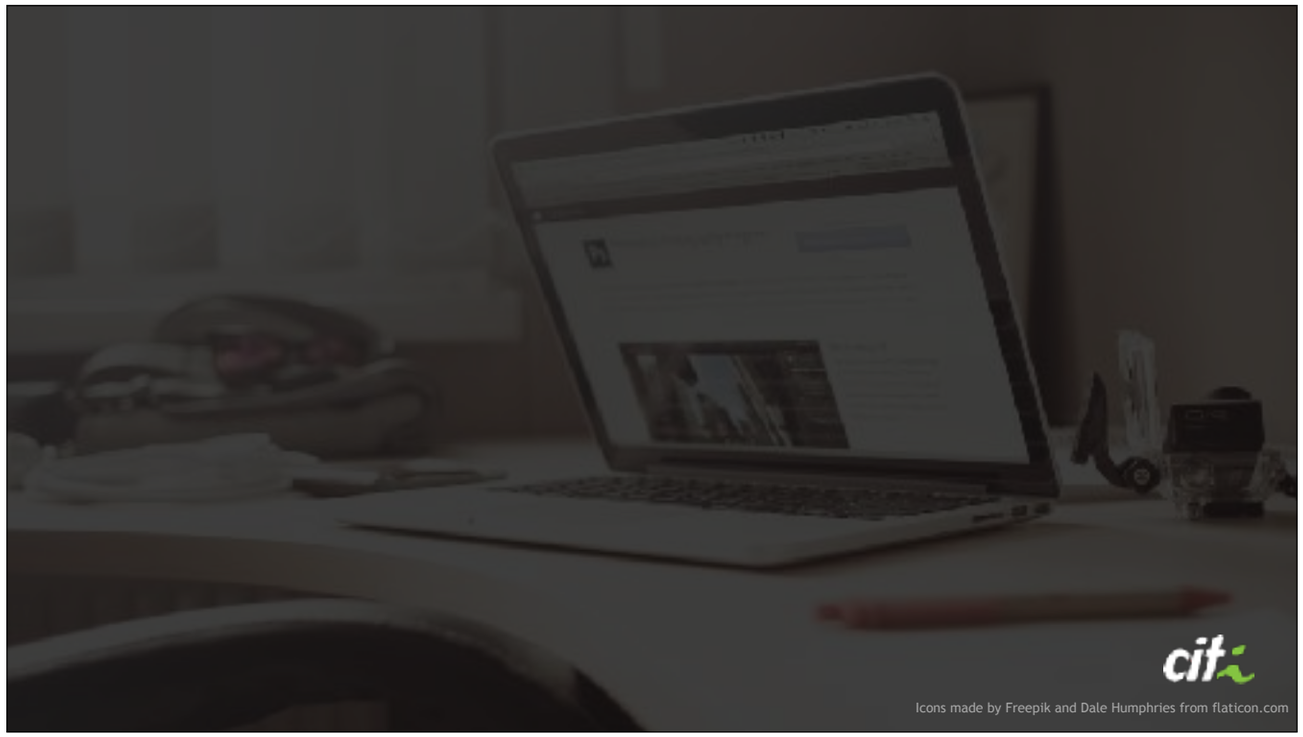
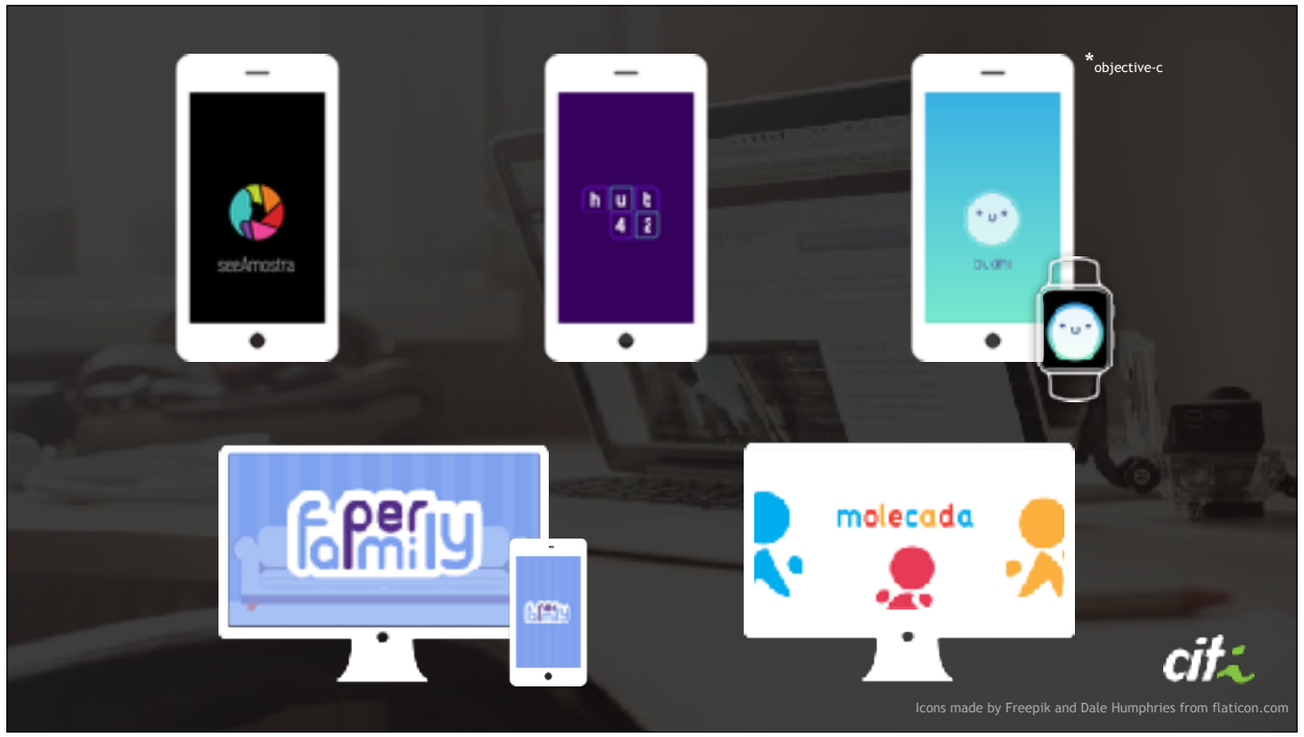

