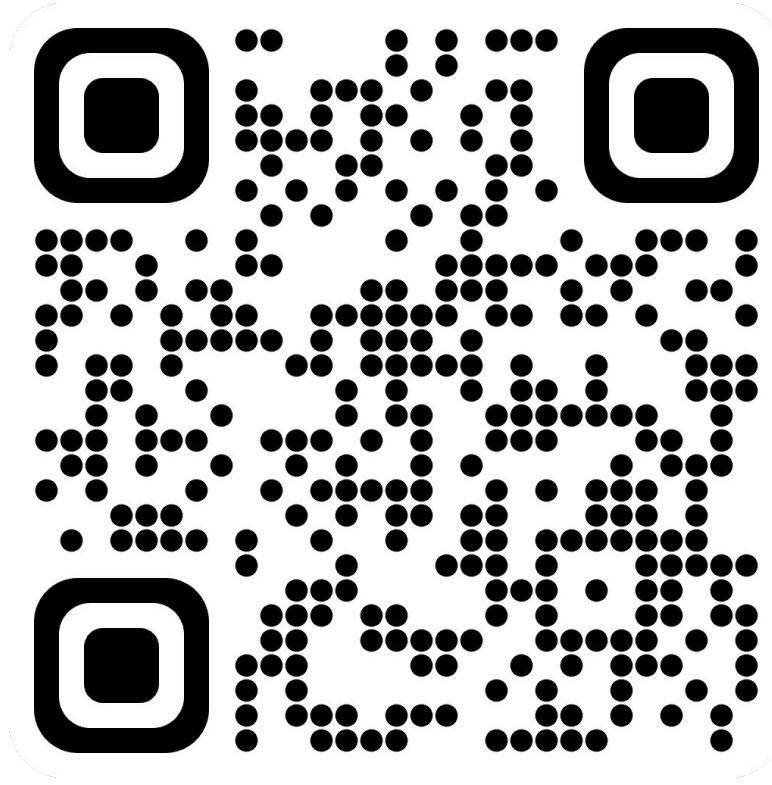
# Mónadas en Python

Menos Excepciones y Más Tipado en Nuestro Código



Artur Costa-Pazo





<https://github.com/alice-biometrics/pycones23>



# Mónada



# Mónada

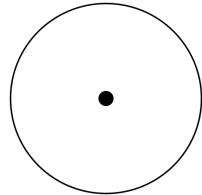
“Eso es un concepto filosófico, ¿no?”

# Mónada

“Eso es un concepto filosófico, ¿no?”

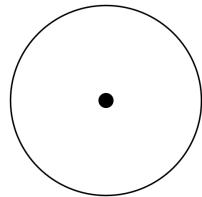


# Mónada



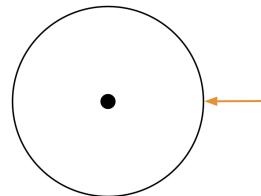
del griego *μονάς* monas, "**unidad**"  
de *μόνος* monos, "**uno**", "**solo**", "**único**"

# Mónada



Término usado en filosofía que significa “**unidad**”, utilizado de diferentes maneras por los filósofos.

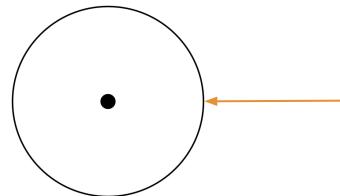
# Mónada



Mónada  
Pitagórica

Término usado en filosofía que significa “**unidad**”, utilizado de diferentes maneras por los filósofos.

# Mónada



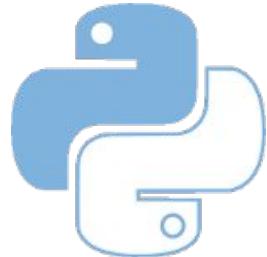
Mónada  
Pitagórica

**Cada una de las sustancias indivisibles, pero de naturaleza distinta,** que componen el universo, según el sistema de Leibniz, filósofo y matemático alemán del siglo XVII.

# ¿Quien soy?

Artur Costa-Pazo 

*Research Software Engineer*



# ¿Quien soy?

**Artur Costa-Pazo** 

*Research Software Engineer*

*PhD on Face Presentation Attack Detection*

*Head of Engineering at*  **Alice**



# ¿Quien soy?

# Artur Costa-Pazo

# *Research Software Engineer*

# *PhD on Face Presentation Attack Detection*

Head of Engineering at  Alice



# ¿Quien soy?

# Artur Costa-Pazo

# *Research Software Engineer*

# *PhD on Face Presentation Attack Detection*

Head of Engineering at  Alice



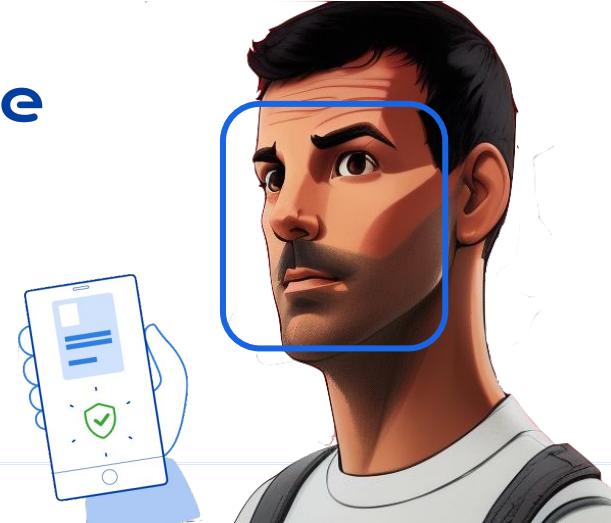
# ¿Quien soy?

# Artur Costa-Pazo

# *Research Software Engineer*

# *PhD on Face Presentation Attack Detection*

Head of Engineering at  Alice



# ¿Quien soy?

# Artur Costa-Pazo

# *Research Software Engineer*

# *PhD on Face Presentation Attack Detection*

Head of Engineering at  Alice



# ¿Por qué vienes a hablar de móndas?

En  **Alice** llevamos 4 años utilizando móndas en todos nuestros servicios gracias a la ayuda del paquete Python **meiga** .

# ¿Por qué vienes a hablar de móndas?

En  Alice llevamos 4 años utilizando móndas en todos nuestros servicios gracias a la ayuda del paquete Python **meiga** .

```
pip install meiga
```

# ¿Por qué vienes a hablar de móndas?

En  **Alice** llevamos 4 años utilizando móndas en todos nuestro servicios gracias a la ayuda del paquete Python **meiga** .

```
pip install meiga
```

<https://github.com/alice-biometrics/meiga>

# ¿Por qué vienes a hablar de móndas?

En  **Alice** llevamos 4 años utilizando móndas en todos nuestro servicios gracias a la ayuda del paquete Python **meiga** .

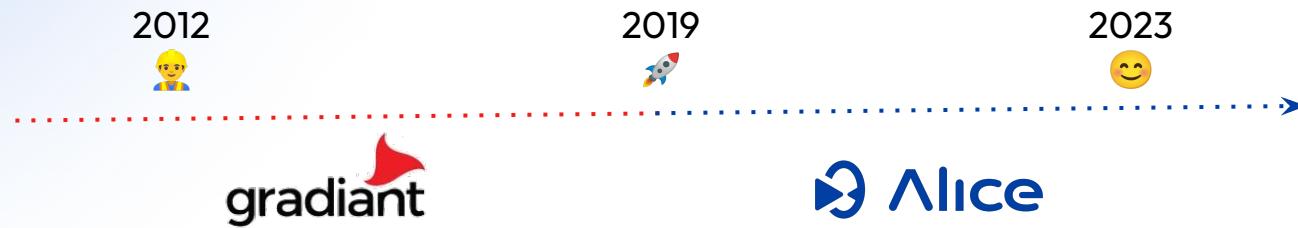
```
pip install meiga
```



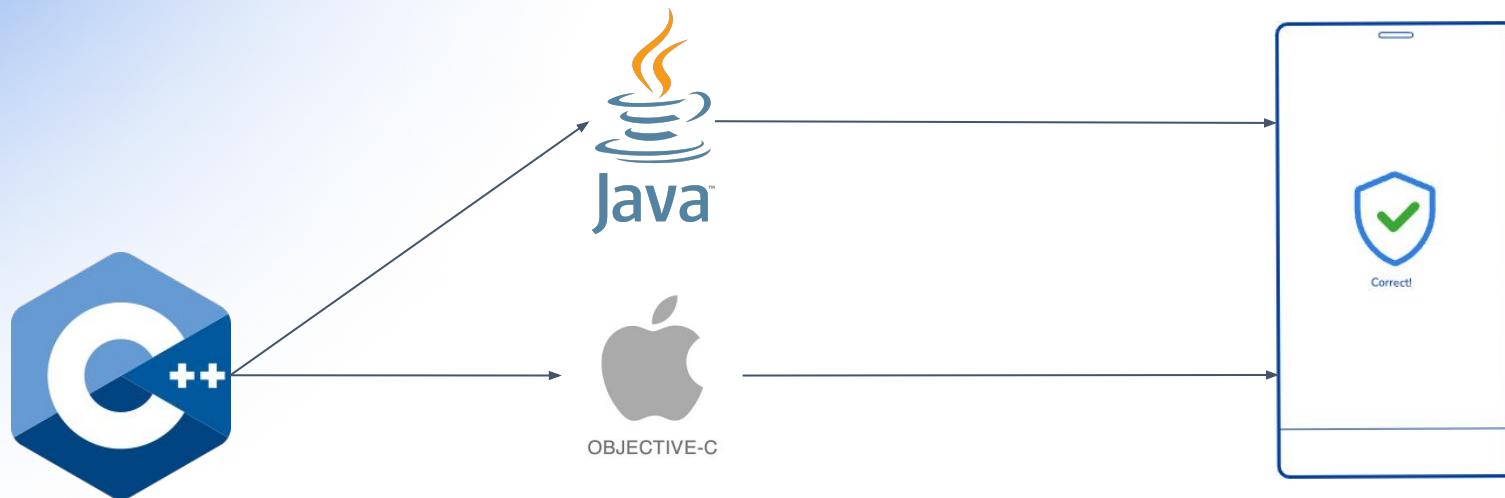
<https://github.com/alice-biometrics/meiga>

# ¿Y cómo empezasteis a utilizarlas?

# ¿Y cómo empezasteis a utilizarlas?



# ¿Y cómo empezasteis a utilizarlas?



Modelos de Visión  
Artificial y proto AI.

# ¿Y cómo empezasteis a utilizarlas?



Modelos de Visión  
Artificial y proto AI.

```
#include <jni.h>
#include <iostream>
#include "TestJNIPrimitive.h"
using namespace std;

JNIEXPORT jdouble JNICALL Java_TestJNIPrimitive_average
    (JNIEnv *env, jobject obj, jint n1, jint n2) {
    jdouble result;
    cout << "In C++, the numbers are " << n1 << " and " << n2 << endl;
    result = ((jdouble)n1 + n2) / 2.0;
    // jint is mapped to int, jdouble is mapped to double
    ...

@implementation MyObjective_c_file
...
- (void)myMethod: (NSString *) dataPayload {
    // Your logic here
}

@end

void callFromCpp(const char *dataPayload) {
    NSString *convertedString = [[NSString alloc] initWithCString:
        dataPayload encoding:NSUTF8StringEncoding];

    MyObjective_c_file myInstance = [[MyObjective_c_file alloc] init];
    [MyObjective_c_file myMethod: convertedString];
}
```

# ¿Y cómo empezasteis a utilizarlas?



```
#include <jni.h>
#include <iostream>
#include "TestJNIPrimitive.h"
using namespace std;

JNIEXPORT jdouble JNICALL Java_TestJNIPrimitive_average
    (JNIEnv *env, jobject obj, jint n1, jint n2) {
    jdouble result;
    cout << "In C++, the numbers are " << n1 << " and " << n2 << endl;
    result = ((jdouble)n1 + n2) / 2.0;
    // jint is mapped to int, jdouble is mapped to double
    ...

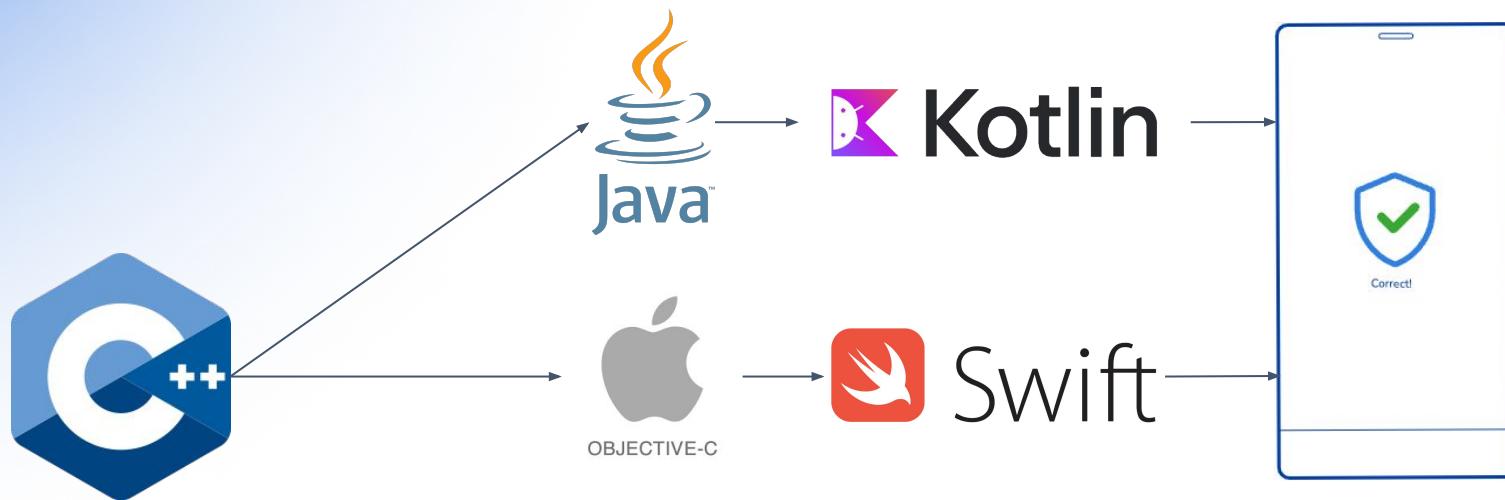
@implementation MyObjective_c_file
...
- (void)myMethod: (NSString *) dataPayload {
    // Your logic here
}

@end

void callFromCpp(const char *dataPayload) {
    NSString *convertedString = [[NSString alloc] initWithCString:
        dataPayload encoding:NSUTF8StringEncoding];

    MyObjective_c_file myInstance = [[MyObjective_c_file alloc] init];
    [MyObjective_c_file myMethod: convertedString];
}
```

# ¿Y cómo empezasteis a utilizarlas?



Modelos de Visión  
Artificial y proto AI.

