

Intro to iOS completion handlers
Intro to iOS anonymous functions
Intro to iOS function literals

Intro to iOS Closures

Intro to iOS lambdas
Intro to iOS blocks

by Fernando Olivares

Agenda

1-hour session

- Why we're here: Underdog devs
- Author Introduction
- Also why we're here: What is a closure?
- Live coding
- Thinking *in* closures
- Conclusion & Where to go from here
- Q&A






Underdog Devs

Supporting the formerly incarcerated & disadvantaged

- Led by Rick Wolter (@rwoltx)
- Trying to help the formerly incarcerated and other disadvantaged developers in their transition into the software industry.
- Currently relies 100% on volunteers
- It doesn't matter if you're junior or senior, we're looking for help:
 - Mentors can provide sage advice to newcomers
 - Juniors can help improve their skills by teaching
- More plans to help people are coming down the line

Author Introduction

Fernando

- ~10 years of experience
- Worked at small startups ( 1SecondEveryday) to publicly traded companies ( J2 Global Inc.)
- Instructor at  Big Nerd Ranch,  bloc.io,  Lambda School
- Won a few awards: The Storyteller Within (Apple), ERA Accelerator Top 10 (ERA NY)
- Product and Project experience
- iOS-only, bought the first iPhone without knowing if he'd be able to use it
- @fromJrToSr

What is a closure?

In theory, which is complex and boring but important.

- Official definition: "Closures are self-contained blocks of functionality that can be passed around and used in your code."¹
- They are First-class citizens², so they can be passed as arguments, returned from a function, modified and assigned to constants or variables,
- Even though they're functions, they have slightly different rules than regular functions (no labels, custom syntax, capturing values, implicit returns)
- They are the building blocks of higher order functions (e.g. map, reduce) and are very common in asynchronous functions.

¹ - <https://docs.swift.org/swift-book/LanguageGuide/Closures.html>

² - https://en.wikipedia.org/wiki/First-class_citizen

What is a closure?

In practice, which is fun and exciting but nothing without the theory

- Fernando definition: "It's a function without labels and some frustrating exceptions."
- You can declare them, bop them, twist them, pull them, but their most common use cases are being a parameter and being called for a single-purpose, or being called sometime in the future.
- Most closures have special memory rules which is uncommon nowadays in Swift.
- Closures are not exceptional. They're as common as Arrays, Dictionaries and Strings. As soon as you demystify them, they become easy to understand.

Live Demo

High-order functions: Create our own

- `func map` that transforms an array with objects/structs of type A, into an array with objects/structs of type B
 - [lowercase String] -> [capitalized String]
 - [lowercase String] -> [uppercase String]
 - DRY with a function
 - DRY with a closure
 - map

Live Demo

Create a function that transforms elements in an array



```
func transform(originalStrings: [String]) -> [String]
```



```
func transform(originalStrings: [🐷]) -> [🍷]
```

[String]



-> [String]

```
func transform(originalStrings: [String]) -> [String]
```

[🐷]



-> [🍖]

func transform(originalPorks: [🐷]) -> [🍖]

[🐷]



-> [🍷]

func transform

Creates a new
array [🍷]

save into new
array [🍷]

Iterate through
[🐷]

transform each
element

🐷->🍷

[🐷]



-> [🍷]

func transform

Creates a new
array [🍷]

save into new
array [🍷]

Iterate through
[🐷]

transform each
element

🐷->🍷

[🐷]



-> [🌮]

func transform

Creates a new
array [🍌]

save into new
array [🍌]

Iterate through
[🐷]

transform each
element

🐷->🍌

[🐷]



-> [🌭]

func transform

Creates a new
array [🍳]

save into new
array [🍳]

Iterate through
[🐷]

transform each
element

🐷->🍳

[🐷]



-> [🥪]

func transform

Creates a new
array [🥓]

save into new
array [🥓]

Iterate through
[🐷]

transform each
element

🐷 -> 🥓

[🐷]



→ [🥪]

func transform

Creates a new
array [🍌]

save into new
array [🍌]

Iterate through
[🐷]



[🐷]



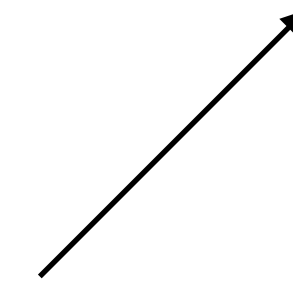
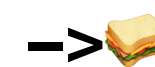
-> [🥪]