cit

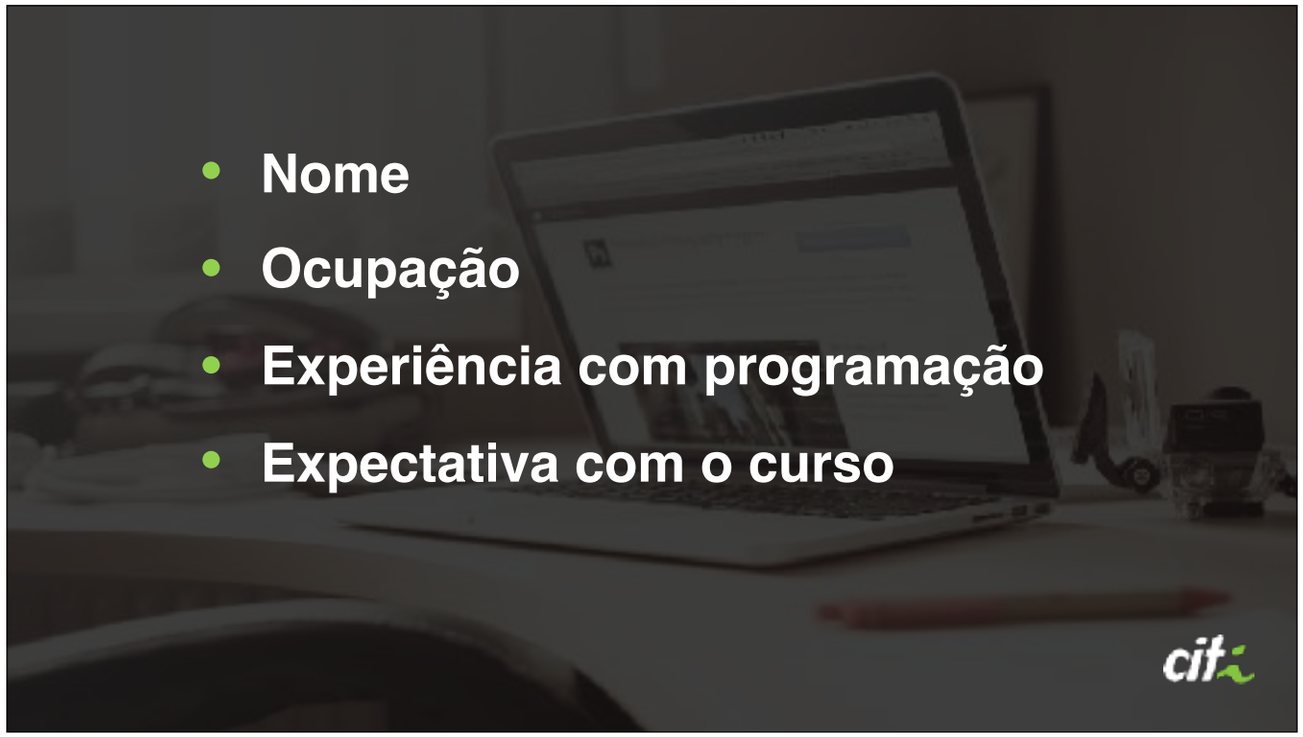

Hilton **Pintor**
Desenvolvedor (iOS/tvOS/watchOS)
hiltonpintor@gmail.com
citi

// Aula 01

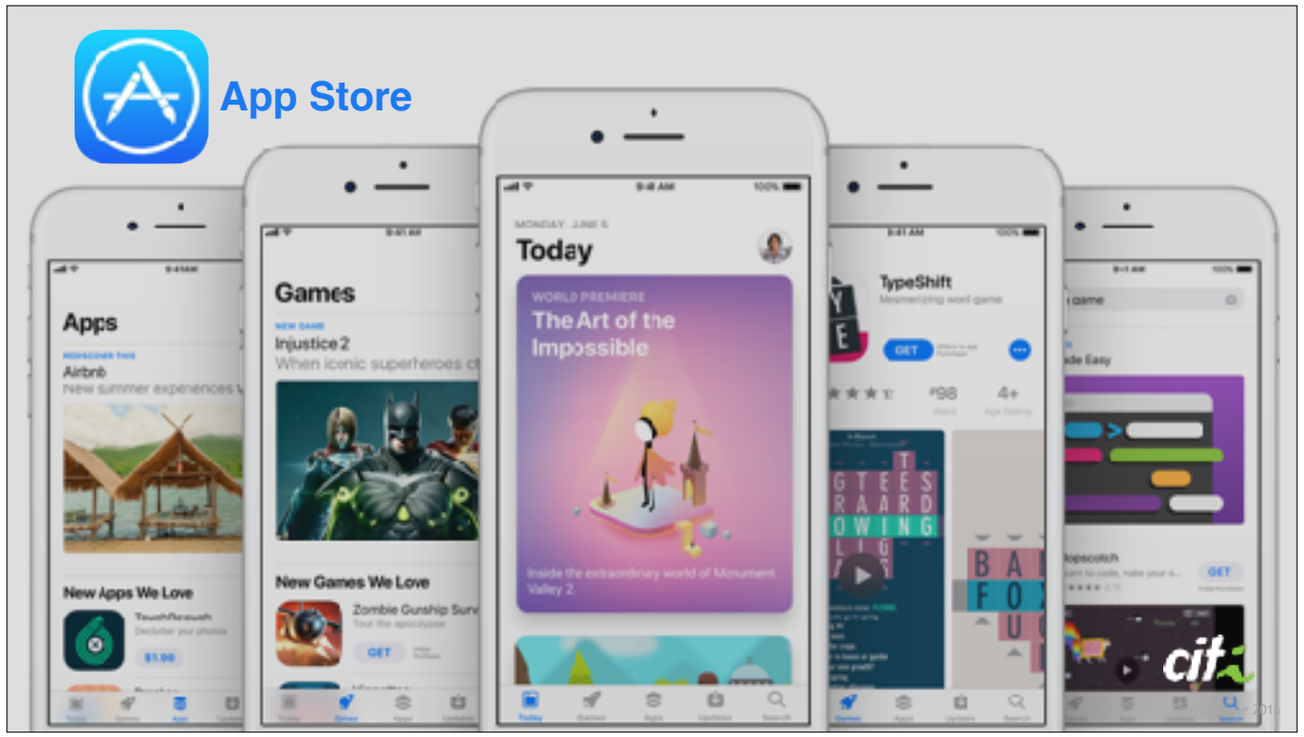

Icons made by Freepik and Dale Humphries from flaticon.com


seeAmostra
h u t 4 2
*objective-c
faperly family
molecada
citi
Icons made by Freepik and Dale Humphries from flaticon.com


iOS
com Swift
citi

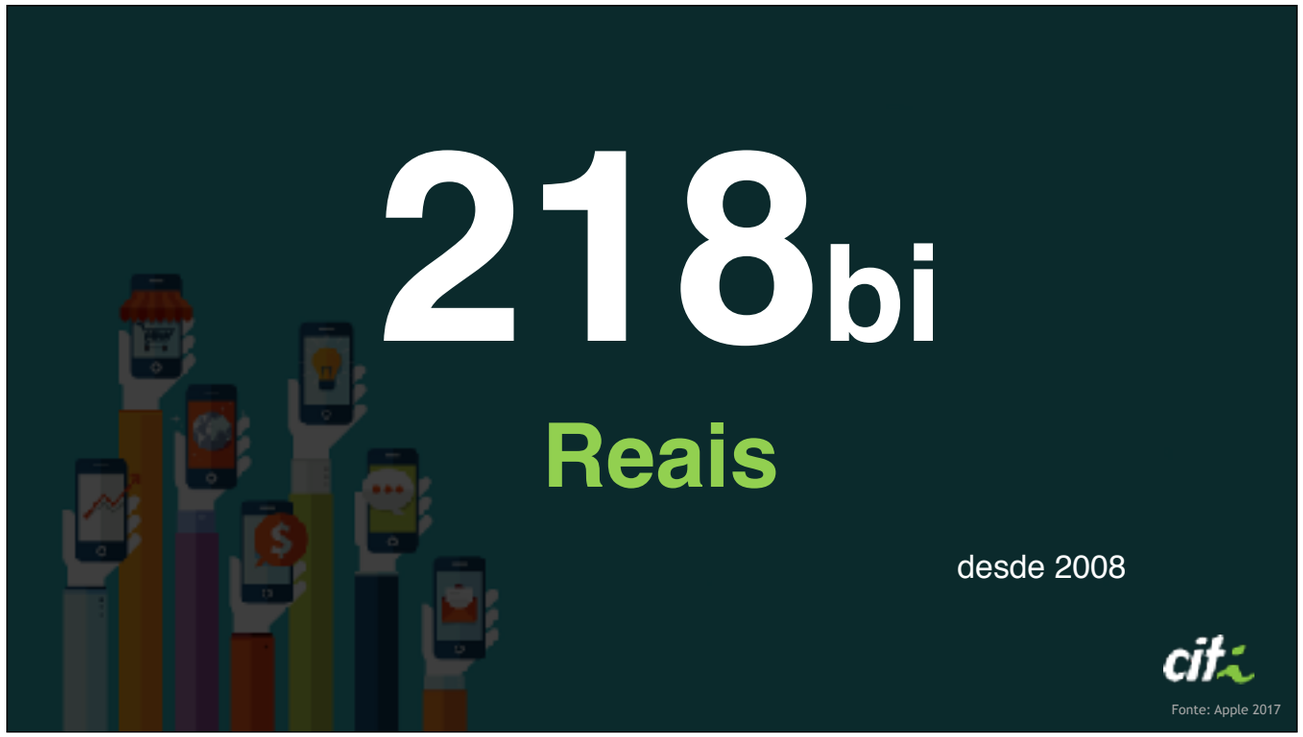

Nome
Ocupação
Experiência com programação
Expectativa com o curso


iOS
com Swift



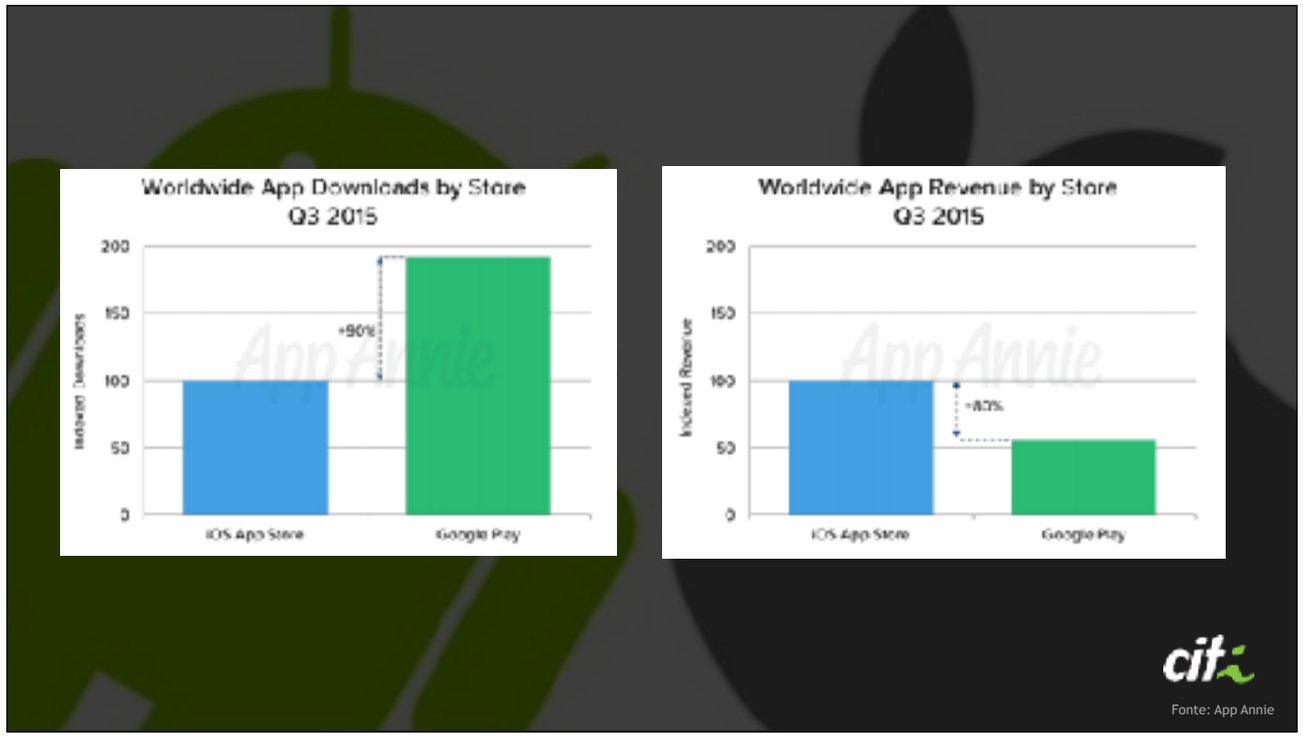
App store é a loja de apps da Apple

Apple announced on 2016 WWDC that there are over **2 million apps** available on the **app store**