# ¿Y cómo empezasteis a utilizarlas?



# Swift

```
typealias JSONObject = [String: Any]

enum JSONError: Error {
    case noSuchKey(String)
    case typeMismatch
}

func stringForKey(json: JSONObject, key: String) → Result<String, JSONError> {
    guard let value = json[key] else {
        return .failure(.noSuchKey(key))
    }

    guard let value = value as? String else {
        return .failure(.typeMismatch)
    }

    return .success(value)
}
```

# ¿Y cómo empezasteis a utilizarlas?



```
typealias JSONObject = [String: Any]

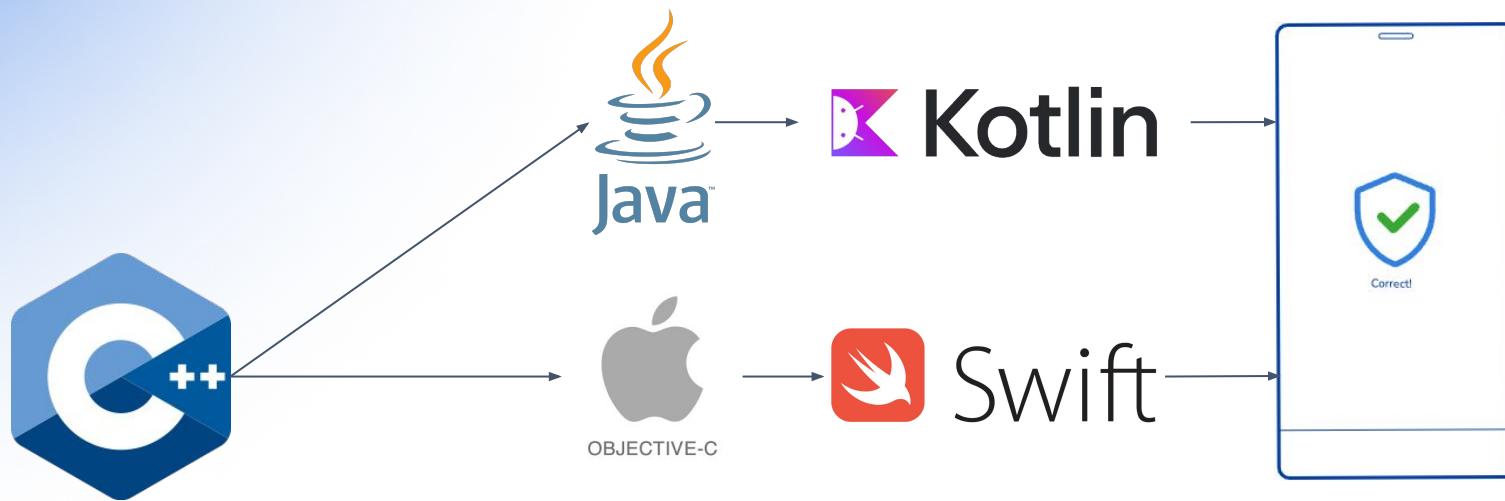
enum JSONError: Error {
    case noSuchKey(String)
    case typeMismatch
}

func stringForKey(json: JSONObject, key: String) → Result<String, JSONError> {
    guard let value = json[key] else {
        return .failure(.noSuchKey(key))
    }

    guard let value = value as? String else {
        return .failure(.typeMismatch)
    }

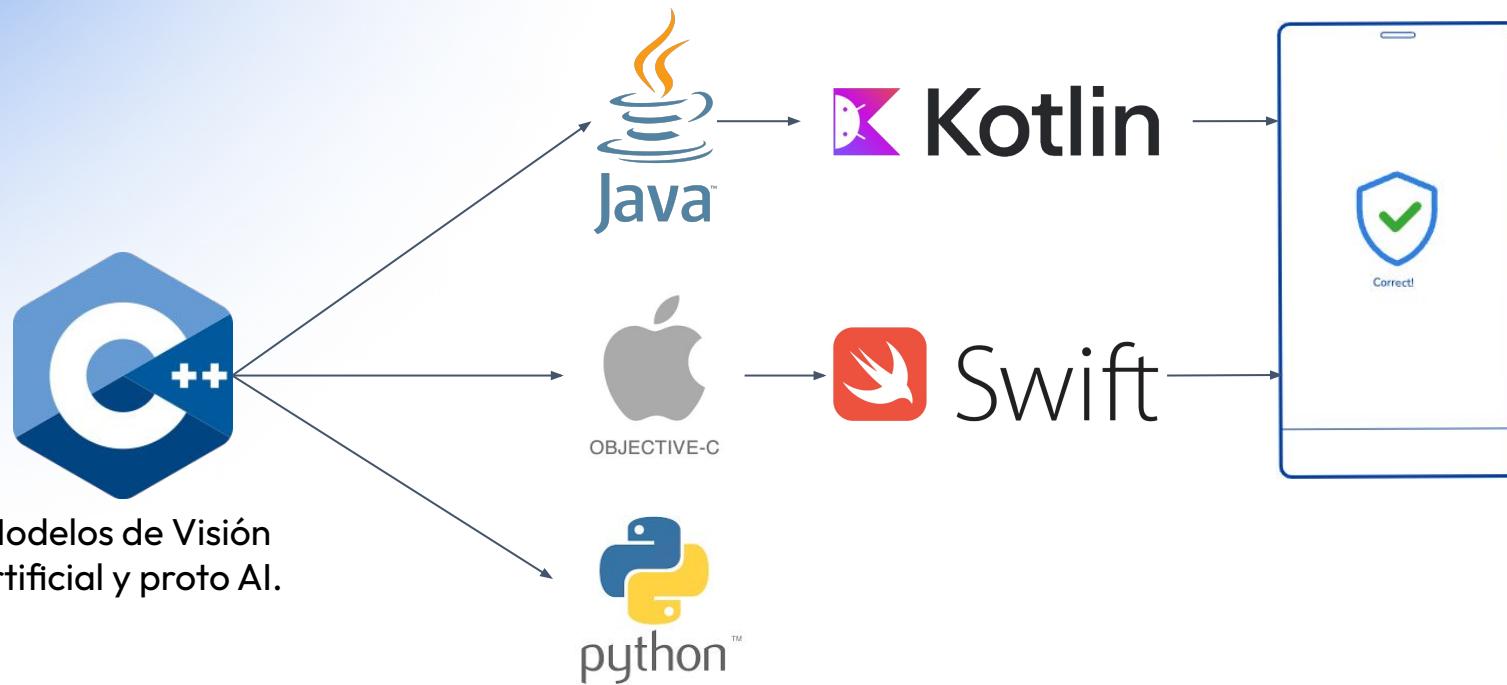
    return .success(value)
}
```

# ¿Y cómo empezasteis a utilizarlas?



Modelos de Visión  
Artificial y proto AI.

# ¿Y cómo empezasteis a utilizarlas?



# ¿Y cómo empezasteis a utilizarlas?

Blog Post:  
[Remove visual noise of logging code with python decorators](#)

## The Problem

„*Error handling [Logging...] (editor's note) is important, but if it obscures logic, it's wrong.*“

— Robert C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*

# ¿Y cómo empezasteis a utilizarlas?

```
import requests
import logging

logger = logging.getLogger(__name__)

class DownloadFailedError(Exception):
    pass


def download(url):
    logger.debug("Starting to download {}".format(url))
    response = requests.get(url)

    if response.status_code != 200:
        logger.error("Download was not successful due to response status {}".format(response.status))
        raise DownloadFailedError

    filename = url.split("/")[-1]
    filename = filename or "index.html"
    logger.debug("Generated filename '{}' from url '{}'".format(filename, url))

    logger.debug("Writing downloaded content into file")
    with open(filename, "wb") as f:
        f.write(response.content)
    logger.info("Downloading of '{}' into file '{}' was successful"
               .format(url, filename))
```

# ¿Y cómo empezasteis a utilizarlas?

```
import requests
import logging

logger = logging.getLogger(__name__)

class DownloadFailedError(Exception):
    pass


def download(url):
    logger.debug("Starting to download {}".format(url))
    response = requests.get(url)

    if response.status_code != 200:
        logger.error("Download was not successful due to response status {}".format(response.status))
        raise DownloadFailedError

    filename = url.split("/")[-1]
    filename = filename or "index.html"
    logger.debug("Generated filename '{}' from url '{}'".format(filename, url))

    logger.debug("Writing downloaded content into file")
    with open(filename, "wb") as f:
        f.write(response.content)
    logger.info("Downloading of '{}' into file '{}' was successful"
               .format(url, filename))
```

# ¿Y cómo empezasteis a utilizarlas?

```
@log_on_start(DEBUG, "Starting to download {url:s}")
@log_on_end(INFO, "Downloading of '{url:s}' into file '{result:s}' was successful")
def download(url):
    content = content_from_url(url)
    filename = url_to_filename(url)

    write_to_file(filename, content)
    return filename
```

# ¿Y cómo empezasteis a utilizarlas?



Swift



*Remove visual noise... =*

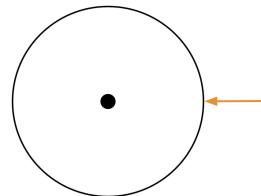


(meiga

# Mónada



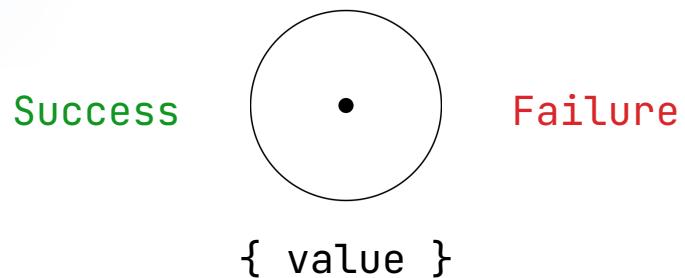
# Mónada



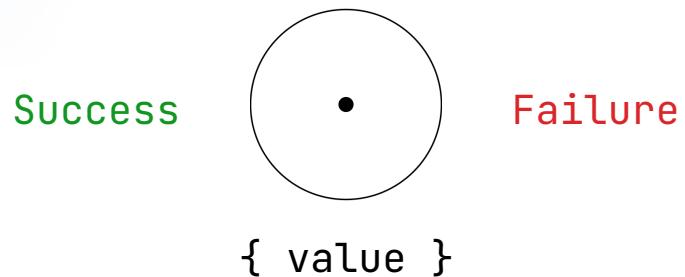
Mónada  
Pitagórica

**Cada una de las sustancias indivisibles, pero de naturaleza distinta,** que componen el universo, según el sistema de Leibniz, filósofo y matemático alemán del siglo XVII.

