LET'S BUILD AN ABSTRACTION

Functional Programming

Siva Jayaraman

TODAY

- Look at a familiar problem and its ‘natural’ solution
- Try to make it ‘better’
- On the way, build an abstraction that solves a pattern that the problem fits in
- Will use python for examples
- Ideas ~~inspired by~~ stolen from Logan Campbell’s YouTube talk
- Banish boring bullet points for the rest of the slides…

FAMILIAR PROBLEM

A chain of functions that may each fail

THE MODEL

Make the functions return None

```
maverick = {
    "name": "Pete Mitchell",
    "commander": viper,
    "contact": {
        "address": {
            "street": "123 Miramar Way",
            "city": "San Diego",
            "zip": 92126
        },
        "phone": "88586667777",
    }
}
```

```
viper = {
    "name": "Mike Metcalf",
    "contact": {
        "address": {
            "street": "1 Miramar Way",
            "city": "San Diego",
            "zip": 92126
        },
        "phone": "1221222",
        "mobile": "8586661111"
    }
}
```

```
charlie = {
    "name": "Charlotte Blackwood",
}
```

FIND A PILOT'S COMMANDER'S BASE CITY

```python
def get_commander_city(person):
    return person.get("commander").get("contact").get("address").get("city")
```

TRY IT OUT

```
get_commander_city(maverick)

get_commander_city(charlie)

get_commander_city(viper)
```

AttributeError: 'NoneType' object has no attribute 'get'

EASY WAY OUT

```
def get_commander_city_excepted(person):
    try:
        return person["commander"]["contact"]["address"]["city"]
except (AttributeError, KeyError):
    return None
```

SMALL FUNCTIONS

```
def get_commander(person):
    return person.get("commander")


def get_contact(person):
    return person.get("contact")
```

```
def get_address(contact):
    return contact.get("address")


def get_city(address):
    return address.get("city")
```

CHECKING FOR NONE

```
def get_commander_city(person):
    if person is not None:
        commander = get_commander(person)
        if commander is not None:
            contact = get_contact(commander)
            if contact is not None:
                address = get_address(contact)
                if address is not None:
                    return get_city(address)
return None
```

ANOTHER FAMILIAR PROBLEM

A chain of functions that fail with a message

THE MODEL

functions that return a value or an error message

```
success = {
    "status": True,
    "value": "Some value that can be of any type"
}
```

```
failure = {
    "status": False,
    "error": "A string representing error"
}
```

```
driving = {
    "header": "some header that must be present",
    "payload": {
        "commands": {
            "pre": {
                "name": "start engine"
            },
            "normal": {
                "name": "drive"
            },
            "post": {
                "name": "stop engine"
            }
        }
    }
}
```

```
sleeping = {
    "header": "some header that must be present",
    "payload": {
        "commands": {
            "pre": {
                "name": "close eyes"
            },
            "post": {
                "name": "open eyes"
            },
        }
    }
}
```

START WITH SMALL FUNCTIONS

```
def get_payload(message):
    if "header" in message:
        # do some processing of header
        if "payload" in message:
            return {"status": True, "value": message["payload"]}
        else:
            return {"status": False, "error": "No payload in message"}
    else:
        return {"status": False, "error": "No header in message"}
```

START WITH SMALL FUNCTIONS

```
def get_commands(payload):
    if "commands" in payload:
        return {"status": True, "value": payload["commands"]}
    else:
        return {"status": False, "value": "No commands section in payload"}


def get_normal_command(commands):
    if "normal" in commands:
        return {"status": True, "value": commands["normal"]}
    else:
        return {"status": False, "value": "No normal command in commands"}


def get_command_name(command):
    if "name" in command:
        return {"status": True, "value": command["name"]}
    else:
        return {"status": False, "value": "No command name specified for
command"}
```

PUTTING THEM TOGETHER

```
def get_normal_command_name_from_message(message):
    payload = get_payload(message)
    if payload["status"]:
        commands = get_commands(payload["value"])
        if commands["status"]:
            normal_command = get_normal_command(commands["value"])
            if normal_command["status"]:
                return get_command_name(normal_command["value"])
            else:
                return normal_command
        else:
            return commands
    else:
        return payload
```

```
def get_commander_city(person):
    if person is not None:
        commander = get_commander(person)
        if commander is not None:
            contact = get_contact(commander)
            if contact is not None:
                address = get_address(contact)
                if address is not None:
                    return get_city(address)
    return None
```