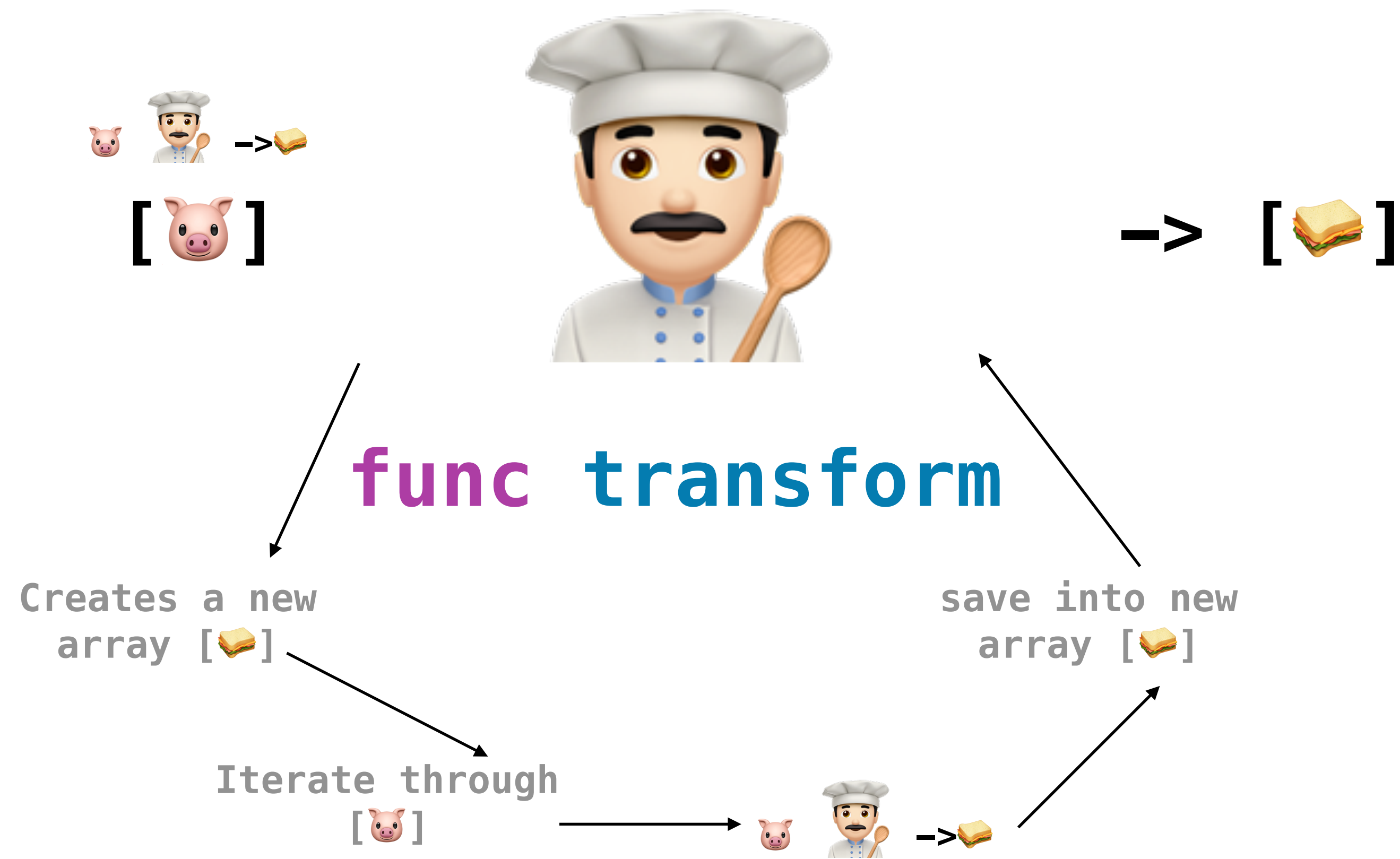
func transform

Creates a new
array [🥪]

save into new
array [🥪]