# Mónada

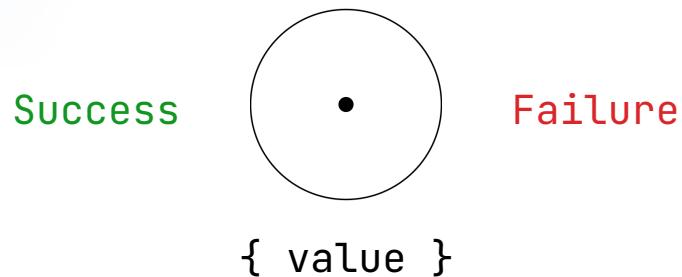


# Mónada



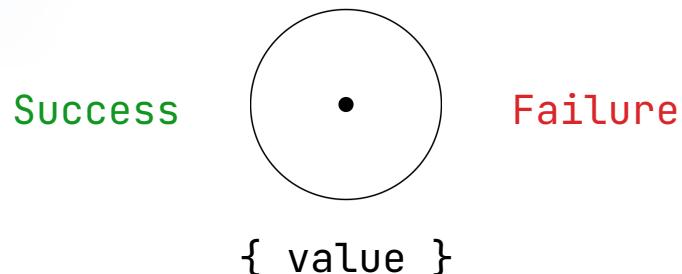
```
def foo() → Result[Success,Failure]:
```

# Mónada



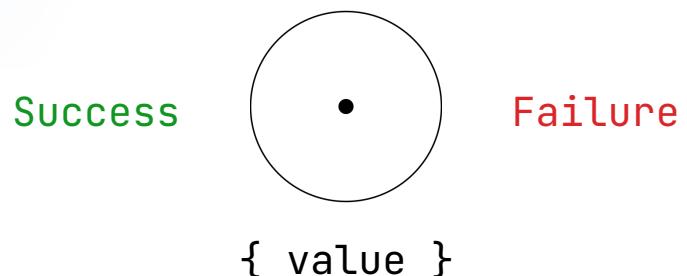
```
def foo() → Result[str,Failure]:
```

# Mónada



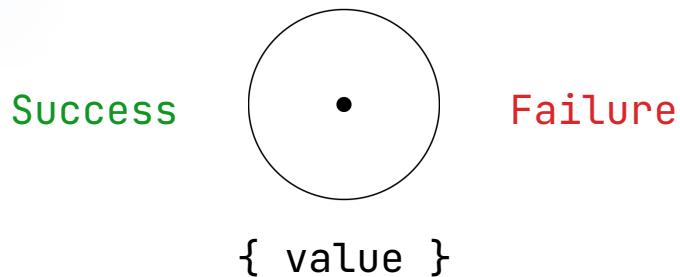
```
def foo() → Result[int,Failure]:
```

# Mónada



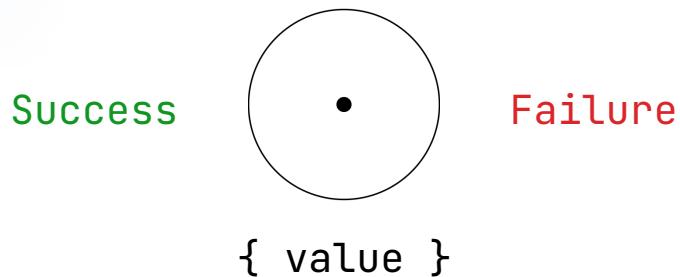
```
def foo() → Result[str|int, Failure]:
```

# Mónada



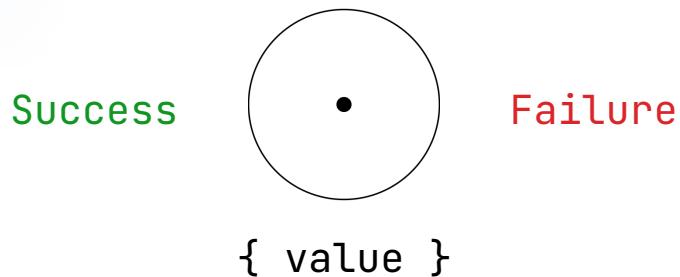
```
def foo() → Result[list, Failure]:
```

# Mónada



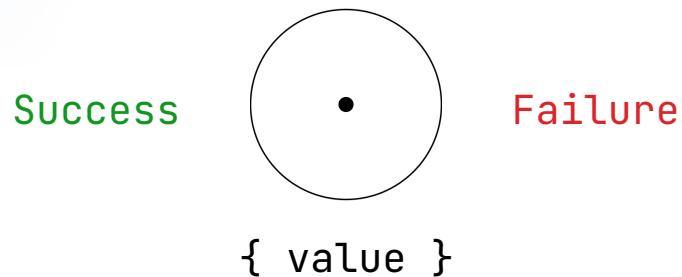
```
def foo() → Result[MyModel, Failure]:
```

# Mónada



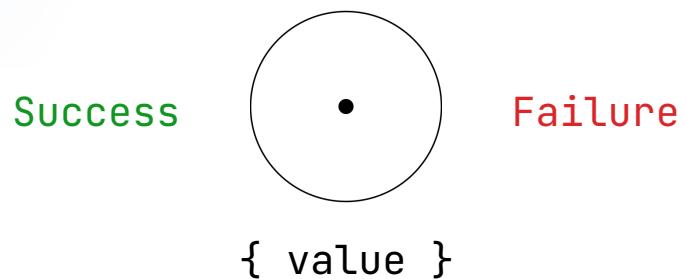
```
def foo() → Result[MyModel, NotFound]:
```

# Mónada



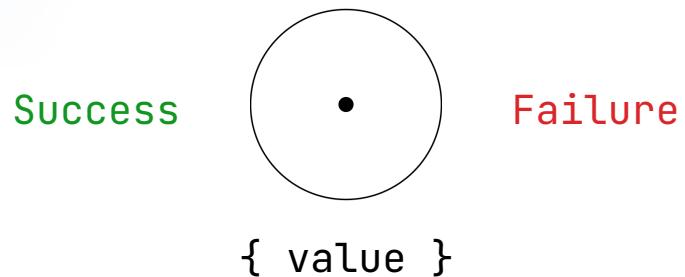
```
def foo() → Result[MyModel,NotAllowed]:
```

# Mónada



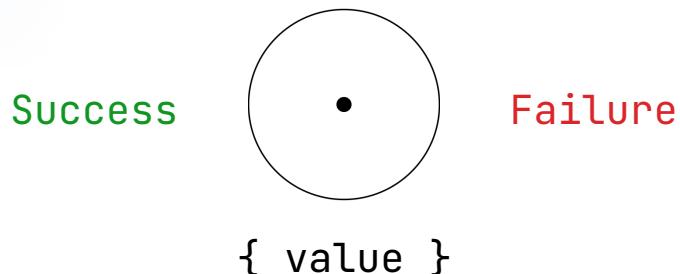
```
def foo() → Result[MyModel, NotFound|NotAllowed]:
```

# Mónada



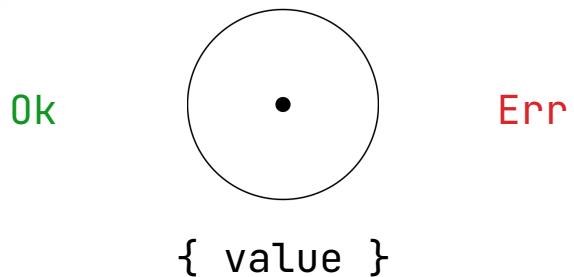
```
def foo() → Result[MyModel, MyError]:
```

# Mónada



```
def foo() → Result[Success,Failure]:
```

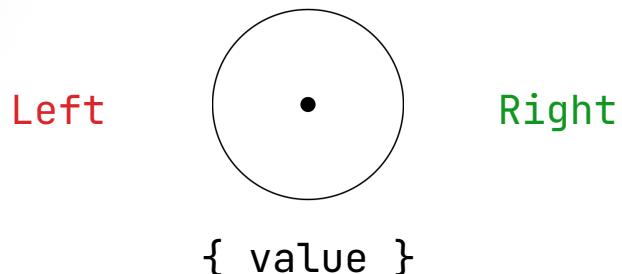
# Mónada



```
def foo() → Result[Ok, Err]:
```



# Mónada



```
def foo() → Either[Left,Right]:
```

 Haskell  
Either



# Ejemplo

```
dictionary: dict[str, Any] = {"key1": "1", "key2": 2}  
value: str = string_from_key(dictionary, "key1")
```

Queremos una función que nos devuelva valores de un dict:

# Ejemplo

```
dictionary: dict[str, Any] = {"key1": "1", "key2": 2}  
value: str = string_from_key(dictionary, "key1")
```

Queremos una función que nos devuelva valores de un dict.

- Solo queremos que nos devuelva `str`.

# Ejemplo

```
dictionary: dict[str, Any] = {"key1": "1", "key2": 2}  
value: str = string_from_key(dictionary, "invalid_key")
```

Queremos una función que nos devuelva valores de un dict.

- Solo queremos que nos devuelva `str`.
- Si la `key` que le pasamos no existe queremos que nos devuelva un error.

# Ejemplo

```
dictionary: dict[str, Any] = {"key1": "1", "key2": 2}  
value: str = string_from_key(dictionary, "key2")
```

Queremos una función que nos devuelva valores de un dict.

- Solo queremos que nos devuelva `str`.
- Si la `key` que le pasamos no existe queremos que nos **devuelva un error**.
- Si el valor recuperado no es un `str` queremos que nos **devuelva un error**.

## Con Excepciones

```
dictionary: dict[str, Any] = {"key1": "1", "key2": 2}  
  
value: str = string_from_key(dictionary, "key1")
```



```
class NoSuchKey(Exception): ...  
class TypeMismatch(Exception): ...  
  
def string_from_key(dictionary: dict, key: str) → str:  
    if key not in dictionary.keys():  
        raise NoSuchKey()  
  
    value = dictionary.get(key)  
    if not isinstance(value, str):  
        raise TypeMismatch()  
  
    return value
```

## Con Excepciones

```
dictionary: dict[str, Any] = {"key1": "1", "key2": 2}  
  
value: str = string_from_key(dictionary, "key1")
```



```
class NoSuchKey(Exception): ...  
class TypeMismatch(Exception): ...
```

```
def string_from_key(dictionary: dict, key: str) → str:  
    if key not in dictionary.keys():  
        raise NoSuchKey()  
  
    value = dictionary.get(key)  
    if not isinstance(value, str):  
        raise TypeMismatch()  
  
    return value
```

Return type hint

Exceptions

## Con Mónadas

```
dictionary: dict[str, Any] = {"key1": "1", "key2": 2}  
  
value: str = string_from_key(dictionary, "key1")
```



```
class NoSuchKey(Error): ...  
class TypeMismatch(Error): ...  
  
def string_from_key(dictionary: dict, key: str) → Result[str, Error]:  
    if key not in dictionary.keys():  
        return Failure(NoSuchKey())  
  
    value = dictionary.get(key)  
    if not isinstance(value, str):  
        return Failure(TypeMismatch())  
  
    return Success(value)
```

## Con Mónadas

```
dictionary: dict[str, Any] = {"key1": "1", "key2": 2}  
  
value: str = string_from_key(dictionary, "key1")
```



```
class NoSuchKey(Error): ...  
class TypeMismatch(Error): ...  
  
def string_from_key(dictionary: dict, key: str) → Result[str, Error]:  
    if key not in dictionary.keys():  
        return Failure(NoSuchKey())  
  
    value = dictionary.get(key)  
    if not isinstance(value, str):  
        return Failure(TypeMismatch())  
  
    return Success(value)
```

- A blue dashed box surrounds the type signature `Result[str, Error]`. A blue arrow points from the word `Monad` to this box.
- Two red arrows point from the text `Failure` to two separate `Failure` return statements in the code.
- A green arrow points from the text `Success` to the final `return Success(value)` statement in the code.

## Con Excepciones

```
class NoSuchKey(Exception): ...
class TypeMismatch(Exception): ...

def string_from_key(dictionary: dict, key: str) → str:
    if key not in dictionary.keys():
        raise NoSuchKey()

    value = dictionary.get(key)
    if not isinstance(value, str):
        raise TypeMismatch()

    return value
```

Return type hint

Exceptions

## Con Mónadas

```
class NoSuchKey(Error): ...
class TypeMismatch(Error): ...

def string_from_key(dictionary: dict, key: str) → Result[str, Error]:
    if key not in dictionary.keys():
        return Failure(NoSuchKey())

    value = dictionary.get(key)
    if not isinstance(value, str):
        return Failure(TypeMismatch())

    return Success(value)
```

Monad

Failure

Success

## Con Excepciones

```
class NoSuchKey(Exception): ...
class TypeMismatch(Exception): ...

@to_result
def string_from_key(dictionary: dict, key: str) → str:
    if key not in dictionary.keys():
        raise NoSuchKey()

    value = dictionary.get(key)
    if not isinstance(value, str):
        raise TypeMismatch()

    return value
```

## Con Mónadas

```
class NoSuchKey(Error): ...
class TypeMismatch(Error): ...

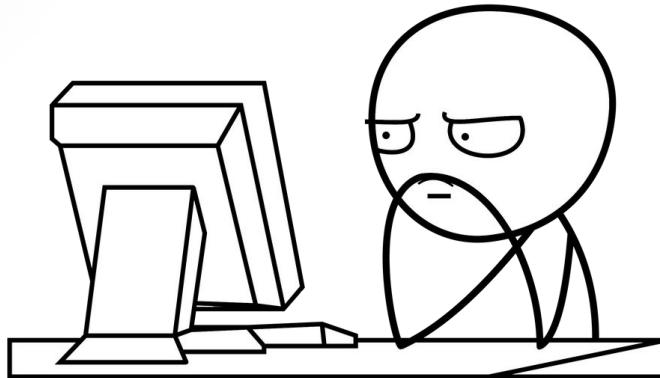
def string_from_key(dictionary: dict, key: str) → Result[str, Error]:
    if key not in dictionary.keys():
        return Failure(NoSuchKey())

    value = dictionary.get(key)
    if not isinstance(value, str):
        return Failure(TypeMismatch())

    return Success(value)
```

Monad

# ¿Por qué utilizar monads?



# ¿Por qué utilizar móndadas?



# ¿Por qué utilizar mónadas?

Nos ayuda a tener un mejor tipado.

Nos facilita el testing.

Nos habilita a usar funciones derivadas de la programación funcional.

Nos ayuda a crear "código limpio".

# ¿Por qué utilizar móndas?

→ Nos ayuda a tener un mejor tipado.

Nos facilita el testing.

Nos habilita a usar funciones derivadas de la programación funcional.

Nos ayuda a crear código limpio.

# Mejor Tipado

```
value = string_from_key(dictionary, "key1")
```

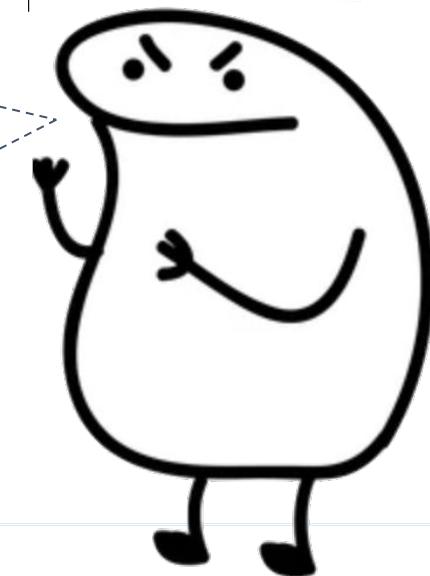
```
value: str = string_from_key(dictionary, "key1")
```

# Mejor Tipado

```
value = string_from_key(dictionary, "key1")
```

```
value: str = string_from_key(dictionary, "key1")
```

Y si falla.....  
¿Cuáles son las  
excepciones que tengo  
que capturar?