.....

```
def get_normal_command_name_from_message(message):
    payload = get_payload(message)
    if payload["status"]:
        commands = get_commands(payload["value"])
        if commands["status"]:
            normal_command = get_normal_command(commands["value"])
            if normal_command["status"]:
                return get_command_name(normal_command["value"])
            else:
                return normal_command
        else:
            return commands
    else:
        return payload
```

```
def get_commander_city(person):
    if person is not None:
        commander = get_commander(person)
        if commander is not None:
            contact = get_contact(commander)
            if contact is not None:
                address = get_address(contact)
                if address is not None:
                    return get_city(address)
    return None
```

.....

```
def call_if_not_none(fn, value):
    if value is not None:
        return fn(value)
    return None
```

```python
def get_commander_city(person):
    if person is not None:
        commander = get_commander(person)
        if commander is not None:
            contact = get_contact(commander)
            if contact is not None:
                address = get_address(contact)
                if address is not None:
                    return get_city(address)
    return None
```

.....

```python
def get_commander_city(person):
    return call_if_not_none(
        get_commander,
        person,
    )
```

```python
def get_commander_city(person):
    if person is not None:
        commander = get_commander(person)
        if commander is not None:
            contact = get_contact(commander)
            if contact is not None:
                address = get_address(contact)
                if address is not None:
                    return get_city(address)
    return None
```

.....

```python
def get_commander_city(person):
    return call_if_not_none(
        lambda x:
            get_commander(x),
        person,
    )
```

```python
def get_commander_city(person):
    if person is not None:
        commander = get_commander(person)
        if commander is not None:
            contact = get_contact(commander)
            if contact is not None:
                address = get_address(contact)
                if address is not None:
                    return get_city(address)
    return None
```

.....

```python
def get_commander_city(person):
    return call_if_not_none(
        lambda x: call_if_not_none(
            lambda y:




                get_contact(y),

            get_commander(x),
        ),
        person,
    )
```

```python
def get_commander_city(person):
    if person is not None:
        commander = get_commander(person)
        if commander is not None:
            contact = get_contact(commander)
            if contact is not None:
                address = get_address(contact)
                if address is not None:
                    return get_city(address)
    return None
```

.....

```python
def get_commander_city(person):
    return call_if_not_none(
        lambda x: call_if_not_none(
            lambda y: call_if_not_none(
                lambda z:

                    get_address(z),

                get_contact(y),
            ),
            get_commander(x),
        ),
        person,
    )
```

```
def get_commander_city(person):
    if person is not None:
        commander = get_commander(person)
        if commander is not None:
            contact = get_contact(commander)
            if contact is not None:
                address = get_address(contact)
                if address is not None:
                    return get_city(address)
    return None
```

.....

```
def get_commander_city(person):
    return call_if_not_none(
        lambda x: call_if_not_none(
            lambda y: call_if_not_none(
                lambda z: call_if_not_none(
                    lambda a: get_city(a),
                    get_address(z),
                ),
                get_contact(y),
            ),
            get_commander(x),
        ),
        person,
    )
```

```python
def get_commander_city(person):
    if person is not None:
        commander = get_commander(person)
        if commander is not None:
            contact = get_contact(commander)
            if contact is not None:
                address = get_address(contact)
                if address is not None:
                    return get_city(address)
    return None
```

```python
def get_commander_city(person):
    return call_if_not_none(
        lambda x: call_if_not_none(
            lambda y: call_if_not_none(
                lambda z: call_if_not_none(
                    lambda a: get_city(a),
                    get_address(z),
                ),
                get_contact(y),
            ),
            get_commander(x),
        ),
        person,
    )
```

```python
def get_normal_command_name_from_message(message):
    payload = get_payload(message)
    if payload["status"]:
        commands = get_commands(payload["value"])
        if commands["status"]:
            normal_command = get_normal_command(commands["value"])
            if normal_command["status"]:
                return get_command_name(normal_command["value"])
            else:
                return normal_command
        else:
            return commands
    else:
        return payload
```