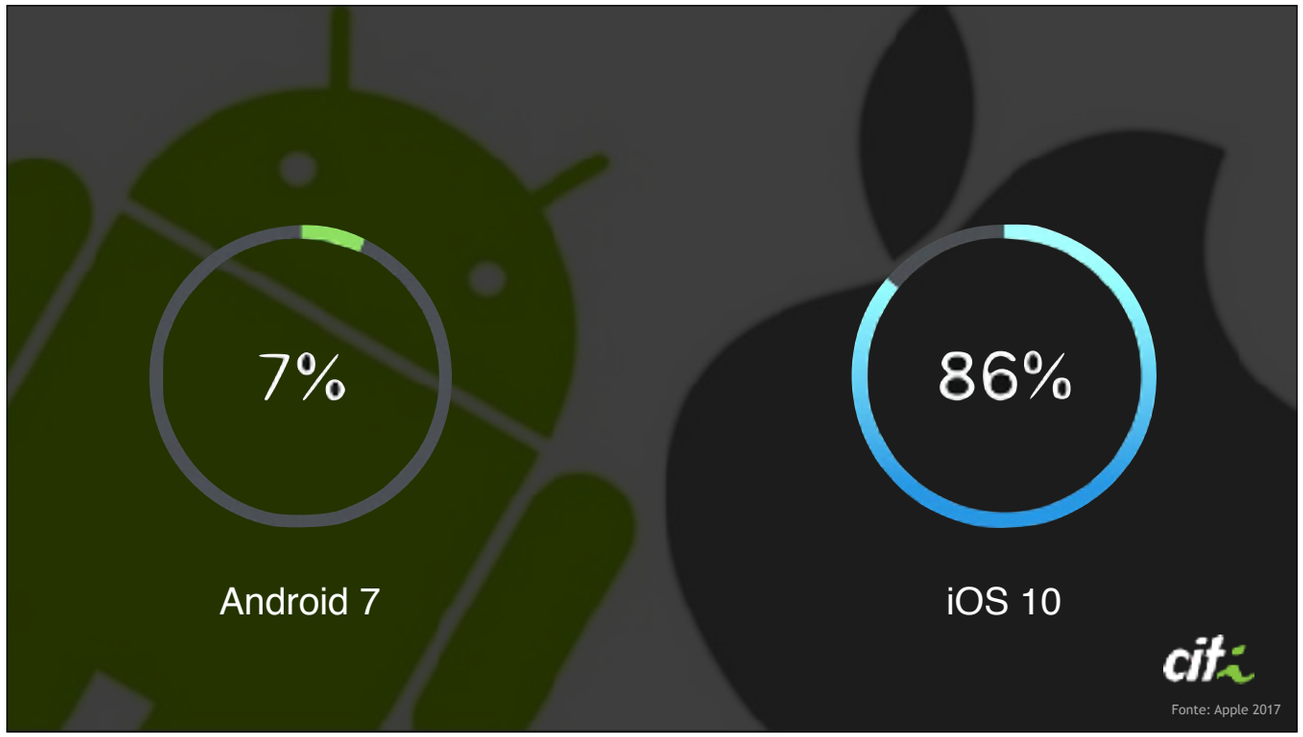


Apple today announced that its global **developer** community has earned over **$70 billion** since the App Store launched in **2008**.

There are many **more devices out there running Android**
the **App Store** still brings in significantly **more revenue** – to the tune of about **75%** according to a report from App Annie.

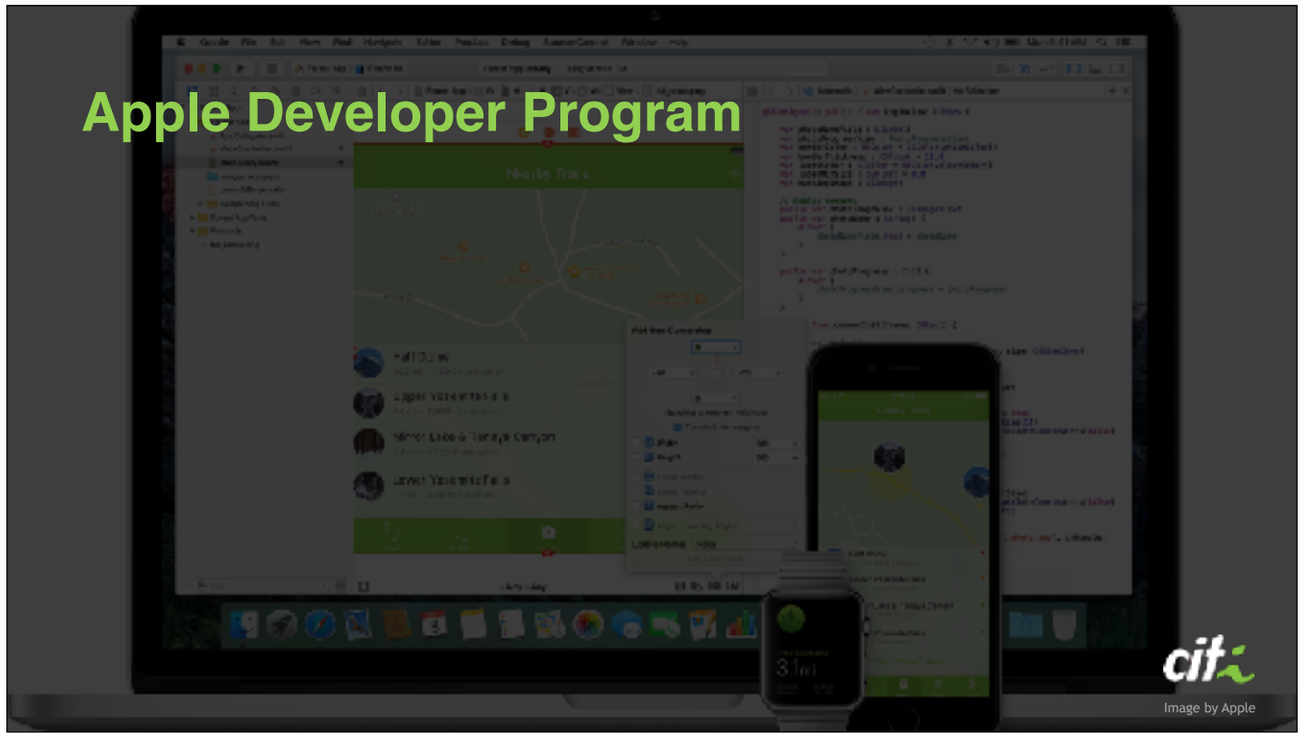


There are many **more devices out there running Android**
the **App Store** still brings in significantly **more revenue** – to the tune of about **75%** according to a report from App Annie.



Menor fragmentação no iOS.
Usuários fazem **updates** com mais **frequência**.