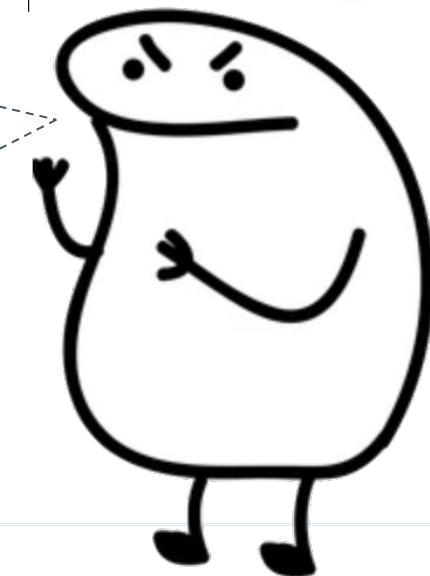


# Mejor Tipado

```
value = string_from_key(dictionary, "key1")
```

```
value: str = string_from_key(dictionary, "key1")
```

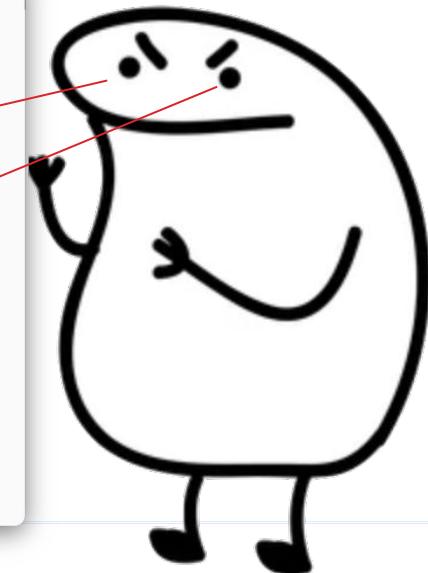
Y si falla.....  
¿Cuáles son las  
excepciones que tengo  
que capturar?



# Mejor Tipado

```
value = string_from_key(dictionary, "key1")  
value: str = string_from_key(dictionary, "key1")
```

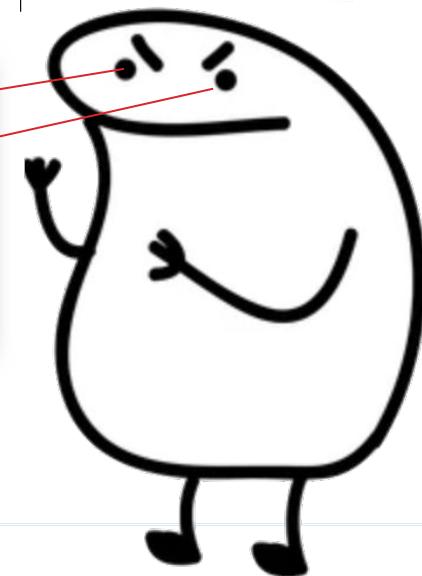
```
class NoSuchKey(Exception): ...  
class TypeMismatch(Exception): ...  
  
def string_from_key(dictionary: dict, key: str) → str:  
    if key not in dictionary.keys():  
        raise NoSuchKey()  
  
    value = dictionary.get(key)  
    if not isinstance(value, str):  
        raise TypeMismatch()  
  
    return value
```



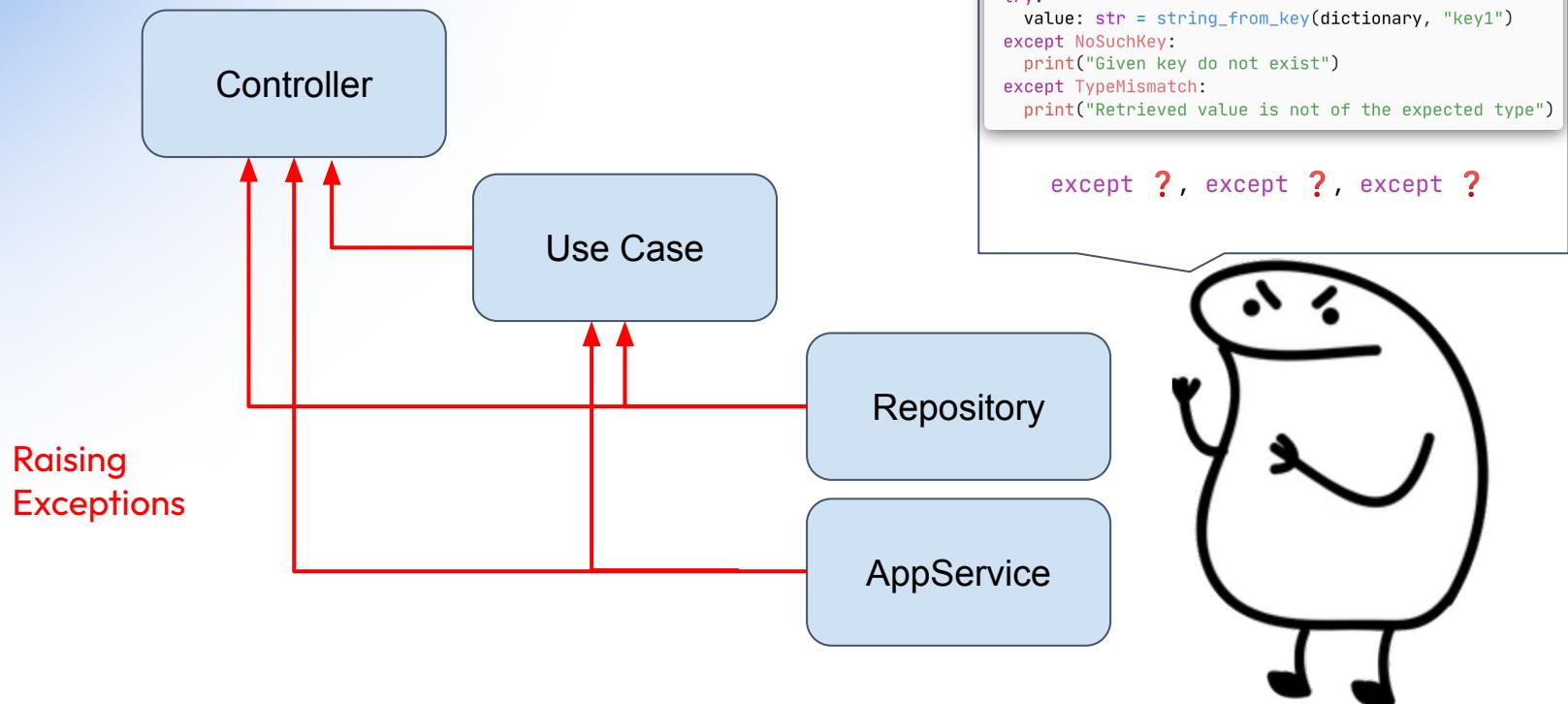
# Mejor Tipado

```
value = string_from_key(dictionary, "key1")  
value: str = string_from_key(dictionary, "key1")
```

```
try:  
    value: str = string_from_key(dictionary, "key1")  
except NoSuchKey:  
    print("Given key do not exist")  
except TypeMismatch:  
    print("Retrieved value is not of the expected type")
```



# Mejor Tipado



# Mejor Tipado

```
value = string_from_key(dictionary, "key1")
```

```
value: str = string_from_key(dictionary, "key1")
```

¿Y qué pasa si utilizo  
la clase Result?



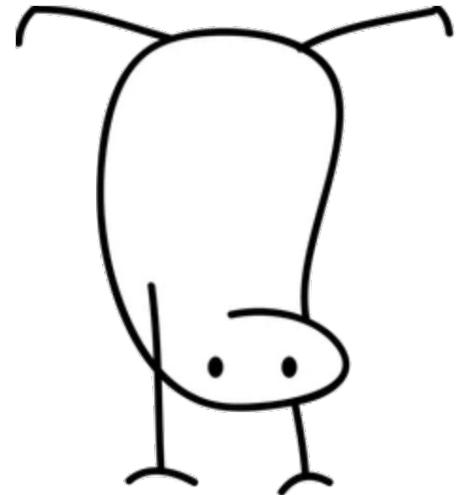
# Mejor Tipado

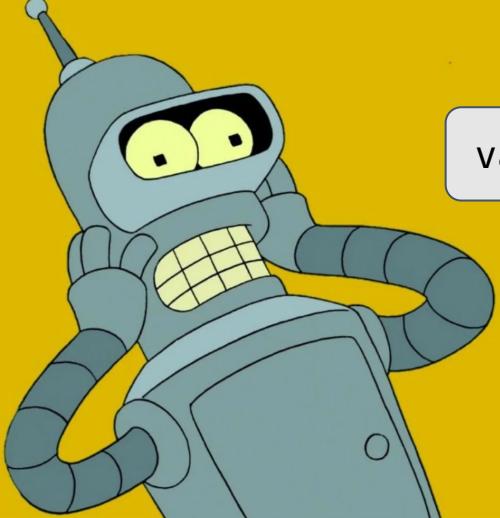
```
result = string_from_key(dictionary, "key1")  
result: Result[str, NoSuchKey | TypeMismatch] = string_from_key(dictionary, "key1")
```

# Mejor Tipado

```
result = string_from_key(dictionary, "key1")  
  
result: Result[str, NoSuchKey | TypeMismatch] = string_from_key(dictionary, "key1")
```

```
result = string_from_key(dictionary, "key1")  
match result:  
    case Success(value):  
        print(value)  
    case Failure(NoSuchKey):  
        print("Given key do not exist")  
    case Failure(TypeMismatch):  
        print("Retrieved value is not of the expected type")
```

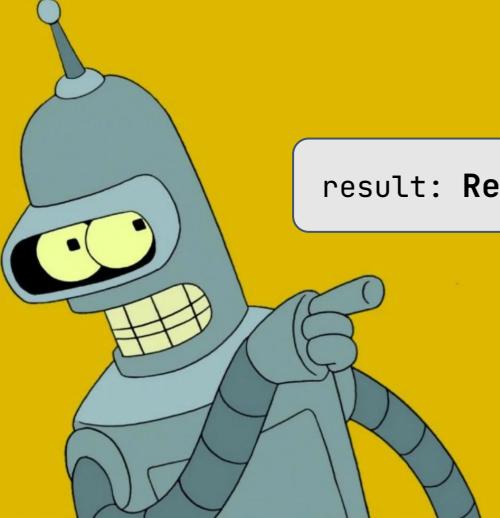




```
value = string_from_key(dictionary, "key1")
```

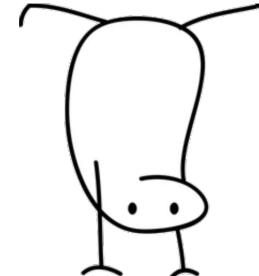
```
value: str = string_from_key(dictionary, "key1")
```

Y si falla.....  
¿Cuáles son las  
excepciones que  
tengo que capturar?



```
result = string_from_key(dictionary, "key1")
```

```
result: Result[str, NoSuchKey | TypeMismatch] = string_from_key(dictionary, "key1")
```



# ¿Por qué utilizar mónadas?

- ✓ Nos ayuda a tener un mejor tipado.
- Nos facilita el testing.
  - Nos habilita a usar funciones derivadas de la programación funcional.
  - Nos ayuda a crear "código limpio".