?

```
def get_normal_command_name_from_message(message):
    payload = get_payload(message)
    if payload["status"]:
        commands = get_commands(payload["value"])
        if commands["status"]:
            normal_command = get_normal_command(commands["value"])
            if normal_command["status"]:
                return get_command_name(normal_command["value"])
            else:
                return normal_command
        else:
            return commands
    else:
        return payload
```

```
def call_upon_valid_value(fn, item):
    if item["status"]:
        return fn(item["value"])
    else:
        return item
```

```
def get_normal_command_name_from_message(message):
    payload = get_payload(message)
    if payload["status"]:
        commands = get_commands(payload["value"])
        if commands["status"]:
            normal_command = get_normal_command(commands["value"])
            if normal_command["status"]:
                return get_command_name(normal_command["value"])
            else:
                return normal_command
        else:
            return commands
    else:
        return payload
```

.....

```
def get_normal_command_name_from_message(message):
    return call_upon_valid_value(
```

```
            get_payload,

        message
    )
```

```
def get_normal_command_name_from_message(message):
    payload = get_payload(message)
    if payload["status"]:
        commands = get_commands(payload["value"])
        if commands["status"]:
            normal_command = get_normal_command(commands["value"])
            if normal_command["status"]:
                return get_command_name(normal_command["value"])
            else:
                return normal_command
        else:
            return commands
    else:
        return payload
```

.....

```
def get_normal_command_name_from_message(message):
    return call_upon_valid_value(
```

```
            get_payload,
```

```
        message
    )
```

```
def get_normal_command_name_from_message(message):
    payload = get_payload(message)
    if payload["status"]:
        commands = get_commands(payload["value"])
        if commands["status"]:
            normal_command = get_normal_command(commands["value"])
            if normal_command["status"]:
                return get_command_name(normal_command["value"])
            else:
                return normal_command
        else:
            return commands
    else:
        return payload
```

.....

```
def get_normal_command_name_from_message(message):
    return call_upon_valid_value(
```

```
            get_payload,
        {"status": True, "value": message}
    )
```

```
def get_normal_command_name_from_message(message):
    payload = get_payload(message)
    if payload["status"]:
        commands = get_commands(payload["value"])
        if commands["status"]:
            normal_command = get_normal_command(commands["value"])
            if normal_command["status"]:
                return get_command_name(normal_command["value"])
            else:
                return normal_command
        else:
            return commands
    else:
        return payload
```

.....

```
def get_normal_command_name_from_message(message):
    return call_upon_valid_value(
        lambda x:
```



```
            get_payload(x),

        {"status": True, "value": message}
    )
```

```python
def get_normal_command_name_from_message(message):
    payload = get_payload(message)
    if payload["status"]:
        commands = get_commands(payload["value"])
        if commands["status"]:
            normal_command = get_normal_command(commands["value"])
            if normal_command["status"]:
                return get_command_name(normal_command["value"])
            else:
                return normal_command
        else:
            return commands
    else:
        return payload
```

```python
def get_normal_command_name_from_message(message):
    return call_upon_valid_value(
        lambda x: call_upon_valid_value(
            lambda y:




                get_commands(y),

            get_payload(x)
        ),
        {"status": True, "value": message}
    )
```

```python
def get_normal_command_name_from_message(message):
    payload = get_payload(message)
    if payload["status"]:
        commands = get_commands(payload["value"])
        if commands["status"]:
            normal_command = get_normal_command(commands["value"])
            if normal_command["status"]:
                return get_command_name(normal_command["value"])
            else:
                return normal_command
        else:
            return commands
    else:
        return payload
```

```python
def get_normal_command_name_from_message(message):
    return call_upon_valid_value(
        lambda x: call_upon_valid_value(
            lambda y: call_upon_valid_value(
                lambda z:

                    get_normal_command(z),

                get_commands(y),
            ),
            get_payload(x)
        ),
        {"status": True, "value": message}
    )
```

```python
def get_normal_command_name_from_message(message):
    payload = get_payload(message)
    if payload["status"]:
        commands = get_commands(payload["value"])
        if commands["status"]:
            normal_command = get_normal_command(commands["value"])
            if normal_command["status"]:
                return get_command_name(normal_command["value"])
            else:
                return normal_command
        else:
            return commands
    else:
        return payload
```