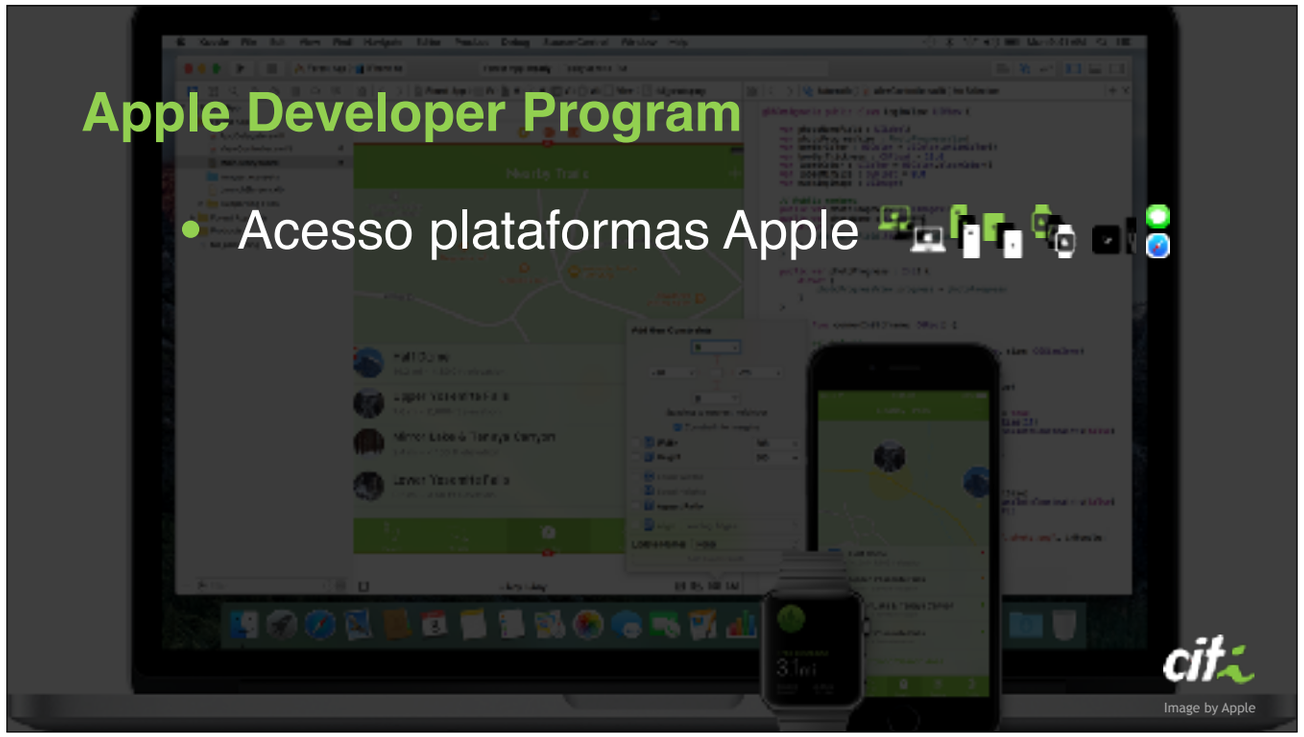
Apple Developer Program



Processo de submissão

Apple Developer Program

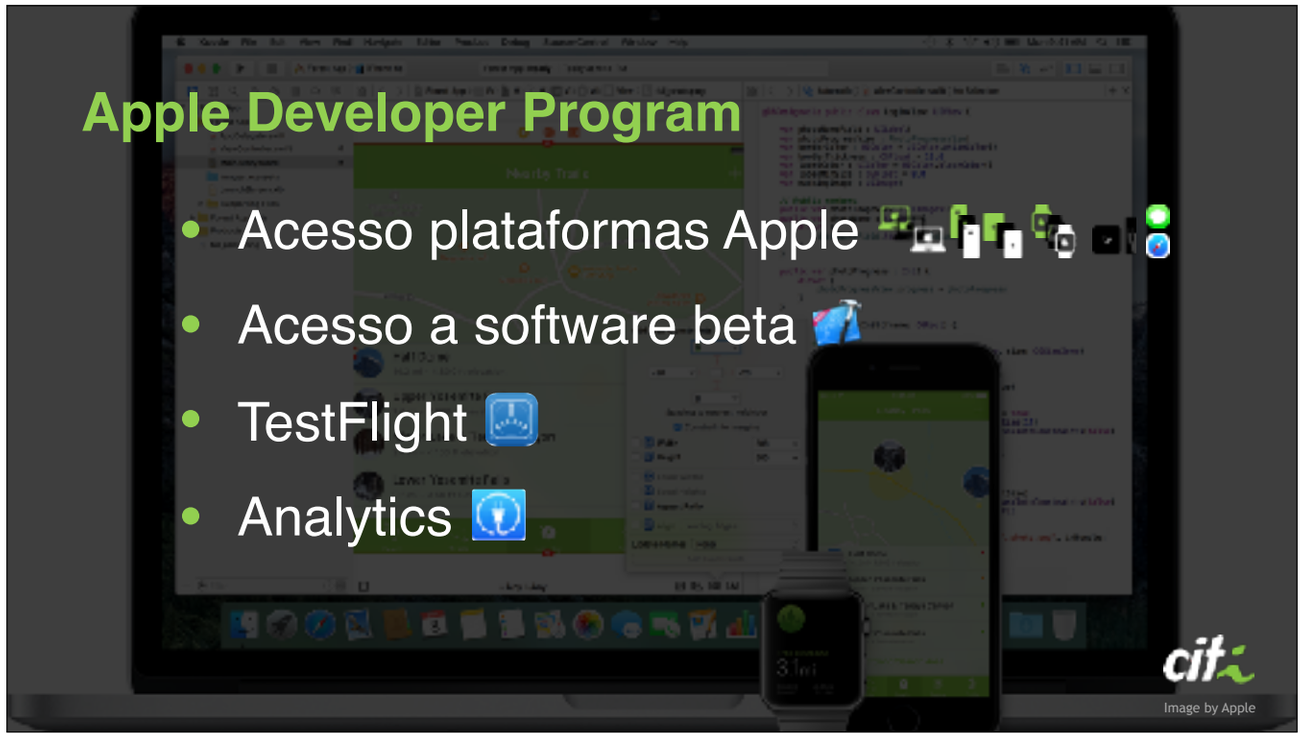
- Acesso plataformas Apple



Processo de submissão

Processo de submissão

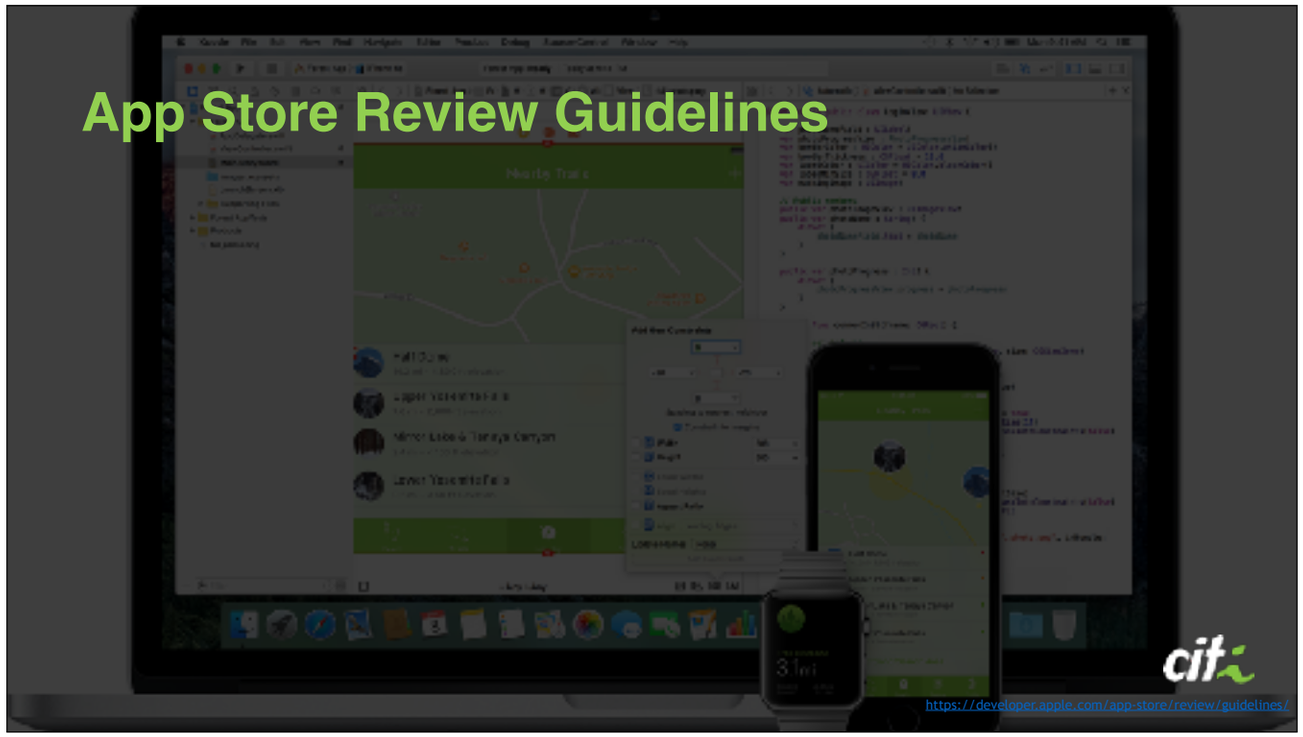
Processo de submissão



Processo de submissão

Processo de submissão

App Store Review Guidelines

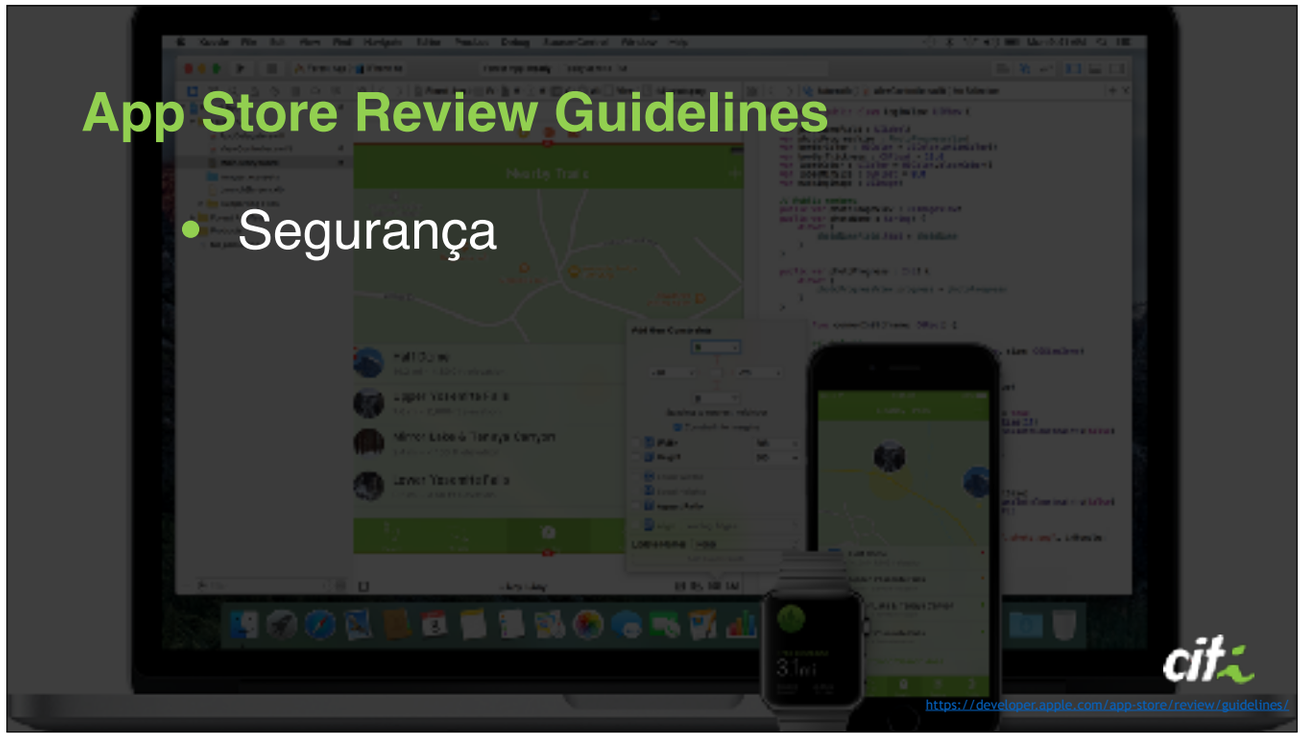


https://developer.apple.com/app-store/review/guidelines/

1. Safety
    - 1.1 Objectionable Content
    - 1.2 User Generated Content
    - 1.3 Kids Category
    - 1.4 Physical Harm
    - 1.5 Developer Information
2. Performance
    - 2.1 App Completeness
    - 2.2 Beta Testing
    - 2.3 Accurate Metadata
    - 2.4 Hardware Compatibility
    - 2.5 Software Requirements
3. Business
    - 3.1 Payments
    - 3.1.1 In-App Purchase
    - 3.1.2 Subscriptions
    - 3.1.3 Content-based “Reader” Apps
    - 3.1.4 Content Codes
    - 3.1.5 Physical Goods and Services Outside of the App
    - 3.1.6 Apple Pay
    - 3.2 Other Business Model Issues
    - 3.2.1 Acceptable
    - 3.2.2 Unacceptable
4. Design
    - 4.1 Copycats