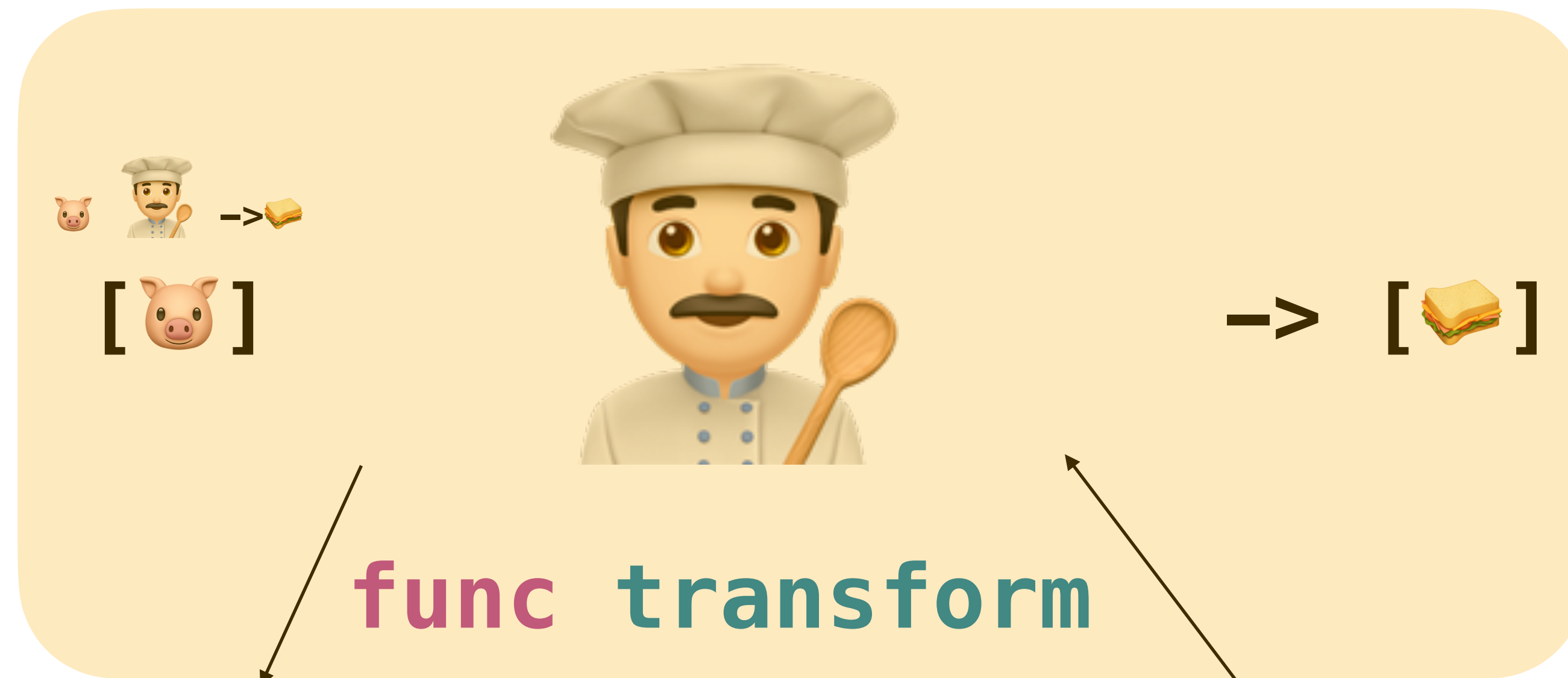
Iterate through
[🐷]





Live Demo

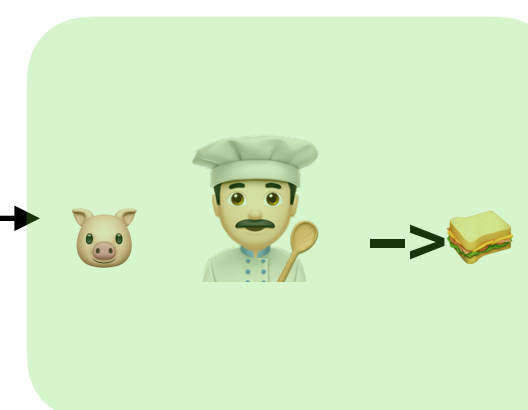
Create a function that transforms elements in an array



Creates a new
array [🥪]

save into new
array [🥪]

Iterate through
[🐷]