# Testing

```
@pytest.mark.unit
class TestExampleWithoutMeiga:
    dictionary = {"key1": "value", "key2": 2}

    def should_return_a_str(self):
        value = string_from_key(
            dictionary=self.dictionary,
            key="key1"
        )
        assert isinstance(value, str)

    def should_raises_non_such_key_exception(self):
        with pytest.raises(NoSuchKey):
            _ = string_from_key(
                dictionary=self.dictionary,
                key="invalid_key"
            )

    def should_raises_type_mismatch_exception(self):
        with pytest.raises(TypeMismatch):
            value = string_from_key(
                dictionary=self.dictionary,
                key="key2"
            )
        assert not isinstance(value, str)
```

```
@pytest.mark.unit
class TestExampleWithMeiga:
    dictionary = {"key1": "value", "key2": 2}

    def should_return_a_str(self):
        result = string_from_key(
            dictionary=self.dictionary,
            key="key1"
        )
        result.assert_success(value_is_instance_of=str)

    def should_raises_non_such_key_exception(self):
        result = string_from_key(
            dictionary=self.dictionary,
            key="invalid_key"
        )
        result.assert_failure(value_is_instance_of=NoSuchKey)

    def should_raises_type_mismatch_exception(self):
        result = string_from_key(
            dictionary=self.dictionary,
            key="key2"
        )
        result.assert_failure(value_is_instance_of=TypeMismatch)
```

# Testing

```
@pytest.mark.unit
class TestExampleWithoutMeiga:
    dictionary = {"key1": "value", "key2": 2}

    def should_return_a_str(self):
        value = string_from_key(
            dictionary=self.dictionary,
            key="key1"
        )
        assert isinstance(value, str)

    def should_raises_non_such_key_exception(self):
        with pytest.raises(NoSuchKey):
            _ = string_from_key(
                dictionary=self.dictionary,
                key="invalid_key"
            )

    def should_raises_type_mismatch_exception(self):
        with pytest.raises(TypeMismatch):
            value = string_from_key(
                dictionary=self.dictionary,
                key="key2"
            )
        assert not isinstance(value, str)
```

```
@pytest.mark.unit
class TestExampleWithMeiga:
    dictionary = {"key1": "value", "key2": 2}

    def should_return_a_str(self):
        result = string_from_key(
            dictionary=self.dictionary,
            key="key1"
        )
        result.assert_success(value_is_instance_of=str)

    def should_raises_non_such_key_exception(self):
        result = string_from_key(
            dictionary=self.dictionary,
            key="invalid_key"
        )
        result.assert_failure(value_is_instance_of=NoSuchKey)

    def should_raises_type_mismatch_exception(self):
        result = string_from_key(
            dictionary=self.dictionary,
            key="key2"
        )
        result.assert_failure(value_is_instance_of=TypeMismatch)
```

# Testing

```
@pytest.mark.unit
class TestExampleWithoutMeiga:
    dictionary = {"key1": "value", "key2": 2}

    def should_return_a_str(self):
        value = string_from_key(
            dictionary=self.dictionary,
            key="key1"
        )
        assert isinstance(value, str)

    def should_raises_non_such_key_exception(self):
        with pytest.raises(NoSuchKey):
            _ = string_from_key(
                dictionary=self.dictionary,
                key="invalid_key"
            )

    def should_raises_type_mismatch_exception(self):
        with pytest.raises(TypeMismatch):
            value = string_from_key(
                dictionary=self.dictionary,
                key="key2"
            )
        assert not isinstance(value, str)
```

```
@pytest.mark.unit
class TestExampleWithMeiga:
    dictionary = {"key1": "value", "key2": 2}

    def should_return_a_str(self):
        result = string_from_key(
            dictionary=self.dictionary,
            key="key1"
        )
        result.assert_success(
            value_is_instance_of=str,
            value_is_equal_to="value"
        )

    def should_raises_non_such_key_exception(self):
        result = string_from_key(
            dictionary=self.dictionary,
            key="invalid_key"
        )
        result.assert_failure(value_is_instance_of=NoSuchKey)

    def should_raises_type_mismatch_exception(self):
        result = string_from_key(
            dictionary=self.dictionary,
            key="key2"
        )
        result.assert_failure(value_is_instance_of=TypeMismatch)
```

# Testing

```
def should_fail_when_app_service_error(self) → None:                                Arrange
    self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
    self.mock_app_service.execute = Mock(return_value=Failure(AppServiceError()))

use_case = UserUpdater(self.mock_repository, self.mock_app_service)                  Act
result = use_case.execute(UuidMother.any())

result.assert_failure(value_is_instance_of=AppServiceError())
self.mock_repository.retrieve.assert_called_once()
self.mock_app_service.execute.assert_called_once()                                  Assert
```

# ¿Por qué utilizar móndas?

- ✓ Nos ayuda a tener un mejor tipado.
- ✓ Nos facilita el testing.
- Nos habilita a usar funciones derivadas de la programación funcional.  
Nos ayuda a crear "código limpio".

# Programación funcional

Result

meiga

Overview

Install

Getting started

Usage

Result

Alias

Decorators

Assertions

Contributing to meiga

Changelog

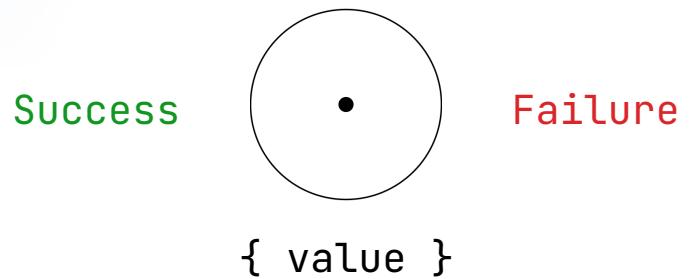
Functions

Functions	Definition
<code>unwrap()</code>	Returns the encapsulated value if this instance is a success or None if it is failure.
<code>unwrap_or_raise()</code>	Returns the encapsulated value if this instance is a success or raise the encapsulated exception if it is failure.
<code>unwrap_or_return()</code>	Returns the encapsulated value if this instance is a success or return Result as long as <code>@early_return</code> decorator wraps the function.
<code>unwrap_or(failure_value)</code>	Returns the encapsulated value if this instance is a success or the selected <code>failure_value</code> if it is failure.
<code>reraise()</code>	Raises the encapsulated failure value if this instance derive from Error or BaseException.
<code>map()</code>	Modifies encapsulate value applying a mapper function.
<code>unwrap_or_else(on_failure_handler)</code>	Returns the encapsulated value if this instance is a success or execute the <code>on_failure_handler</code> when it is failure.
<code>unwrap_and(on_success_handler)</code>	Returns the encapsulated value if this instance is a success and execute the <code>on_success_handler</code> when it is success.
<code>handle(on_success_handler, on_failure_handler)</code>	Returns itself and execute the <code>on_success_handler</code> when the instance is a success and the <code>on_failure_handler</code> when it is failure.
<code>bind(func)</code>	Returns itself binding success value with input func
<code>transform()</code>	Transform the result with a transformer function. You can give the transformer callable or use the <code>set_transformer</code> function to pre-set the callable to be used.

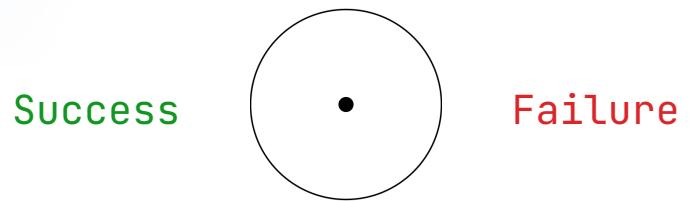
Table of contents

- Functions
- Properties
- Introduction
- Detail
  - `unwrap`
  - `unwrap_or_raise`
  - `unwrap_or_return`
  - `unwrap_or`
  - `reraise`
  - `map`
  - `unwrap_or_else`
  - `unwrap_and`
  - `handle`
  - `bind`
  - `transform`
  - `match`
- Deprecated  $\triangle$ 
  - `throw`
  - `unwrap_or_throw`

# bind



# bind



`f( { value } )` if Success

# bind

```
user = {  
    "name": "rosalia de castro",  
    "age": 186  
}
```

*f( user )* Nombre en Mayúsculas



*f( user )* Aniversario 🎉



*f( user )* E ti, de onde ves sendo?



# bind

```
user = {  
    "name": "rosalia de castro",  
    "age": 186  
}
```

*f( user )* Nombre en Mayúsculas



*f( user )* Aniversario 🎉



*f( user )* E ti, de onde ves sendo?

```
user = {  
    "name": "ROSALIA DE CASTRO",  
    "age": 187,  
    "location": "GALIZA",  
}
```

# bind

```
user = {  
    "name": "rosalia de castro",  
    "age": 186  
}
```

*f( user )* Nombre en Mayúsculas



*f( user )* Aniversario 🎉



*f( user )* E ti, de onde ves sendo?

```
from typing import Any  
  
from meiga import Result, Success, Error  
  
def upper_name(value: dict) → dict:  
    value.update({"name": value["name"].upper()})  
    return value  
  
def update_age(value: dict) → dict:  
    value.update({"age": value["age"] + 1})  
    return value  
  
def add_location(value: dict) → dict:  
    value.update({"location": "GALIZA"})  
    return value  
  
def update(user: dict) → dict:  
    if user:  
        user = upper_name(user)  
    if user:  
        user = update_age(user)  
    if user:  
        user = add_location(user)  
    return user
```



# bind

```
user = {  
    "name": "rosalia de castro",  
    "age": 186  
}  
check .....
```

f( user ) Nombre en Mayúsculas

check .....

f( user ) Aniversario 🎉

check .....

f( user ) E ti, de onde ves sendo?

```
from typing import Any  
  
from meiga import Result, Success, Error  
  
def upper_name(value: dict) → dict:  
    value.update({"name": value["name"].upper()})  
    return value  
  
def update_age(value: dict) → dict:  
    value.update({"age": value["age"] + 1})  
    return value  
  
def add_location(value: dict) → dict:  
    value.update({"location": "GALIZA"})  
    return value  
  
def update(user: dict) → dict:  
    if user:  
        user = upper_name(user)  
    if user:  
        user = update_age(user)  
    if user:  
        user = add_location(user)  
    return user
```



# bind

```
user = {  
    "name": "rosalia de castro",  
    "age": 186  
}  
check .....
```

f( user ) Nombre en Mayúsculas

check .....

f( user ) Aniversario 🎉

check .....

f( user ) E ti, de onde ves sendo?



```
from typing import Any  
  
from meiga import Result, Success, Error  
  
def upper_name(value: dict) → dict:  
    value.update({"name": value["name"].upper()})  
    return value  
  
def update_age(value: dict) → dict:  
    value.update({"age": value["age"] + 1})  
    return value  
  
def add_location(value: dict) → dict:  
    value.update({"location": "GALIZA"})  
    return value  
  
def update(user: dict) → dict:  
    if user:  
        user = upper_name(user)  
    if user:  
        user = update_age(user)  
    if user:  
        user = add_location(user)  
    return user
```



# bind

```
Success({  
  "name": "rosalia de castro",  
  "age": 186  
})
```

*f( user )* Nombre en Mayúsculas



*f( user )* Aniversario 🎉



*f( user )* E ti, de onde ves sendo?



```
Success({  
  "name": "ROSALIA DE CASTRO",  
  "age": 187,  
  "location": "GALIZA",  
})
```

# bind

```
Success({  
    "name": "rosalia de castro",  
    "age": 186  
})
```

*f( user )* Nombre en Mayúsculas

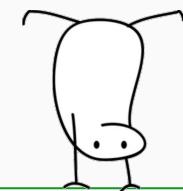


*f( user )* Aniversario 🎉

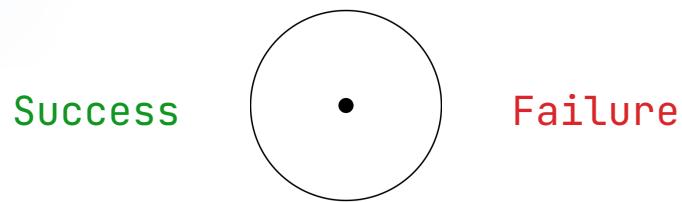


*f( user )* E ti, de onde ves sendo?

```
from typing import Any  
  
from meiga import Result, Success, Error  
  
def upper_name(value: dict) → dict:  
    value.update({"name": value["name"].upper()})  
    return value  
  
def update_age(value: dict) → dict:  
    value.update({"age": value["age"] + 1})  
    return value  
  
def add_location(value: dict) → dict:  
    value.update({"location": "GALIZA"})  
    return value  
  
def update(result: Result[dict, Error]) → Result[dict, Error]:  
    return (  
        result  
        .bind(upper_name)  
        .bind(update_age)  
        .bind(add_location)  
    )
```



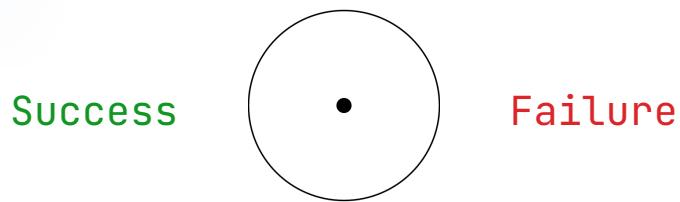
# handle



*f( { value }, args ) if Success*

*f( { value }, args ) if Failure*

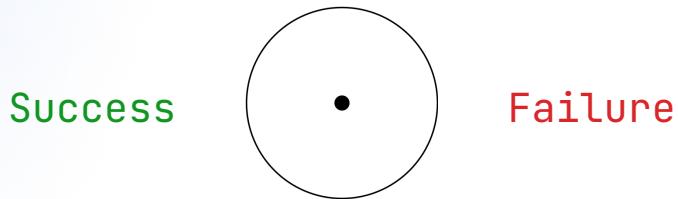
# handle



`f( { value }, args ) if Success → Publish to RabbitMQ` 

`f( { value }, args ) if Failure → Notify to Slack` 

# handle



`f( { value }, args )` if Success

`f( { value }, args )` if Failure

```
.handle(  
    on_success_handler: OnSuccessHandler(func=f, args=(arg1, arg2)),  
    on_failure_handler: OnFailureHandler(func=f, args=(arg1, arg2))  
)
```

# handle

Success

Failure

*f( { value }, args ) if Success*

*f( { value }, args ) if Failure*

*.handle(*

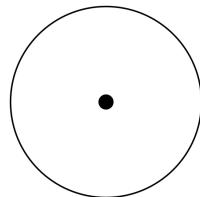
*on\_success\_handler: OnSuccessHandler(func=f, args=(arg1, arg2)),*

*on\_failure\_handler: OnFailureHandler(func=f, args=(arg1, arg2))*

*)*

# handle

Success



Failure

*f( { value }, args ) if Success*

*f( { value }, args ) if Failure*

```
.handle(  
    on_success_handler: OnSuccessHandler(func=f, args=(arg1,  
    on_failure_handler: OnFailureHandler(func=f, args=(arg1,  
))
```

# handle

```
from meiga import OnSuccessHandler, OnFailureHandler

def publish_event(domain_event_bus: DomainEventBus):
    domain_event_bus.publish(ValueRetrieved())

def notify_error(notifier: Notifier, use_case: str):
    notifier.post(use_case, "Value cannot be retrieved")

result = string_from_key(dictionary=user, key="key1")

result.handle(
    on_success_handler=OnSuccessHandler(
        func=publish_event, args=(domain_event_bus,)
    ),
    on_failure_handler=OnFailureHandler(
        func=notify_error, args=(notifier, "PyCon ES")
    )
)
```