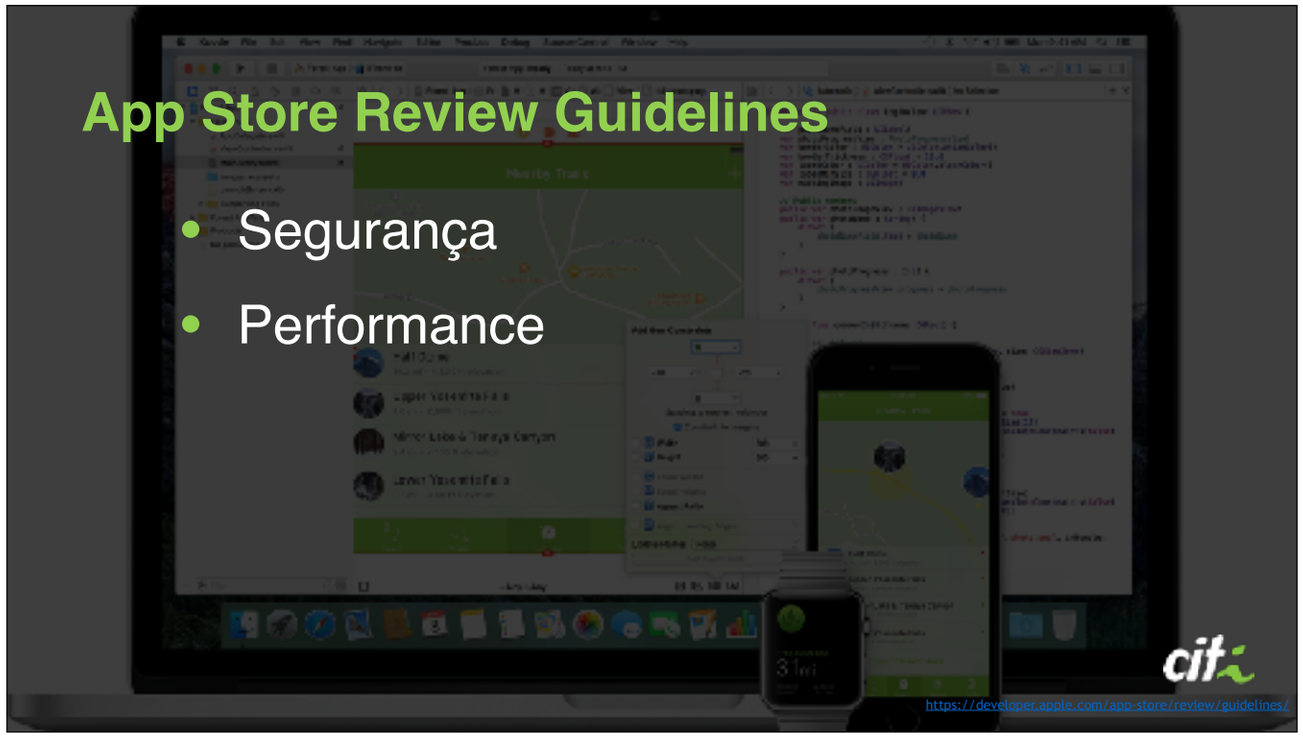
App Store Review Guidelines

- Segurança

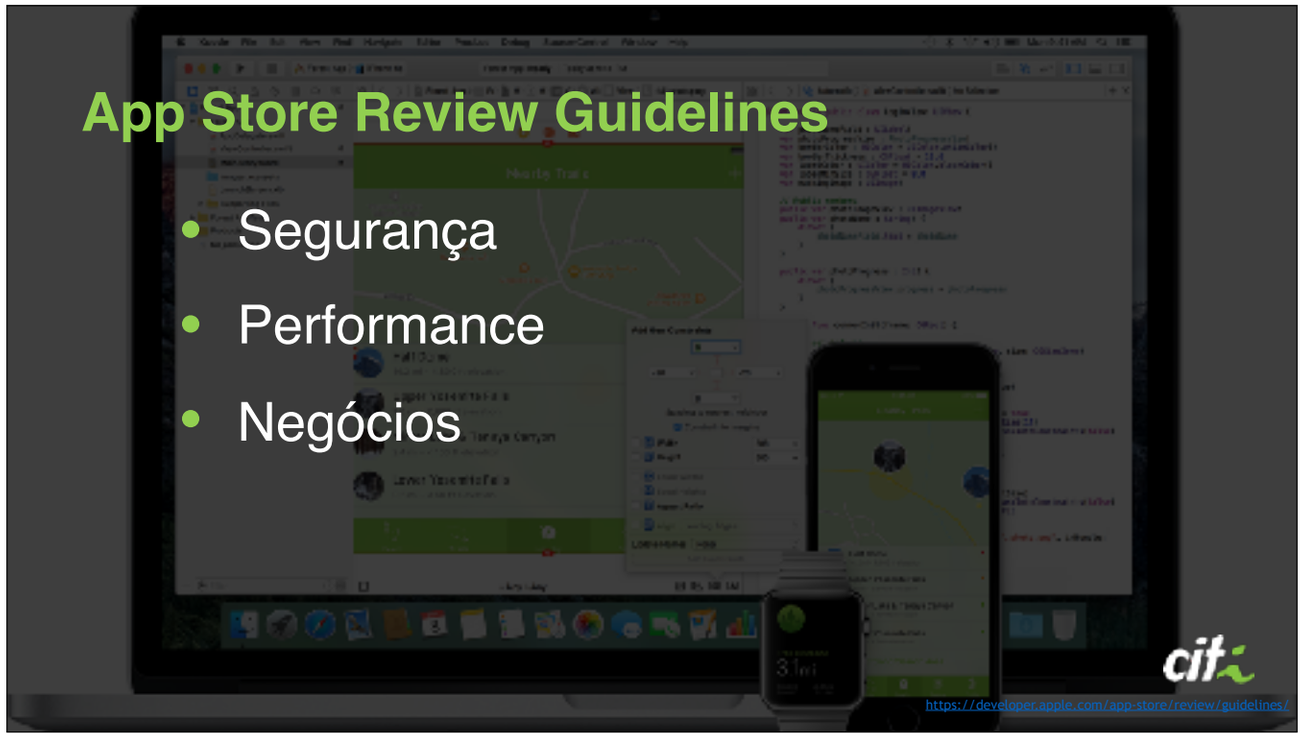


https://developer.apple.com/app-store/review/guidelines/

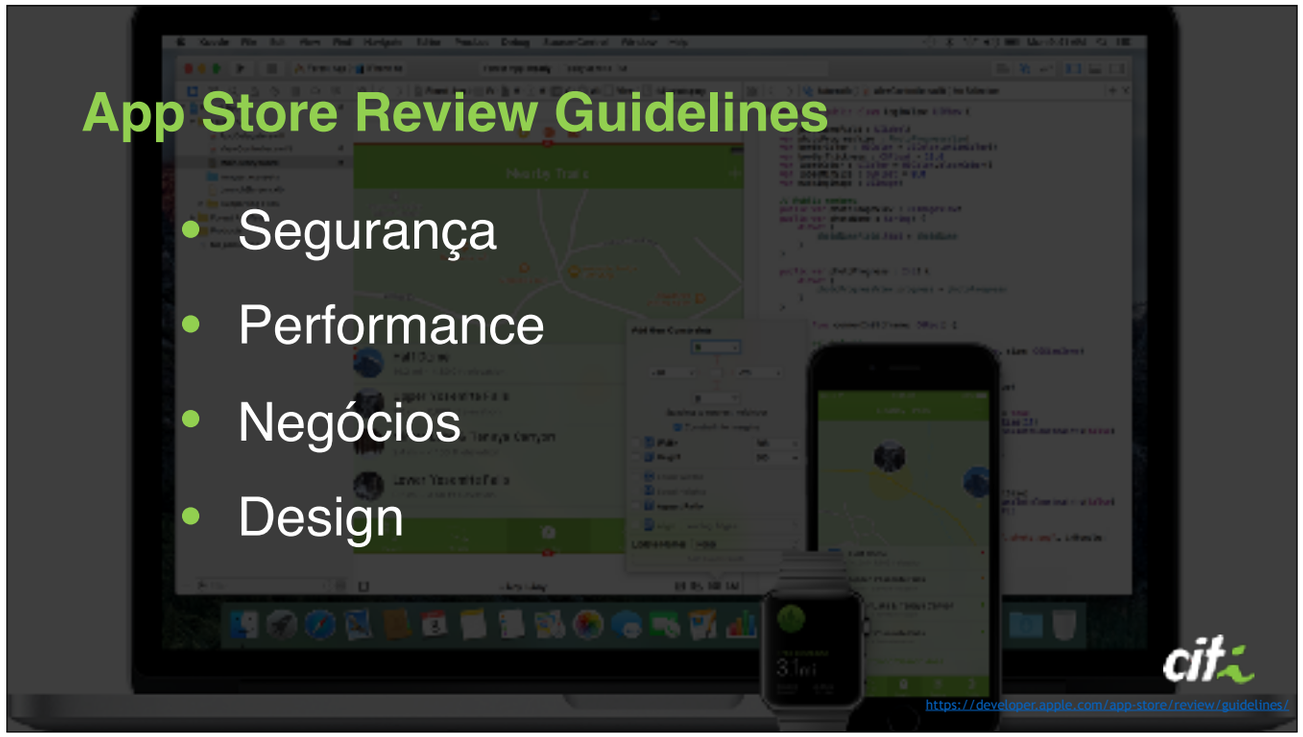
1. Safety
    - 1.1 Objectionable Content
    - 1.2 User Generated Content
    - 1.3 Kids Category
    - 1.4 Physical Harm
    - 1.5 Developer Information
2. Performance
    - 2.1 App Completeness
    - 2.2 Beta Testing
    - 2.3 Accurate Metadata
    - 2.4 Hardware Compatibility
    - 2.5 Software Requirements
3. Business
    - 3.1 Payments
    - 3.1.1 In-App Purchase
    - 3.1.2 Subscriptions
    - 3.1.3 Content-based “Reader” Apps
    - 3.1.4 Content Codes
    - 3.1.5 Physical Goods and Services Outside of the App
    - 3.1.6 Apple Pay
    - 3.2 Other Business Model Issues
    - 3.2.1 Acceptable
    - 3.2.2 Unacceptable
4. Design
    - 4.1 Copycats