Thinking in closures

They're like functions except when they're not

- In high-order functions,
 - declaration happens at the implementation of the containing function
 - implementation happens at the call site of the containing function
- Parameter labels are missing at the implementation
- Parameter types and return type are inferred at implementation
- Trailing closure syntax
- Shorthand arguments
- Omitting return keyword
- Keypaths

Thinking in closures

They're like functions except when they're not

- In high-order functions,
 - declaration happens at the implementation of the containing function

```
func transform(givenStrings: [String], closure: (String) -> String) -> [String]
```

- implementation happens at the call site of the containing function

```
let capitalizedNames = transform(strings: names, closure: { (singleString: String) -> String in })
```


Thinking in closures

They're like functions except when they're not

- Parameter labels are missing at the implementation

```
func transform(givenStrings: [String], closure: (String) -> String) -> [String]
```

- Parameter types and return type are inferred at implementation

```
let capitalizedNames = transform(strings: names, closure: { singleString in })
```

- Trailing closure syntax

```
let capitalizedNames = transform(strings: names) { singleString in }
```

Thinking in closures

They're like functions except when they're not

- Shorthand arguments

```
let capitalizedNames = transform(strings: names, closure: { return $0.capitalized })
```

- Omitting return keyword

```
let capitalizedNames = transform(strings: names, closure: { $0.capitalized })
```

- Keypaths

```
let capitalizedNames = names.map(\.capitalized)
```

Live Demo

What about networking?



```
func dataTask(with request: URLRequest,  
              completionHandler: @escaping (Data?,  
URLResponse?, Error?) -> Void) -> URLSessionDataTask
```

 @escaping (Data?, URLResponse?, Error?) -> Void)
completionHandler:


 request: URLRequest  -> URLSessionDataTask 

 @escaping (Data?, URLResponse?, Error?) -> Void)
completionHandler:

 request: URLRequest  -> URLSessionDataTask 

Build a new 

Add the request 
to the envelope

Add the
completionHandler
 to the envelope


Return the
envelope 

 @escaping (Data?, URLResponse?, Error?) -> Void)
completionHandler:

 request: URLRequest  -> URLSessionDataTask 

Build a new 

Add the request 
to the envelope

Add the
completionHandler
 to the envelope

Return the
envelope 



```
func resume()
```



 @escaping (Data?, URLResponse?, Error?) -> Void)
completionHandler:

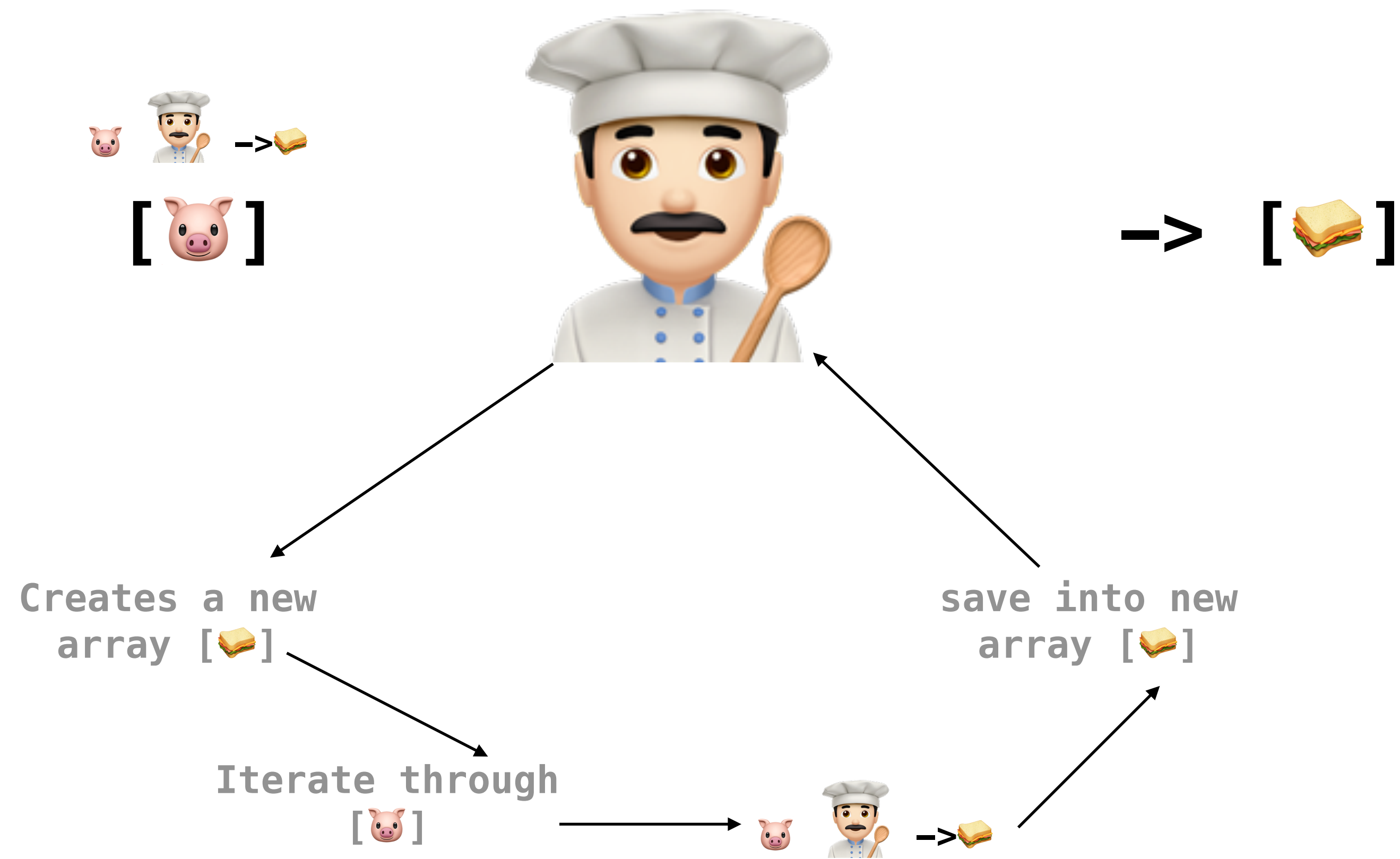
 request: URLRequest  -> URLSessionDataTask 