# handle

```
from meiga import OnSuccessHandler, OnFailureHandler

def publish_event(domain_event_bus: DomainEventBus):
    domain_event_bus.publish(ValueRetrieved())

def notify_error(notifier: Notifier, use_case: str):
    notifier.post(use_case, "Value cannot be retrieved")

result = string_from_key(dictionary=user, key="key1")

result.handle(
    on_success_handler=OnSuccessHandler(
        func=publish_event, args=(domain_event_bus,))
),
    on_failure_handler=OnFailureHandler(
        func=notify_error, args=(notifier, "PyCon ES"))
)
```



# handle

```
from meiga import OnSuccessHandler, OnFailureHandler

def publish_event(domain_event_bus: DomainEventBus):
    domain_event_bus.publish(ValueRetrieved())

def notify_error(notifier: Notifier, use_case: str):
    notifier.post(use_case, "Value cannot be retrieved")

result = string_from_key(dictionary=user, key="key1")

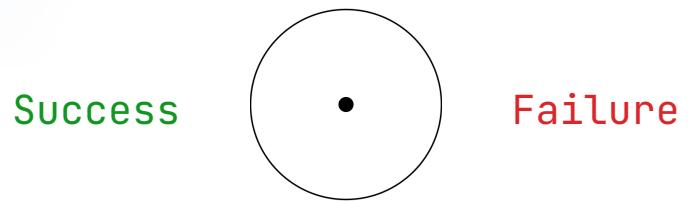
result.handle(
    on_success_handler=OnSuccessHandler(
        func=publish_event, args=(domain_event_bus,),
    ),
    on_failure_handler=OnFailureHandler(
        func=notify_error, args=(notifier, "PyCon ES"),
    )
)
```



PyCon ES APP 7:37 PM

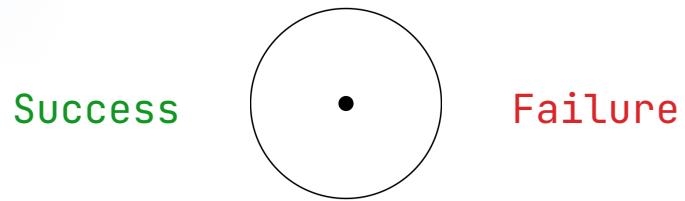
Value cannot be retrieved

# map



`f( { value } ) if Success or Failure`

# transformer



*f( Result ) if Success or Failure*

# map

```
def mapper(value: str):
    if not isinstance(value, str):
        return value
    return value.capitalize()

result = string_from_key(dictionary, key)

result.map(mapper)
```

# transformer

```
def transformer(result: Result) → tuple[int, str]:
    match result:
        case Success(value):
            return 200, value.capitalize()
        case Failure(NoSuchKey):
            raise 404, "Key not found"
        case Failure(TypeMismatch):
            raise 422, "Given key doesn't have a str"

result = string_from_key(dictionary, key)
status_code, message = result.transform(transformer)
```

# map

```
def mapper(value: str):
    if not isinstance(value, str):
        return value
    return value.capitalize()

result = string_from_key(dictionary, key)

result.map(mapper)
```

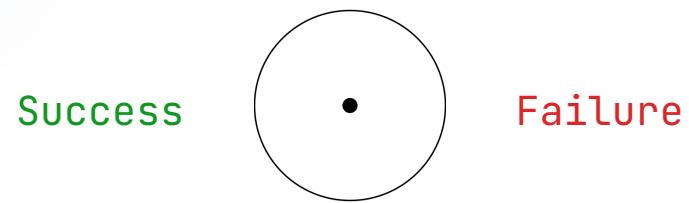
# transformer

```
def my_controller() -> Result[str, Error]:
    def transformer(result: Result) -> tuple[int, str]:
        match result:
            case Success(value):
                return 200, value.capitalize()
            case Failure(NoSuchKey):
                raise 404, "Key not found"
            case Failure(TypeMismatch):
                raise 422, "Given key doesn't have a str"

        result = string_from_key(dictionary, key)
        result.set_transformer(transformer)
        return result

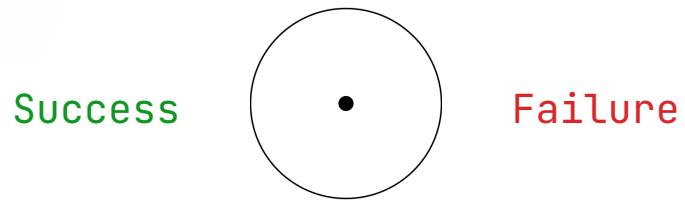
    status_code, message = my_controller.transform()
```

# unwrap



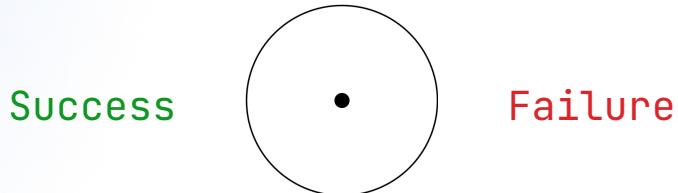
```
{ value } if Success  
None     if Failure
```

# unwrap\_or



```
{ value } if Success  
<given value> if Failure
```

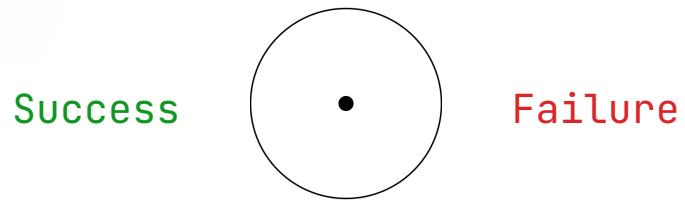
# unwrap\_or



{ value } if Success  
<*given value*> if Failure

```
users = fetch_usernames().unwrap_or([])  
for user in users:  
    notify(user)
```

# unwrap\_or\_return



{ value } if Success

*force return* if Failure

# unwrap\_or\_return

```
from meiga import Result, isSuccess, early_return

def CreateUser(UseCase)
    sanitizer: UserSanitizer
    repository: UserRepository
    bus: DomainEventBus

    @early_return
    def execute(self, user: User) → Result[bool, InvalidInput | UserAlreadyExist]:
        sanitized_user = self.sanitizer.execute(user).unwrap_or_return()
        self.repository.save(sanitized_user).unwrap_or_return()
        self.bus.publish(UserCreated()).unwrap_or_return()
        return isSuccess
```

# unwrap\_or\_return

```
from meiga import Result, isSuccess, early_return

def CreateUser(UseCase)
    sanitizer: UserSanitizer
    repository: UserRepository
    bus: DomainEventBus

    @early_return
    def execute(self, user: User) → Result[bool, InvalidInput | UserAlreadyExist]:
        sanitized_user = self.sanitizer.execute(user).unwrap_or_return()
        sanitized_user: User = self.sanitizer.execute(..).unwrap_or_return()
        self.bus.publish(UserCreated(...)).unwrap_or_return()
        return isSuccess
```

# unwrap\_or\_return

```
from meiga import Result, isSuccess, early_return

def CreateUser(UseCase)
    sanitizer: UserSanitizer
    repository: UserRepository
    bus: DomainEventBus

    @early_return
    def execute(self, user: User) → Result[bool, InvalidInput | UserAlreadyExist]:
        sanitized_user = self.sanitizer.execute(user).unwrap_or_return()
        self.repository.save(sanitized_user)
        self.bus.publish(UserCreated(sanitized_user))
        return isSuccess
```

# unwrap\_or\_return

```
from meiga import Result, isSuccess, early_return

def CreateUser(UseCase)
    sanitizer: UserSanitizer
    repository: UserRepository
    bus: DomainEventBus

    @early_return
    def execute(self, user: User) → Result[bool, InvalidInput | UserAlreadyExist]:
        sanitized_user = self.sanitizer.execute(user).unwrap_or_return()
        self.repository.save(sanitized_user).unwrap_or_return()
        self.bus.publish(UserCreated()).unwrap_or_return()
        return isSuccess
```

OnFailureException(InvalidInput())

← @early\_return

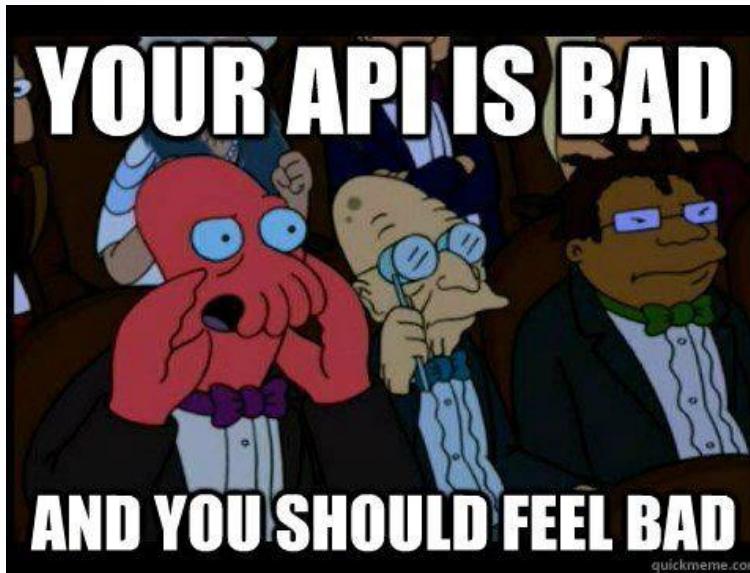
← self.repository.save(sanitized\_user).unwrap\_or\_return()

# ¿Por qué utilizar mónadas?

- ✓ Nos ayuda a tener un mejor tipado.
- ✓ Nos facilita el testing.
- ✓ Nos habilita a usar funciones derivadas de la programación funcional.
- Nos ayuda a crear "código limpio".

# Dónde nos ayuda realmente

A la hora de desarrollar aplicaciones web en backend.



# Dónde nos ayuda realmente

A la hora de desarrollar aplicaciones web en backend.

- ✓ Nos ayuda a tener un mejor tipado.
- ✓ Nos facilita el testing.

Mapear los errores de dominio a errores HTTP.

Nos reduce el ruido visual de los casos de uso.

# Dónde nos ayuda realmente

A la hora de desarrollar aplicaciones web en backend.

Nuestro stack



petisco A small orange cookie icon with a bite taken out of it.

meiga A small blue and yellow cartoon character wearing a pointed hat and a blue coat.

# Dónde nos ayuda realmente

A la hora de desarrollar aplicaciones web en backend.

Nuestro stack



petisco 🍪



```
pip install petisco[fastapi] ⭐
```

<https://github.com/alice-biometrics/petisco>

## Nuestro stack





GET

/acknowledge/{conference}

Application

⚡ FastAPI

```
from fastapi import FastAPI
from petisco.extra.fastapi import as_fastapi

app = FastAPI()

@app.get("acknowledge/{conference}")
def get_acknowledge(conference: str) -> HTML:
    result = GetAcknowledgeController().execute(conference)
    return as_fastapi(result)
```



GET

/acknowledge/{conference}

Application

Controller ➤ Use Case

```
class GetAcknowledgeController(Controller):

    class Config:
        error_map = {
            OrganizersNotFound: HttpError(status_code=404, detail="Organizer not found"),
            SponsorsNotFound: HttpError(status_code=404, detail="Sponsors not found"),
            MessageTooLarge: HttpError(status_code=422, detail="The html message exceeds the limits"),
        }

    def execute(self, conference: str) → Result[dict, OrganizersNotFound | SponsorsNotFound | MessageTooLarge]:
        return AcknowledgeCreator(
            organizers_repository=Container.get(OrganizersRepository),
            sponsors_repository=Container.get(SponsorsRepository),
            html_formatter=Container.get(HtmlFormatter)
        ).execute(conference)
```



GET  
`/acknowledge/{conference}`

Application

Controller ➤ Use Case

```
class GetAcknowledgeController(Controller):

    class Config:
        error_map = {
            OrganizersNotFound: HttpError(status_code=404, detail="Organizer not found"),
            SponsorsNotFound: HttpError(status_code=404, detail="Sponsors not found"),
            MessageTooLarge: HttpError(status_code=422, detail="The html message exceeds the limits"),
        }

    def execute(self, conference: str) → Result[dict, OrganizersNotFound | SponsorsNotFound | MessageTooLarge]:
        return AcknowledgeCreator(
            organizers_repository=Container.get(OrganizersRepository),
            sponsors_repository=Container.get(SponsorsRepository),
            html_formatter=Container.get(HtmlFormatter)
        ).execute(conference)
```

Instanciar un Caso de Uso



GET  
`/acknowledge/{conference}`

Application

Controller ➤ Use Case

```
class GetAcknowledgeController(Controller):

    class Config:
        error_map = {
            OrganizersNotFound: HttpError(status_code=404, detail="Organizer not found"),
            SponsorsNotFound: HttpError(status_code=404, detail="Sponsors not found"),
            MessageTooLarge: HttpError(status_code=422, detail="The html message exceeds the limits"),
        }

    def execute(self, conference: str) → Result[dict, OrganizersNotFound | SponsorsNotFound | MessageTooLarge]:
        return AcknowledgeCreator(
            organizers_repository=Container.get(OrganizersRepository),
            sponsors_repository=Container.get(SponsorsRepository),
            html_formatter=Container.get(HtmlFormatter)
        ).execute(conference)
```