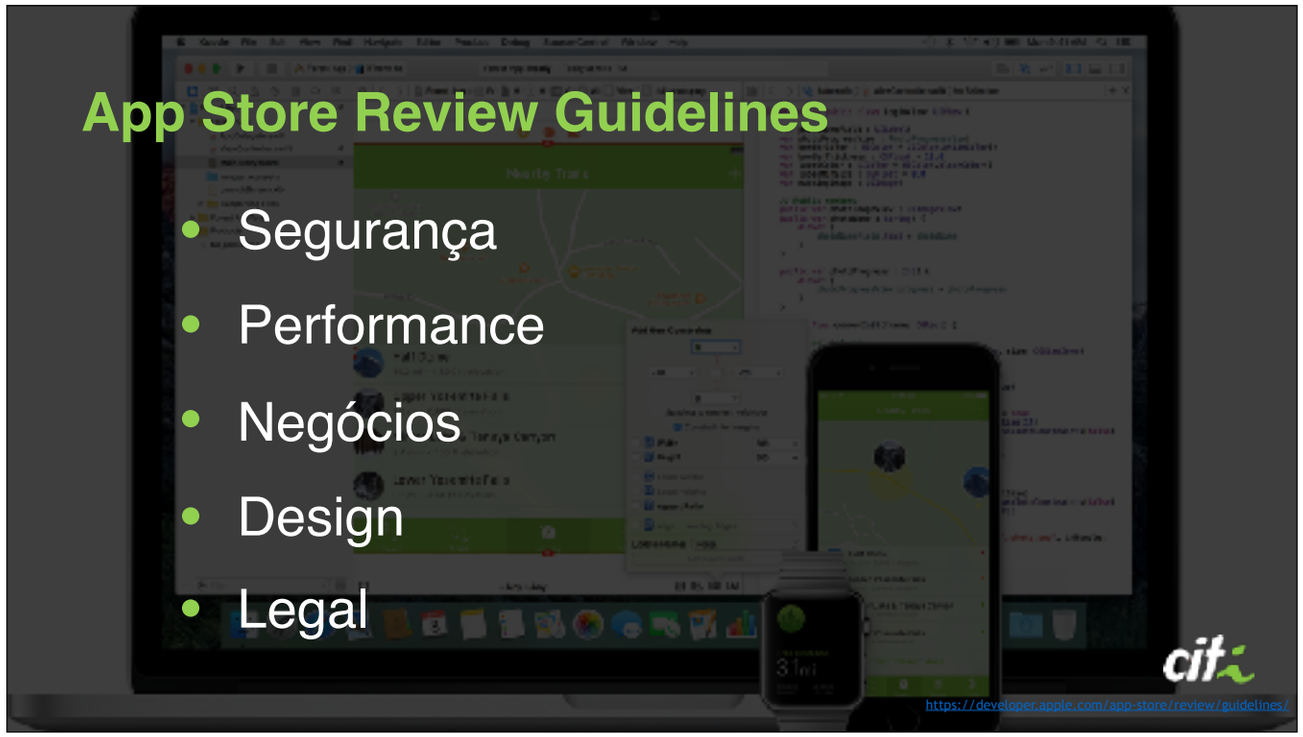
1. Safety
    - 1.1 Objectionable Content
    - 1.2 User Generated Content
    - 1.3 Kids Category
    - 1.4 Physical Harm
    - 1.5 Developer Information
2. Performance
    - 2.1 App Completeness
    - 2.2 Beta Testing
    - 2.3 Accurate Metadata
    - 2.4 Hardware Compatibility
    - 2.5 Software Requirements
3. Business
    - 3.1 Payments
    - 3.1.1 In-App Purchase
    - 3.1.2 Subscriptions
    - 3.1.3 Content-based “Reader” Apps
    - 3.1.4 Content Codes
    - 3.1.5 Physical Goods and Services Outside of the App
    - 3.1.6 Apple Pay
    - 3.2 Other Business Model Issues
    - 3.2.1 Acceptable
    - 3.2.2 Unacceptable
4. Design
    - 4.1 Copycats



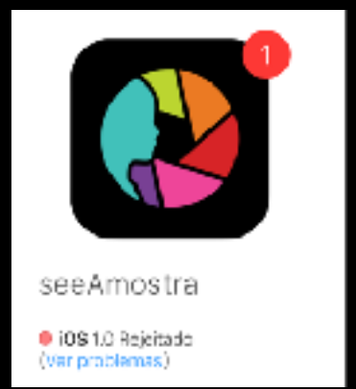
1. Safety
    - 1.1 Objectionable Content
    - 1.2 User Generated Content
    - 1.3 Kids Category
    - 1.4 Physical Harm
    - 1.5 Developer Information
2. Performance
    - 2.1 App Completeness
    - 2.2 Beta Testing
    - 2.3 Accurate Metadata
    - 2.4 Hardware Compatibility
    - 2.5 Software Requirements
3. Business
    - 3.1 Payments
    - 3.1.1 In-App Purchase
    - 3.1.2 Subscriptions
    - 3.1.3 Content-based “Reader” Apps
    - 3.1.4 Content Codes
    - 3.1.5 Physical Goods and Services Outside of the App
    - 3.1.6 Apple Pay
    - 3.2 Other Business Model Issues
    - 3.2.1 Acceptable
    - 3.2.2 Unacceptable
4. Design
    - 4.1 Copycats



1. Safety
    - 1.1 Objectionable Content
    - 1.2 User Generated Content
    - 1.3 Kids Category
    - 1.4 Physical Harm
    - 1.5 Developer Information
2. Performance
    - 2.1 App Completeness
    - 2.2 Beta Testing
    - 2.3 Accurate Metadata
    - 2.4 Hardware Compatibility
    - 2.5 Software Requirements
3. Business
    - 3.1 Payments
    - 3.1.1 In-App Purchase
    - 3.1.2 Subscriptions
    - 3.1.3 Content-based “Reader” Apps
    - 3.1.4 Content Codes
    - 3.1.5 Physical Goods and Services Outside of the App
    - 3.1.6 Apple Pay
    - 3.2 Other Business Model Issues
    - 3.2.1 Acceptable
    - 3.2.2 Unacceptable
4. Design
    - 4.1 Copycats



1. Safety
    - 1.1 Objectionable Content
    - 1.2 User Generated Content
    - 1.3 Kids Category
    - 1.4 Physical Harm
    - 1.5 Developer Information
2. Performance
    - 2.1 App Completeness
    - 2.2 Beta Testing
    - 2.3 Accurate Metadata
    - 2.4 Hardware Compatibility
    - 2.5 Software Requirements
3. Business
    - 3.1 Payments
    - 3.1.1 In-App Purchase
    - 3.1.2 Subscriptions
    - 3.1.3 Content-based “Reader” Apps
    - 3.1.4 Content Codes
    - 3.1.5 Physical Goods and Services Outside of the App
    - 3.1.6 Apple Pay
    - 3.2 Other Business Model Issues
    - 3.2.1 Acceptable
    - 3.2.2 Unacceptable
4. Design
    - 4.1 Copycats



13 de mai de 2016 às 18:54

De Apple

14. PERSONAL ATTACKS

14.3 Details

Your app enables users to post content anonymously but does not have the required precautions in place.

Next Steps

It is necessary that you put all of the following precautions in place:

- Age rating must reflect 17+
- The ability to report a post
- The ability to block a user
- The ability to immediately remove a post from the feed
- Contact information available in the app for users to report inappropriate activity
- A backend mechanism for identifying and blocking users who violate terms and conditions

Aesthetic Integrity
Aesthetic integrity represents how well an app's appearance and behavior integrate with its function. For example, an app that helps people perform a serious task can keep them focused by using subtle, unobtrusive graphics, standard controls, and predictable behaviors. On the other hand, an immersive app, such as a game, can deliver a captivating appearance that promises fun and excitement, while encouraging discovery.

Consistency
A consistent app implements familiar standards and paradigms by using system-provided interface elements, well-known icons, standard text styles, and uniform terminology. The app incorporates features and behaviors in ways people expect.

Direct Manipulation
The direct manipulation of onscreen content engages people and facilitates understanding. Users experience direct manipulation when they rotate the device or use gestures to affect onscreen content. Through direct manipulation, they can see the immediate, visible results of their actions.

Feedback
Feedback acknowledges actions and shows results to keep people informed. The built-in iOS apps provide perceptible feedback in response to every user action. Interactive elements are highlighted briefly when tapped, progress indicators communicate the status of long-running operations, and animation and sound help clarify the results of actions.

Metaphors
People learn more quickly when an app's virtual objects and actions are metaphors for familiar experiences—whether rooted in the real or digital world. Metaphors work well in iOS because people physically interact with the screen. They move views out of the way to expose content beneath. They drag and swipe content. They toggle switches, move sliders, and scroll through picker values. They even flick through pages of books and magazines.