Build a new 

Add the request 
to the envelope

Return the
envelope 

Add the
completionHandler
 to the envelope




 @escaping (Data?, URLResponse?, Error?) -> Void)
completionHandler:

 request: URLRequest  -> URLSessionDataTask 

Build a new 

Add the request 
to the envelope

Add the
completionHandler
 to the envelope

Return the
envelope 



```
func resume()
```



```
func dataTask(with request: URLRequest,  
              completionHandler: @escaping (Data?,  
URLResponse?, Error?) -> Void) -> URLSessionDataTask
```

Other types of closures

They're like other closures except when they're not

- Other types of closures exist:
 - Global functions
 - Nested functions
 - var initialization

Other types of closures

They're like other closures except when they're not

- Global functions

```
func transform(originalStrings: [String]) ->  
[String] { ... }
```

Other types of closures

They're like other closures except when they're not

- Nested functions

```
func transform(originalStrings: [String]) ->
[String] {

    func internalFunction() {}

}
```


Other types of closures

They're like other closures except when they're not

- var initialization

```
struct Person {  
    lazy var familyTree: [Person] = {  
        return fetchFromDisk()  
    }()  
}
```

Closures and more

Where to go from here?

- Will be uploaded to [https://github.com/olivaresf/intro to closures](https://github.com/olivaresf/intro_to_closures)
- You can reach me @fromJrToSr
- Additional info:
 - <https://docs.swift.org/swift-book/LanguageGuide/Closures.html>
 - <https://medium.com/better-programming/everything-you-wanted-to-know-about-closures-in-swift-e7d3a6ff5a74>

Underdog Devs

Supporting the formerly incarcerated & disadvantaged

- Led by Rick Wolter (@rwoltx)
- Trying to help the formerly incarcerated and other disadvantaged developers in their transition into the software industry.
- Currently relies 100% on volunteers
- It doesn't matter if you're junior or senior, we're looking for help:
 - Mentors can provide sage advice to newcomers
 - Juniors can help improve their skills by teaching
- More plans to help people are coming down the line

Fernando (From Junior to Senior)

Supporting the Fernando and his craft

- If you like the presentation, follow me on Twitter (@fromJrToSr)
- I would really appreciate if you join my mailing list, where I send weekly 15-minute Swift exercises.
 - <https://eepurl.com/g9P8eT>

Q&A