Injecta las dependencias



GET

/acknowledge/{conference}

Application

Controller ➤ Use Case

```
class GetAcknowledgeController(Controller):

    class Config:
        error_map = {
            OrganizersNotFound: HttpError(status_code=404, detail="Organizer not found"),
            SponsorsNotFound: HttpError(status_code=404, detail="Sponsors not found"),
            MessageTooLarge: HttpError(status_code=422, detail="The html message exceeds the limits"),
        }

    def execute(self, conference: str) → Result[dict, OrganizersNotFound | SponsorsNotFound | MessageTooLarge]:
        return AcknowledgeCreator(
            organizers_repository=Container.get(OrganizersRepository),
            sponsors_repository=Container.get(SponsorsRepository),
            html_formatter=Container.get(HtmlFormatter)
        ).execute(conference)
```





GET

/acknowledge/{conference}

Application

Controller ➤ Use Case

```
class GetAcknowledgeController(Controller):

    class Config:
        error_map = {
            OrganizersNotFound: HttpError(status_code=404, detail="Organizer not found"),
            SponsorsNotFound: HttpError(status_code=404, detail="Sponsors not found"),
            MessageTooLarge: HttpError(status_code=422, detail="The html message exceeds the limits"),
        }

    def execute(self, conference: str) → Result[dict, OrganizersNotFound | SponsorsNotFound | MessageTooLarge]:
        return AcknowledgeCreator(
            organizers_repository=Container.get(OrganizersRepository),
            sponsors_repository=Container.get(SponsorsRepository),
            html_formatter=Container.get(HtmlFormatter)
        ).execute(conference)
```



GET

/acknowledge/{conference}

Application

Controller ➤ Use Case

```
class GetAcknowledgeController(Controller):

    class Config:
        error_map = {
            OrganizersNotFound: HttpError(status_code=404, detail="Organizer not found"),
            SponsorsNotFound: HttpError(status_code=404, detail="Sponsors not found"),
            MessageTooLarge: HttpError(status_code=422, detail="The html message exceeds the limits"),
        }

    def execute(self, conference: str) → Result[dict, OrganizersNotFound | SponsorsNotFound | MessageTooLarge]:
        return AcknowledgeCreator(
            organizers_repository=Container.get(OrganizersRepository),
            sponsors_repository=Container.get(SponsorsRepository),
            html_formatter=Container.get(HtmlFormatter)
        ).execute(conference)
```



GET  
`/acknowledge/{conference}`

Application

Controller ➤ Use Case

```
from meiga import Result, Error
from petisco import UseCase

class OrganizersNotFound(Error): ...
class SponsorsNotFound(Error): ...
class MessageTooLarge(Error): ...


class AcknowledgeCreator(UseCase):
    organizers_repository: OrganizersRepository
    sponsors_repository: SponsorsRepository
    html_formatter: HtmlFormatter

    def execute(
        self, conference: str
    ) -> Result[HTML, OrganizersNotFound | SponsorsNotFound | MessageTooLarge]:
        organizers = self.organizers_repository.retrieve(conference).unwrap_or_return()
        sponsors = self.sponsors_repository.retrieve(conference).unwrap_or_return()
        return self.html_formatter.execute(organizers, sponsors)
```



GET

/acknowledge/{conference}

Application

Controller ➤ Use Case

```
from meiga import Result, Error
from petisco import UseCase

class OrganizersNotFound(Error): ...
class SponsorsNotFound(Error): ...
class MessageTooLarge(Error): ...


class AcknowledgeCreator(UseCase):
    organizers_repository: OrganizersRepository
    sponsors_repository: SponsorsRepository
    html_formatter: HtmlFormatter

    def execute(
        self, conference: str
    ) -> Result[HTML, OrganizersNotFound | SponsorsNotFound | MessageTooLarge]:
        organizers = self.organizers_repository.retrieve(conference).unwrap_or_return()
        self.sponsors_repository.retrieve(conference).unwrap_or_return()
        organizers: list[Organizer] = self.organizers_repository..isors)
```



GET

/acknowledge/{conference}

Application

Controller ➤ Use Case

```
from meiga import Result, Error
from petisco import UseCase

class OrganizersNotFound(Error): ...
class SponsorsNotFound(Error): ...
class MessageTooLarge(Error): ...


class AcknowledgeCreator(UseCase):
    organizers_repository: OrganizersRepository
    sponsors_repository: SponsorsRepository
    html_formatter: HtmlFormatter

    def execute(
        self, conference: str
    ) -> Result[HTML, OrganizersNotFound | SponsorsNotFound | MessageTooLarge]:
        organizers = self.organizers_repository.retrieve(conference).unwrap_or_return()
        sponsors = self.sponsors_repository.retrieve(conference).unwrap_or_return()
        return self.def retrieve(self, conference: str) -> Result[list[Organizer], OrganizersNotFound]
```



GET

/acknowledge/{conference}

Application

Controller ➤ Use Case

```
from meiga import Result, Error
from petisco import UseCase

class OrganizersNotFound(Error): ...
class SponsorsNotFound(Error): ...
class MessageTooLarge(Error): ...


class AcknowledgeCreator(UseCase):
    organizers_repository: OrganizersRepository
    sponsors_repository: SponsorsRepository
    html_formatter: HtmlFormatter

    def execute(
        self, conference: str
    ) -> Result[HTML, OrganizersNotFound | SponsorsNotFound | MessageTooLarge]:
        organizers = self.organizers_repository.retrieve(conference).unwrap_or_return()
        sponsors = self.sponsors_repository.retrieve(conference).unwrap_or_return()
        <-- html_formatter.execute(organizers, sponsors)
        sponsors: list[Sponsor] = self.sponsors_repository..
```



GET  
`/acknowledge/{conference}`

Application

Controller ➤ Use Case

```
from meiga import Result, Error
from petisco import UseCase

class OrganizersNotFound(Error): ...
class SponsorsNotFound(Error): ...
class MessageTooLarge(Error): ...


class AcknowledgeCreator(UseCase):
    organizers_repository: OrganizersRepository
    sponsors_repository: SponsorsRepository
    html_formatter: HtmlFormatter

    def execute(
        self, conference: str
    ) -> Result[HTML, OrganizersNotFound | SponsorsNotFound | MessageTooLarge]:
        organizers = self.organizers_repository.retrieve(conference).unwrap_or_return()
        sponsors = self.sponsors_repository.retrieve(conference).unwrap_or_return()
        return self._build_email(organizers, sponsors)

    def retrieve(self, conference: str) -> Result[List[Sponsor], SponsorsNotFound]
```



GET

/acknowledge/{conference}

Application

Controller ➤ Use Case

```
from meiga import Result, Error
from petisco import UseCase

class OrganizersNotFound(Error): ...
class SponsorsNotFound(Error): ...
class MessageTooLarge(Error): ...


class AcknowledgeCreator(UseCase):
    organizers_repository: OrganizersRepository
    sponsors_repository: SponsorsRepository
    html_formatter: HtmlFormatter

    def execute(
        self, conference
    ) -> Result[HTML]:
        organizers = self.organizers_repository.find_all_by_conference(conference)
        sponsors = self.sponsors_repository.find_all_by_conference(conference)

        if len(organizers) == 0:
            raise OrganizersNotFound()

        if len(sponsors) == 0:
            raise SponsorsNotFound()

        if len(organizers) * len(sponsors) > 1000000:
            raise MessageTooLarge()

        return self.html_formatter.execute(organizers, sponsors)
```



GET

/acknowledge/{conference}

Application



GET

/acknowledge/{conference}



Application

1. Recupera los Organizadores.
2. Recupera los Patrocinadores.
3. Crea un HTML para presentar la info.



GET /acknowledge/pycones23

Application

The screenshot shows a web browser window with the URL [meiga.gal/conference/pycones23](https://meiga.gal/conference/pycones23). The page header includes the PyConES Canarias 2023 logo, navigation links for Programa, Ciudad, Viaje, Patrocinios, Diversidad y Becas, Ofertas de Trabajo, FAQ, and Entradas, and language links for EN and ES.

## Equipo Organizador

La edición 2023 de la PyConES será posible gracias a muchas personas voluntarias que provienen de comunidades tecnológicas como **Python Canarias**, **Python España**, **Ada Lovers**, **PyLadies España** entre otras:



Cristián Maureira-Fredes

Senior R&D Manager



[Python España](#) [Programa](#) [Patrocinios](#) [Web](#)



Juan Ignacio Rodríguez de León

Python Developer



[Python Canarias](#) [Infraestructura](#) [Web](#)



Andrés Orcajo

Senior Python Developer





GET /acknowledge/pycones23

Application

The screenshot shows a web browser window with the URL [meiga.gal/conference/pycones23](https://meiga.gal/conference/pycones23). The page is titled "PyConES Canarias 2023". The main content area lists five volunteers with their profiles and social media links:

- Victor Vicente-Palacios**  
Clinical Data Scientist  
Programa, Diversidad  
Social media: GitHub, LinkedIn, Twitter, another platform
- Nazaret Miranda López**  
Software Developer  
AdaLoveDev, Voluntariado  
Social media: LinkedIn
- Jose Alberto Torres Aguera**  
Technical Leader  
Python Málaga, Infraestructura, Patrocinios, Web  
Social media: GitHub, Twitter
- Esther**  
Software Developer  
Voluntariado  
Social media: GitHub, LinkedIn, Twitter
- Jesús Torres Jorge**  
Profesor y Director Académico  
Python Canarias, Diversidad, Infraestructura, Patrocinios, Voluntariado  
Social media: GitHub, LinkedIn, Twitter



GET /acknowledge/pycones23

Application

The screenshot shows a web browser displaying the PyConES Canarias 2023 website at [meiga.gal/conference/pycones23](https://meiga.gal/conference/pycones23). The page features a purple header with navigation links: Programa, Ciudad, Viaje, Patrocinios, Diversidad y Becas, Ofertas de Trabajo, FAQ, and Entradas. On the left, the PyConES Canarias 2023 logo is visible. The main content area displays five speaker profiles with their names, titles, and social media links.

Speaker	Title	Topics	Social Media
Álex Samarin	Technical Lead	Python Canarias, Infraestructura, Voluntariado	<a href="#">Twitter</a>
Lucia Cabrera Garabote	Estudiante de Ingeniería Informática	Patrocinios, Voluntariado	<a href="#">Twitter</a> , <a href="#">Instagram</a>
Yodra López Herrera	Software Developer	AdaLoveDev, Programa, Diversidad, Infraestructura, Patrocinios, Voluntariado	<a href="#">Twitter</a> , <a href="#">LinkedIn</a>
Fran Escobar González	Senior Backend developer	Python Canarias, Programa, Patrocinios	<a href="#">Twitter</a>
Johanna Sanchez	Fullstack Developer y Química	Python España, Diversidad, Redes sociales, Patrocinios, Web	<a href="#">Twitter</a> , <a href="#">LinkedIn</a>



GET /acknowledge/pycones23

Application

The screenshot shows a web browser window with the following details:

- Address Bar:** meiga.gal/conference/pycones23
- Header:** PyConES Canarias 2023, Programa, Ciudad, Viaje, Patrocinios, Diversidad y Becas, Ofertas de Trabajo, FAQ, Entradas.
- Content:** A list of acknowledgments for PyConES 2023, each with a profile picture, name, title, and social media links.

Profile Picture	Name	Title	Social Media Links
	Johanna Sanchez	Fullstack Developer y Química	
	Silvia García Hernández	Estudiante	
	Javier Alonso Silva	Ingeniero I+D en Teldat	
	Sergio Delgado Quintero	Ingéniero Informático y Profesor en FP	
	Jimena E. Bermúdez	Ingeniera de Software	



GET /acknowledge/pycones23

Application

The screenshot shows a web browser displaying the PyConES Canarias 2023 website at [meiga.gal/conference/pycones23](http://meiga.gal/conference/pycones23). The page has a purple header with navigation links: Programa, Ciudad, Viaje, Patrocinios, Diversidad y Becas, Ofertas de Trabajo, FAQ, and Entradas. The main content area features a yellow background with a green diagonal shape on the left. The title 'Patrocinios' is displayed in bold black font. Below it is a text block: 'Gracias a las empresas que colaboran con la PyConES podemos ofrecer el mejor evento y experiencia posible. Somos una conferencia con un bajo coste de entrada capaz de ofrecer una experiencia de 3 días incluyendo regalos, almuerzos, comidas y meriendas. Con la ayuda de estas empresas conseguimos hacer un evento diverso e inclusivo enfocado en cuidar la comunidad de Python.' A section titled 'Comunidades Patrocinadoras' is shown with three logos: Python Software Foundation, Python España, and Nivel Teide. Below this are partial views of other sponsors like ansel and bluetab.

PyConES  
Canarias  
2023

Programa Ciudad Viaje Patrocinios Diversidad y Becas Ofertas de Trabajo FAQ Entradas

## Patrocinios

Gracias a las empresas que colaboran con la PyConES podemos ofrecer el mejor evento y experiencia posible. Somos una conferencia con un bajo coste de entrada capaz de ofrecer una experiencia de 3 días incluyendo regalos, almuerzos, comidas y meriendas. Con la ayuda de estas empresas conseguimos hacer un evento diverso e inclusivo enfocado en cuidar la comunidad de Python.

### Comunidades Patrocinadoras

Nivel Teide



GET /acknowledge/pycones23

Application

The screenshot shows a web browser displaying the PyConES Canarias 2023 website at [meiga.gal/conference/pycones23](https://meiga.gal/conference/pycones23). The page is titled "Patrocinios". A banner message states: "Gracias a las empresas que colaboran con la PyConES podemos ofrecer el mejor evento y experiencia posible. Somos una conferencia con un bajo coste de entrada capaz de ofrecer una experiencia de 3 días incluyendo regalos, almuerzos, comidas y meriendas. Con la ayuda de estas empresas conseguimos hacer un evento diverso e inclusivo enfocado en cuidar la comunidad de Python." Below this, there is a section titled "Comunidades Patrocinadoras" featuring logos for Python Software Foundation, Python España, and Nivel Teide. Further down, there are logos for ANSL and Bluetab.