User Control
Throughout iOS, people—not apps—are in control. An app can suggest a course of action or warn about dangerous consequences, but it's usually a mistake for the app to take over the decision-making. The best apps find the correct balance between enabling users and avoiding unwanted outcomes. An app can make people feel like they're in control by keeping interactive elements familiar and predictable, confirming destructive actions, and making it easy to cancel operations, even when they're already underway.

Aesthetic Integrity
Aesthetic integrity represents how well an app's appearance and behavior integrate with its function. For example, an app that helps people perform a serious task can keep them focused by using subtle, unobtrusive graphics, standard controls, and predictable behaviors. On the other hand, an immersive app, such as a game, can deliver a captivating appearance that promises fun and excitement, while encouraging discovery.

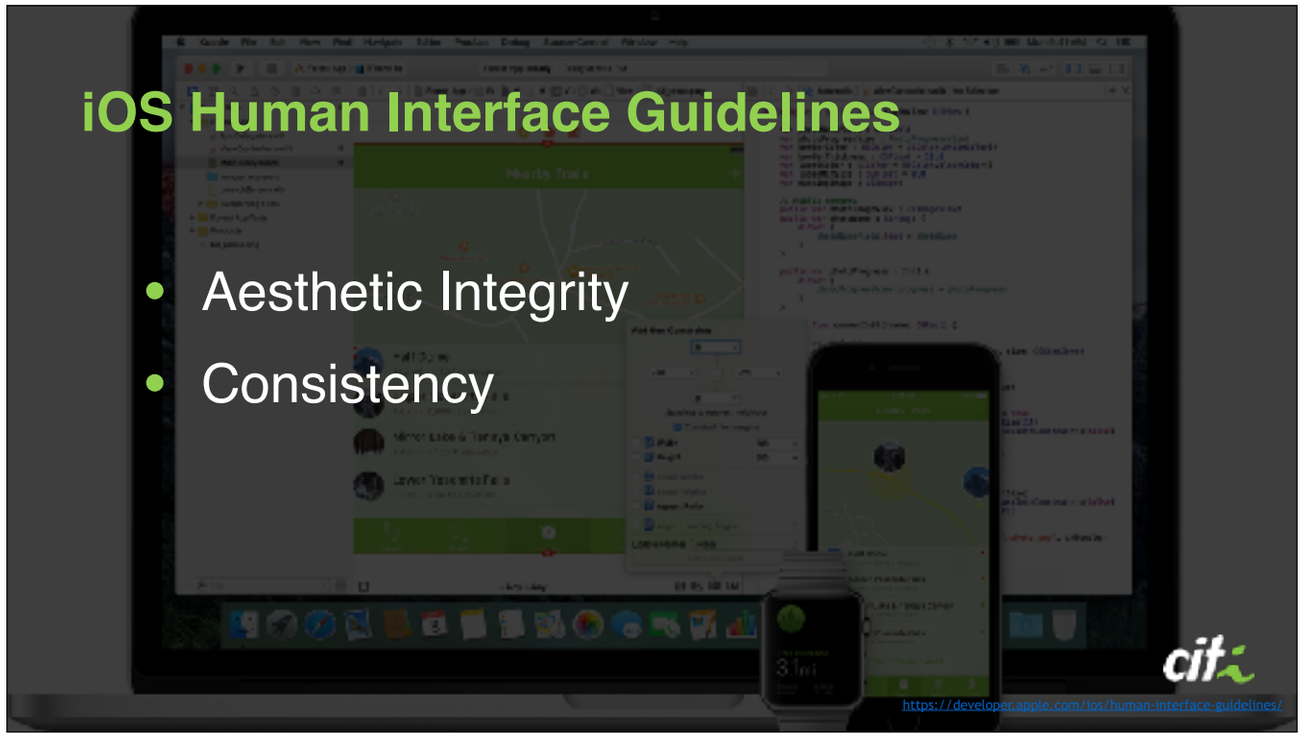
Consistency
A consistent app implements familiar standards and paradigms by using system-provided interface elements, well-known icons, standard text styles, and uniform terminology. The app incorporates features and behaviors in ways people expect.

Direct Manipulation
The direct manipulation of onscreen content engages people and facilitates understanding. Users experience direct manipulation when they rotate the device or use gestures to affect onscreen content. Through direct manipulation, they can see the immediate, visible results of their actions.

Feedback
Feedback acknowledges actions and shows results to keep people informed. The built-in iOS apps provide perceptible feedback in response to every user action. Interactive elements are highlighted briefly when tapped, progress indicators communicate the status of long-running operations, and animation and sound help clarify the results of actions.

Metaphors
People learn more quickly when an app's virtual objects and actions are metaphors for familiar experiences—whether rooted in the real or digital world. Metaphors work well in iOS because people physically interact with the screen. They move views out of the way to expose content beneath. They drag and swipe content. They toggle switches, move sliders, and scroll through picker values. They even flick through pages of books and magazines.

User Control
Throughout iOS, people—not apps—are in control. An app can suggest a course of action or warn about dangerous consequences, but it's usually a mistake for the app to take over the decision-making. The best apps find the correct balance between enabling users and avoiding unwanted outcomes. An app can make people feel like they're in control by keeping interactive elements familiar and predictable, confirming destructive actions, and making it easy to cancel operations, even when they're already underway.



Aesthetic Integrity
Aesthetic integrity represents how well an app's appearance and behavior integrate with its function. For example, an app that helps people perform a serious task can keep them focused by using subtle, unobtrusive graphics, standard controls, and predictable behaviors. On the other hand, an immersive app, such as a game, can deliver a captivating appearance that promises fun and excitement, while encouraging discovery.

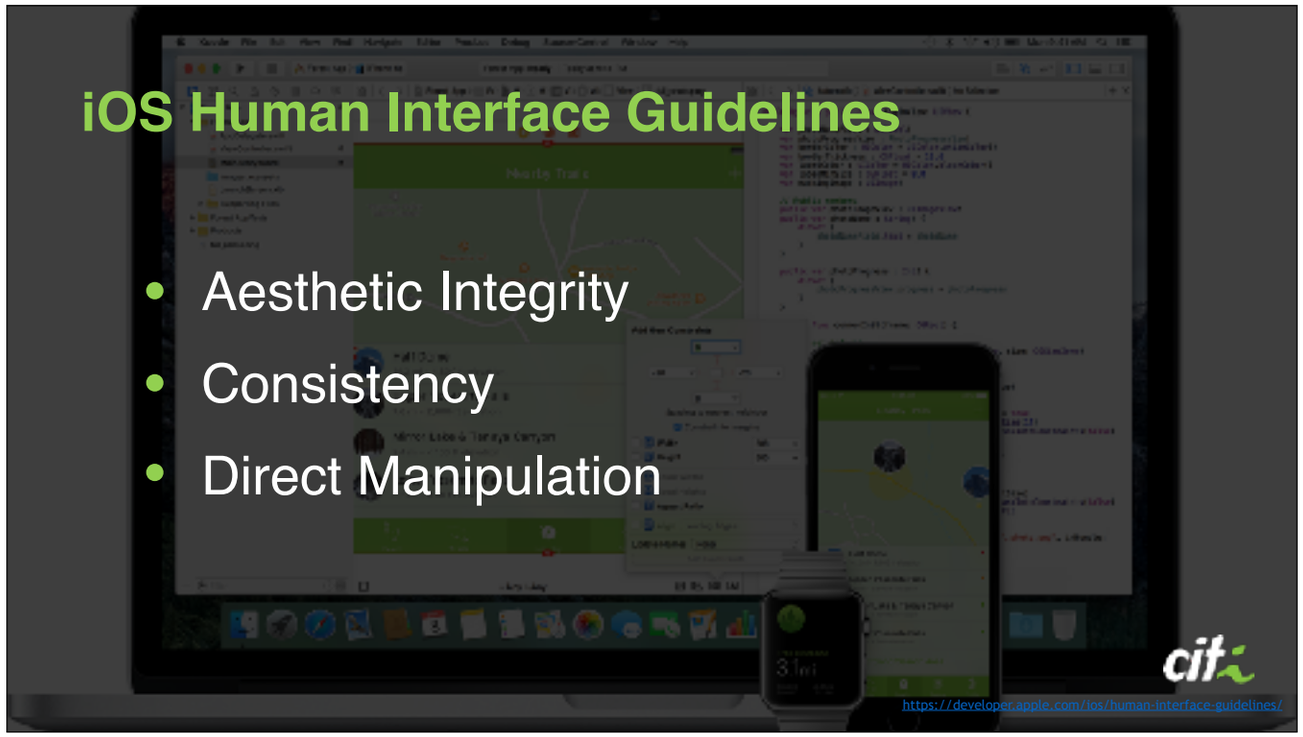
Consistency
A consistent app implements familiar standards and paradigms by using system-provided interface elements, well-known icons, standard text styles, and uniform terminology. The app incorporates features and behaviors in ways people expect.

Direct Manipulation
The direct manipulation of onscreen content engages people and facilitates understanding. Users experience direct manipulation when they rotate the device or use gestures to affect onscreen content. Through direct manipulation, they can see the immediate, visible results of their actions.

Feedback
Feedback acknowledges actions and shows results to keep people informed. The built-in iOS apps provide perceptible feedback in response to every user action. Interactive elements are highlighted briefly when tapped, progress indicators communicate the status of long-running operations, and animation and sound help clarify the results of actions.

Metaphors
People learn more quickly when an app's virtual objects and actions are metaphors for familiar experiences—whether rooted in the real or digital world. Metaphors work well in iOS because people physically interact with the screen. They move views out of the way to expose content beneath. They drag and swipe content. They toggle switches, move sliders, and scroll through picker values. They even flick through pages of books and magazines.

User Control
Throughout iOS, people—not apps—are in control. An app can suggest a course of action or warn about dangerous consequences, but it's usually a mistake for the app to take over the decision-making. The best apps find the correct balance between enabling users and avoiding unwanted outcomes. An app can make people feel like they're in control by keeping interactive elements familiar and predictable, confirming destructive actions, and making it easy to cancel operations, even when they're already underway.