.....

```python
def get_normal_command_name_from_message(message):
    return call_upon_valid_value(
        lambda x: call_upon_valid_value(
            lambda y: call_upon_valid_value(
                lambda z: call_upon_valid_value(
                    lambda a: get_command_name(a),
                    get_normal_command(z)
                ),
                get_commands(y),
            ),
            get_payload(x)
        ),
        {"status": True, "value": message}
    )
```

```python
def get_normal_command_name_from_message(message):
    payload = get_payload(message)
    if payload["status"]:
        commands = get_commands(payload["value"])
        if commands["status"]:
            normal_command = get_normal_command(commands["value"])
            if normal_command["status"]:
                return get_command_name(normal_command["value"])
            else:
                return normal_command
        else:
            return commands
    else:
        return payload
```

```python
def get_normal_command_name_from_message(message):
    return call_upon_valid_value(
        lambda x: call_upon_valid_value(
            lambda y: call_upon_valid_value(
                lambda z: call_upon_valid_value(
                    lambda a: get_command_name(a),
                    get_normal_command(z)
                ),
                get_commands(y),
            ),
            get_payload(x)
        ),
        {"status": True, "value": message}
    )
```

“

I see a pattern!

- A Keen Programmer!

```
def get_commander_city(person):
    return call_if_not_none(
        lambda x: call_if_not_none(
            lambda y: call_if_not_none(
                lambda z: call_if_not_none(
                    lambda a: get_city(a),
                    get_address(z),
                ),
                get_contact(y),
            ),
            get_commander(x),
        ),
        person,
    )
```

```
def get_normal_command_name_from_message(message):
    return call_upon_valid_value(
        lambda x: call_upon_valid_value(
            lambda y: call_upon_valid_value(
                lambda z: call_upon_valid_value(
                    lambda a: get_command_name(a),
                    get_normal_command(z)
                ),
                get_commands(y),
            ),
            get_payload(x)
        ),
        {"status": True, "value": message}
    )
```

```
def get_commander_city(person):
    return call_if_not_none(
        lambda x: call_if_not_none(
            lambda y: call_if_not_none(
                lambda z: call_if_not_none(
                    lambda a: get_city(a),
                    get_address(z),
                ),
                get_contact(y),
            ),
            get_commander(x),
        ),
        person,
    )
```

```
def get_normal_command_name_from_message(message):
    return call_upon_valid_value(
        lambda x: call_upon_valid_value(
            lambda y: call_upon_valid_value(
                lambda z: call_upon_valid_value(
                    lambda a: get_command_name(a),
                    get_normal_command(z)
                ),
                get_commands(y),
            ),
            get_payload(x)
        ),
        {"status": True, "value": message}
    )
```

```
def get_commander_city(person):
    return call_if_not_none(
        lambda x: call_if_not_none(
            lambda y: call_if_not_none(
                lambda z: call_if_not_none(
                    lambda a: get_city(a),
                    get_address(z),
                ),
                get_contact(y),
            ),
            get_commander(x),
        ),
        person,
    )
```

```
def get_normal_command_name_from_message(message):
    return call_upon_valid_value(
        lambda x: call_upon_valid_value(
            lambda y: call_upon_valid_value(
                lambda z: call_upon_valid_value(
                    lambda a: get_command_name(a),
                    get_normal_command(z)
                ),
                get_commands(y),
            ),
            get_payload(x)
        ),
        {"status": True, "value": message}
    )
```

```
def identity(a):
    return a
```

```
def wrapped_in_success(x):
    return {"status": True, "value": x}
```

```
def get_commander_city(person):
    return call_if_not_none(
        lambda x: call_if_not_none(
            lambda y: call_if_not_none(
                lambda z: call_if_not_none(
                    lambda a: get_city(a),
                    get_address(z),
                ),
                get_contact(y),
            ),
            get_commander(x),
        ),
        identity(person),
    )
```

```
def get_normal_command_name_from_message(message):
    return call_upon_valid_value(
        lambda x: call_upon_valid_value(
            lambda y: call_upon_valid_value(
                lambda z: call_upon_valid_value(
                    lambda a: get_command_name(a),
                    get_normal_command(z)
                ),
                get_commands(y),
            ),
            get_payload(x)
        ),
        wrapped_in_success(message)
    )
```

```
def get_commander_city(person):
    return call_if_not_none(
        lambda x: call_if_not_none(
            lambda y: call_if_not_none(
                lambda z: call_if_not_none(
                    lambda a: get_city(a),
                    get_address(z),
                ),
                get_contact(y),
            ),
            get_commander(x),
        ),
        identity(person),
    )
```

```
def get_normal_command_name_from_message(message):
    return call_upon_valid_value(
        lambda x: call_upon_valid_value(
            lambda y: call_upon_valid_value(
                lambda z: call_upon_valid_value(
                    lambda a: get_command_name(a),
                    get_normal_command(z)
                ),
                get_commands(y),
            ),
            get_payload(x)
        ),
        wrapped_in_success(message)
    )
```