PyConES  
Canarias  
2023

Programa Ciudad Viaje Patrocinios Diversidad y Becas Ofertas de Trabajo FAQ Entradas

## Patrocinios

Gracias a las empresas que colaboran con la PyConES podemos ofrecer el mejor evento y experiencia posible. Somos una conferencia con un bajo coste de entrada capaz de ofrecer una experiencia de 3 días incluyendo regalos, almuerzos, comidas y meriendas. Con la ayuda de estas empresas conseguimos hacer un evento diverso e inclusivo enfocado en cuidar la comunidad de Python.

### Comunidades Patrocinadoras

python SOFTWARE FOUNDATION Python España

Nivel Teide

ANSI /bluetab



GET /acknowledge/pycones23

Application

The screenshot shows the PyConES 2023 conference website. At the top, there's a navigation bar with links for Programa, Ciudad, Viaje, Patrocinios, Diversidad y Becas, Ofertas de Trabajo, FAQ, and Entradas. Below the navigation, there are three sections of sponsor logos:

- Nivel Tamadaba:** SKYDANCE ANIMATION, octopusenergy, CEPSA, and pwc.
- Nivel Teneguía:** INDITEX TECH, Ebury (with tagline "What borders?"), and Bloomberg.
- Nivel Timanfaya:** (partially visible at the bottom)

Each sponsor logo is displayed within a white rectangular box. The background of the page features a yellow-to-purple gradient and some abstract shapes.



GET /acknowledge/pycones23

Application

The screenshot shows a web browser displaying the PyConES 2023 conference website at [meiga.gal/conference/pycones23](http://meiga.gal/conference/pycones23). The page features a purple header with the PyConES Canarias 2023 logo and navigation links for Programa, Ciudad, Viaje, Patrocinios, Diversidad y Becas, Ofertas de Trabajo, FAQ, and Entradas. Below the header, there are four boxes for special sponsors: Instituto de Astrofísica de Canarias (IAC), Qt, EdgeTier, and Multiverse Computing. Under the 'Patrocinio Especial' section, logos for Universidad de La Laguna and Eventbrite are shown. Under the 'Colaboradores' section, logos for PCTT, Cabildo de Tenerife, tenerife innova!, E-CAN, tcb!, and GitHub are displayed.

PyConES  
Canarias  
2023

Programa Ciudad Viaje Patrocinios Diversidad y Becas Ofertas de Trabajo FAQ Entradas

Patrocinio Especial

Colaboradores





## Teorema de Pitágoras encontrado en una tablilla de arcilla 1.000 años más antigua que Pitágoras

Es anterior a Pitágoras en más de 1.000 años.



JAMES FELTON

Redactor sénior



La prueba está tallada en arcilla.

Crédito de la imagen: Osama Shukir Muhammed Amin FRCP (Glasg) vía [Wikimedia Commons \(CC BY-SA 4.0\)](#); girado, recortado

“

**“La conclusión es ineludible. Los babilonios conocían la relación entre la longitud de la diagonal de un cuadrado y su lado:  $d=\sqrt{2}$ ”,** escribe el matemático Bruce Ratner en un artículo sobre el tema . “Este fue



En  Alice estamos contratando  
**Frontend | ML Engineer**

# Grazas!





# Testing

```
class UserUpdater(UseCase):
    repository: Repository
    app_service: AppService

    def execute(self, user_id: Uuid) → Result[bool, Error]:
        user = self.repository.retrieve(user_id).unwrap_or_return()
        return self.app_service.execute(user)
```

# Testing

```
class UserUpdater(UseCase):
    repository: Repository
    app_service: AppService

    def execute(self, user_id: Uuid) → Result[bool, Error]:
        user = self.repository.retrieve(user_id).unwrap_or_return()
        return self.app_service.execute(user)
```

```
@pytest.mark.unit
class TestUserUpdater:
    mock_repository: Mock
    mock_app_service: Mock

    def setup_method(self) → None:
        self.mock_repository = Mock(Repository)
        self.mock_app_service = Mock(AppService)

    def should_success(self) → None:
        self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
        self.mock_app_service.execute = Mock(return_value=Success(True))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UidMother.any())

        result.assert_success()
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_called_once()

    def should_fail_when_repository_not_found_user(self) → None:
        self.mock_repository.retrieve = Mock(return_value=Failure(UserNotFoundError()))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UidMother.any())

        result.assert_failure(value_is_instance_of=UserNotFoundError)
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_not_called()

    def should_fail_when_app_service_error(self) → None:
        self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
        self.mock_app_service.execute = Mock(return_value=Failure(AppServiceError()))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UidMother.any())

        result.assert_failure(value_is_instance_of=AppServiceError())
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_called_once()
```

# Testing

```
class UserUpdater(UseCase):
    repository: Repository
    app_service: AppService

    def execute(self, user_id: Uuid) → Result[bool, Error]:
        user = self.repository.retrieve(user_id).unwrap_or_return()
        return self.app_service.execute(user)
```

**Arrange:** Configuración del test.

**Act:** Acción a testear.

**Assert:** Comprobaciones necesarias.

```
@pytest.mark.unit
class TestUserUpdater:
    mock_repository: Mock
    mock_app_service: Mock

    def setup_method(self) → None:
        self.mock_repository = Mock(Repository)
        self.mock_app_service = Mock(AppService)

    def should_success(self) → None:
        self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
        self.mock_app_service.execute = Mock(return_value=Success(True))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UuidMother.any())

        result.assert_success()
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_called_once()

    def should_fail_when_repository_not_found_user(self) → None:
        self.mock_repository.retrieve = Mock(return_value=Failure(UserNotFoundError()))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UuidMother.any())

        result.assert_failure(value_is_instance_of=UserNotFoundError)
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_not_called()

    def should_fail_when_app_service_error(self) → None:
        self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
        self.mock_app_service.execute = Mock(return_value=Failure(AppServiceError()))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UuidMother.any())

        result.assert_failure(value_is_instance_of=AppServiceError())
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_called_once()
```

# Testing

```
class UserUpdater(UseCase):
    repository: Repository
    app_service: AppService

    def execute(self, user_id: Uuid) → Result[bool, Error]:
        user = self.repository.retrieve(user_id).unwrap_or_return()
        return self.app_service.execute(user)
```

```
@pytest.mark.unit
class TestUserUpdater:
    mock_repository: Mock
    mock_app_service: Mock

    def setup_method(self) → None:
        self.mock_repository = Mock(Repository)
        self.mock_app_service = Mock(AppService)

    def should_success(self) → None:
        self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
        self.mock_app_service.execute = Mock(return_value=Success(True))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UidMother.any())

        result.assert_success()
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_called_once()

    def should_fail_when_repository_not_found_user(self) → None:
        self.mock_repository.retrieve = Mock(return_value=Failure(UserNotFound()))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UidMother.any())

        result.assert_failure(value_is_instance_of=UserNotFound)
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_not_called()

    def should_fail_when_app_service_error(self) → None:
        self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
        self.mock_app_service.execute = Mock(return_value=Failure(AppServiceError()))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UidMother.any())

        result.assert_failure(value_is_instance_of=AppServiceError())
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_called_once()
```



# Testing

```
class UserUpdater(UseCase):
    repository: Repository
    app_service: AppService

    def execute(self, user_id: Uuid) → Result[bool, Error]:
        user = self.repository.retrieve(user_id).unwrap_or_return()
        return self.app_service.execute(user)
```

```
@pytest.mark.unit
class TestUserUpdater:
    mock_repository: Mock
    mock_app_service: Mock

    def setup_method(self) → None:
        self.mock_repository = Mock(Repository)
        self.mock_app_service = Mock(AppService)

    def should_success(self) → None:
```

```
def should_success(self) → None:
```

Arrange

```
    self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
    self.mock_app_service.execute = Mock(return_value=Success(True))
```

Act

```
use_case = UserUpdater(self.mock_repository, self.mock_app_service)
result = use_case.execute(UuidMother.any())
```

Assert

```
result.assert_success()
self.mock_repository.retrieve.assert_called_once()
self.mock_app_service.execute.assert_called_once()
```

```
use_case = UserUpdater(self.mock_repository, self.mock_app_service)
result = use_case.execute(UuidMother.any())

result.assert_failure(value_is_instance_of=AppServiceError())
self.mock_repository.retrieve.assert_called_once()
self.mock_app_service.execute.assert_called_once()
```

# Testing

```
class UserUpdater(UseCase):
    repository: Repository
    app_service: AppService

    def execute(self, user_id: Uuid) → Result[bool, Error]:
        user = self.repository.retrieve(user_id).unwrap_or_return()
        return self.app_service.execute(user)
```

```
@pytest.mark.unit
class TestUserUpdater:
    mock_repository: Mock
    mock_app_service: Mock

    def setup_method(self) → None:
        self.mock_repository = Mock(Repository)
        self.mock_app_service = Mock(AppService)

    def should_success(self) → None:
        self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
        self.mock_app_service.execute = Mock(return_value=Success(True))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UidMother.any())

        result.assert_success()
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_called_once()

    def should_fail_when_repository_not_found_user(self) → None:
        self.mock_repository.retrieve = Mock(return_value=Failure(UserNotFoundError()))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UidMother.any())

        result.assert_failure(value_is_instance_of=UserNotFoundError)
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_not_called()

    def should_fail_when_app_service_error(self) → None:
        self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
        self.mock_app_service.execute = Mock(return_value=Failure(AppServiceError()))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UidMother.any())

        result.assert_failure(value_is_instance_of=AppServiceError())
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_called_once()
```



# Testing

```
class UserUpdater(UseCase):
    repository: Repository
    app_service: AppService

    def execute(self, user_id: Uuid) → Result[bool, Error]:
        user = self.repository.retrieve(user_id).unwrap_or_return()
        return self.app_service.execute(user)
```

```
@pytest.mark.unit
class TestUserUpdater:
    mock_repository: Mock
    mock_app_service: Mock

    def setup_method(self) → None:
        self.mock_repository = Mock(Repository)
        self.mock_app_service = Mock(AppService)

    def should_success(self) → None:
        ...
```

def should\_fail\_when\_repository\_not\_found\_user(self) → None:

Arrange

```
self.mock_repository.retrieve = Mock(return_value=Failure(UserNotFound()))
```

Act

```
use_case = UserUpdater(self.mock_repository, self.mock_app_service)
result = use_case.execute(UuidMother.any())
```

Assert

```
result.assert_failure(value_is_instance_of=UserNotFound)
self.mock_repository.retrieve.assert_called_once()
self.mock_app_service.execute.assert_not_called()
```

```
use_case = UserUpdater(self.mock_repository, self.mock_app_service)
result = use_case.execute(UuidMother.any())

result.assert_failure(value_is_instance_of=AppServiceError())
self.mock_repository.retrieve.assert_called_once()
self.mock_app_service.execute.assert_called_once()
```

# Testing

```
class UserUpdater(UseCase):
    repository: Repository
    app_service: AppService

    def execute(self, user_id: Uuid) → Result[bool, Error]:
        user = self.repository.retrieve(user_id).unwrap_or_return()
        return self.app_service.execute(user)
```

```
@pytest.mark.unit
class TestUserUpdater:
    mock_repository: Mock
    mock_app_service: Mock

    def setup_method(self) → None:
        self.mock_repository = Mock(Repository)
        self.mock_app_service = Mock(AppService)

    def should_success(self) → None:
        self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
        self.mock_app_service.execute = Mock(return_value=Success(True))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UidMother.any())

        result.assert_success()
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_called_once()

    def should_fail_when_repository_not_found_user(self) → None:
        self.mock_repository.retrieve = Mock(return_value=Failure(UserNotFoundError()))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UidMother.any())

        result.assert_failure(value_is_instance_of=UserNotFoundError)
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_not_called()

    def should_fail_when_app_service_error(self) → None:
        self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
        self.mock_app_service.execute = Mock(return_value=Failure(AppServiceError()))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UidMother.any())

        result.assert_failure(value_is_instance_of=AppServiceError())
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_called_once()
```



# Testing

```
class UserUpdater(UseCase):
    repository: Repository
    app_service: AppService

    def execute(self, user_id: Uuid) → Result[bool, Error]:
        user = self.repository.retrieve(user_id).unwrap_or_return()
        return self.app_service.execute(user)
```

```
@pytest.mark.unit
class TestUserUpdater:
    mock_repository: Mock
    mock_app_service: Mock

    def setup_method(self) → None:
        self.mock_repository = Mock(Repository)
        self.mock_app_service = Mock(AppService)
```

```
def should_fail_when_app_service_error(self) → None:
```

Arrange

```
self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
self.mock_app_service.execute = Mock(return_value=Failure(AppServiceError()))
```

Act

```
use_case = UserUpdater(self.mock_repository, self.mock_app_service)
result = use_case.execute(UuidMother.any())
```

Assert

```
result.assert_failure(value_is_instance_of=AppServiceError())
self.mock_repository.retrieve.assert_called_once()
self.mock_app_service.execute.assert_called_once()
```

```
self.mock_app_service.execute = Mock(return_value=Failure(AppServiceError()))
```

```
use_case = UserUpdater(self.mock_repository, self.mock_app_service)
result = use_case.execute(UuidMother.any())
```

```
result.assert_failure(value_is_instance_of=AppServiceError())
self.mock_repository.retrieve.assert_called_once()
self.mock_app_service.execute.assert_called_once()
```