THE CORE

```
none_stepper_def = {
    "stepper": call_if_not_none,
    "wrapper": identity
}
```

.....

```
status_stepper_def = {
    "stepper": call_upon_valid_value,
    "wrapper": wrapped_in_success
}
```

WE'VE JUST INVENTED A MONAD!

TRAVERSAL

```
person.get("commander").get("contact").get("address").get("city")
```

```
def get_commander_city(person):
    return call_if_not_none(
        lambda x: call_if_not_none(
            lambda y: call_if_not_none(
                lambda z: call_if_not_none(
                    lambda a: get_city(a),
                    get_address(z),
                ),
                get_contact(y),
            ),
            get_commander(x),
        ),
        person,
    )
```

TRAVERSAL

```
person.get("commander").get("contact").get("address").get("city")
```

```
link(
    person,
    [get_commander, get_contact, get_address, get_city],
    none_stepper_def
)
```

```
def link(data, steps, stepper_def):
    def reduction_fn(acc_data, fn):
        return stepper_def["stepper"](
                fn,
                acc_data
        )

    return reduce(
        reduction_fn,                           # reduction function
        steps,                                  # sequence of steps
        stepper_def["wrapper"](data)            # initial value
    )
```

TRAVERSAL

```
link(
    person,
    [get_commander, get_contact, get_address, get_city],
    none_stepper_def
)
```

```
link(
    message,
    [get_payload, get_commands, get_normal_command, get_command_name],
    status_stepper_def
)
```

INTRODUCING MONADS

WHAT'S A MONAD NOT!

Not A Burrito! (unless your burrito contains a stepper def)!

Not unique to Haskell

Not just about IO

Not impure or about only state

Not a DSL

Not always about ordering or sequencing

SO, WHAT IS A MONAD?

A contract / interface / protocol containing two functions!

And a set of rules that these functions must follow

```
status_stepper_def = {
    "stepper": call_upon_valid_value,
    "wrapper": wrapped_in_success
}
```

SO, WHAT IS A MONAD?

A contract / interface / protocol containing two functions!

And a set of rules that these functions must follow

```
status_stepper_def = {
    "stepper": call_upon_valid_value,
    "wrapper": wrapped_in_success
}
```

SO, WHAT IS A MONAD?

A contract / interface / protocol containing two functions!

And a set of rules that these functions must follow

```
status_stepper_def = {
    "bind": call_upon_valid_value,
    "return": wrapped_in_success
}
```

SO, WHAT IS A MONAD?

A contract / interface / protocol containing two functions!

And a set of rules that these functions must follow

```
status_stepper_def = {
    "bind": call_upon_valid_value,
    "return": wrapped_in_success
}
```

In Haskell, the way the function link is implemented is in terms of syntactic sugar (do notation)

LET'S LOOK AT DATA TYPES

Return type

wrapped_type

function

unwrapped_type -> wrapped_type

e.g. get_commander, get_contact, …

unwrapped_type

```
def call_upon_valid_value(fn, item):
    if item["status"]:
        return fn(item["value"])
    else:
        return item
```

“

I believe in intuitions and inspirations...I sometimes FEEL that I am right. I do not KNOW that I am.

-Albert Einstein

ASK WHAT A MONAD CAN DO FOR YOU!

Sequencing*

Model computations that have a component of non-determinism

“piping” things through computations. Think ENV/Config

“recording” things as they happen through computations. Think logs, etc

“Thread” “State” through computations

and many many more…

SOME USES OF MONADS IN THE WILD

IO (Haskell's IO is on top of a monad and a topic by itself)

Context-sensitive parsers

Passing configuration through computations

Stacking monads on top of each other!

many many more…

SOME INTERESTING MONADS

Maybe - A variant of the None chaining we saw before

Either - The status we saw before

List* - Yes, list is a monad

Reader

Writer

State

Function composition* - Yes, this is a monad too!

THE JOURNEY BEGINS