Aesthetic Integrity
Aesthetic integrity represents how well an app's appearance and behavior integrate with its function. For example, an app that helps people perform a serious task can keep them focused by using subtle, unobtrusive graphics, standard controls, and predictable behaviors. On the other hand, an immersive app, such as a game, can deliver a captivating appearance that promises fun and excitement, while encouraging discovery.

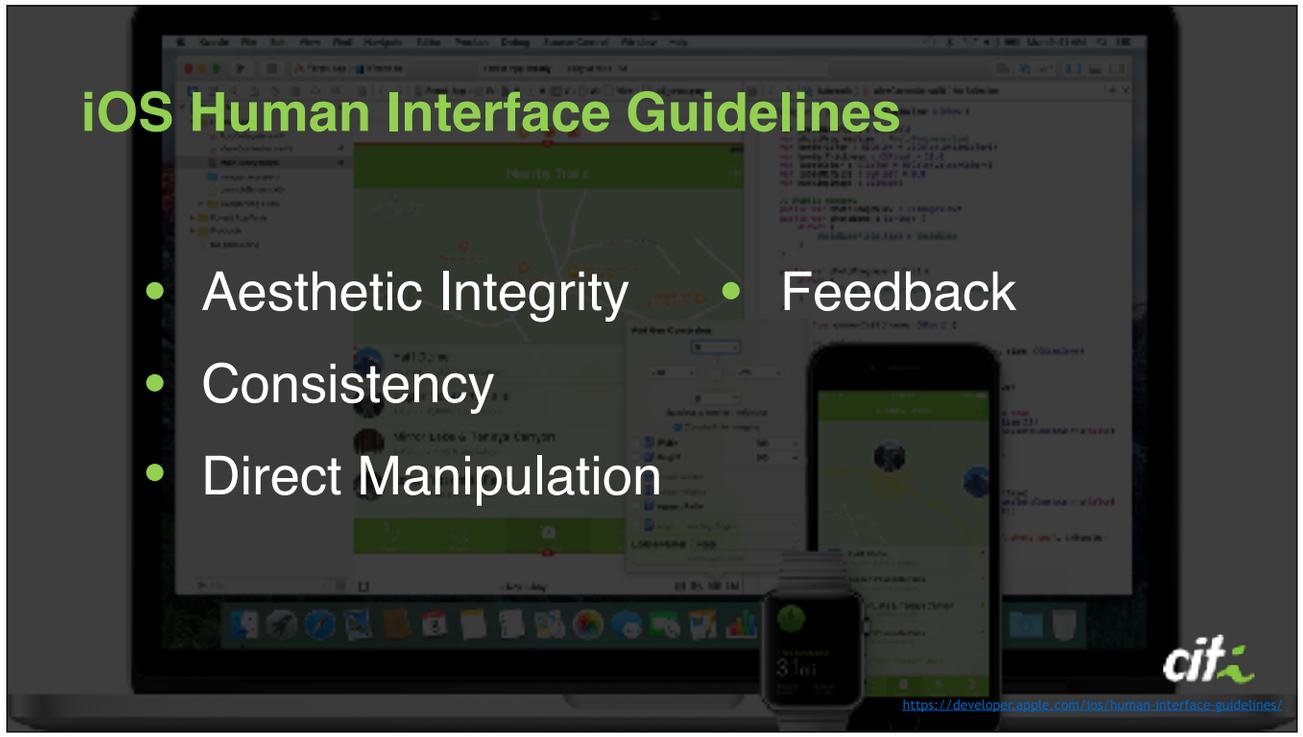
Consistency
A consistent app implements familiar standards and paradigms by using system-provided interface elements, well-known icons, standard text styles, and uniform terminology. The app incorporates features and behaviors in ways people expect.

Direct Manipulation
The direct manipulation of onscreen content engages people and facilitates understanding. Users experience direct manipulation when they rotate the device or use gestures to affect onscreen content. Through direct manipulation, they can see the immediate, visible results of their actions.

Feedback
Feedback acknowledges actions and shows results to keep people informed. The built-in iOS apps provide perceptible feedback in response to every user action. Interactive elements are highlighted briefly when tapped, progress indicators communicate the status of long-running operations, and animation and sound help clarify the results of actions.

Metaphors
People learn more quickly when an app's virtual objects and actions are metaphors for familiar experiences—whether rooted in the real or digital world. Metaphors work well in iOS because people physically interact with the screen. They move views out of the way to expose content beneath. They drag and swipe content. They toggle switches, move sliders, and scroll through picker values. They even flick through pages of books and magazines.

User Control
Throughout iOS, people—not apps—are in control. An app can suggest a course of action or warn about dangerous consequences, but it's usually a mistake for the app to take over the decision-making. The best apps find the correct balance between enabling users and avoiding unwanted outcomes. An app can make people feel like they're in control by keeping interactive elements familiar and predictable, confirming destructive actions, and making it easy to cancel operations, even when they're already underway.



Aesthetic Integrity
Aesthetic integrity represents how well an app's appearance and behavior integrate with its function. For example, an app that helps people perform a serious task can keep them focused by using subtle, unobtrusive graphics, standard controls, and predictable behaviors. On the other hand, an immersive app, such as a game, can deliver a captivating appearance that promises fun and excitement, while encouraging discovery.

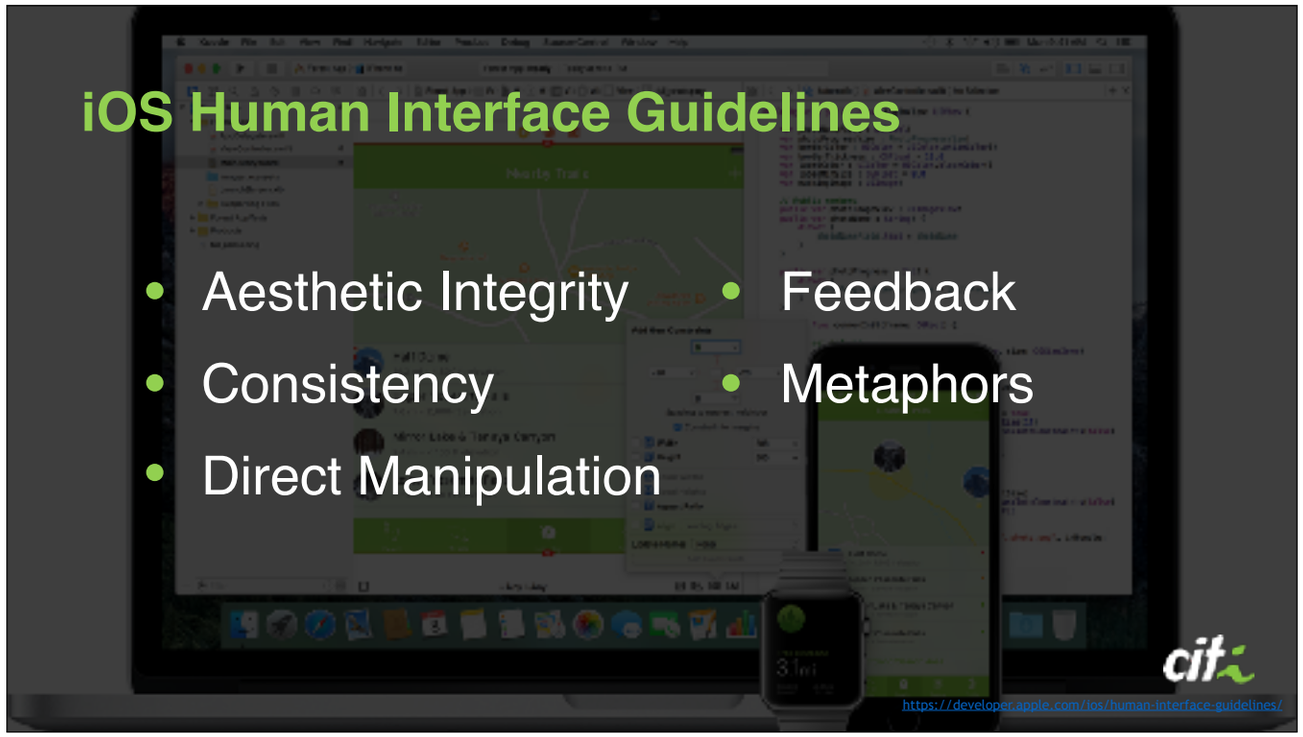
Consistency
A consistent app implements familiar standards and paradigms by using system-provided interface elements, well-known icons, standard text styles, and uniform terminology. The app incorporates features and behaviors in ways people expect.

Direct Manipulation
The direct manipulation of onscreen content engages people and facilitates understanding. Users experience direct manipulation when they rotate the device or use gestures to affect onscreen content. Through direct manipulation, they can see the immediate, visible results of their actions.

Feedback
Feedback acknowledges actions and shows results to keep people informed. The built-in iOS apps provide perceptible feedback in response to every user action. Interactive elements are highlighted briefly when tapped, progress indicators communicate the status of long-running operations, and animation and sound help clarify the results of actions.

Metaphors
People learn more quickly when an app's virtual objects and actions are metaphors for familiar experiences—whether rooted in the real or digital world. Metaphors work well in iOS because people physically interact with the screen. They move views out of the way to expose content beneath. They drag and swipe content. They toggle switches, move sliders, and scroll through picker values. They even flick through pages of books and magazines.

User Control
Throughout iOS, people—not apps—are in control. An app can suggest a course of action or warn about dangerous consequences, but it's usually a mistake for the app to take over the decision-making. The best apps find the correct balance between enabling users and avoiding unwanted outcomes. An app can make people feel like they're in control by keeping interactive elements familiar and predictable, confirming destructive actions, and making it easy to cancel operations, even when they're already underway.



Aesthetic Integrity
Aesthetic integrity represents how well an app's appearance and behavior integrate with its function. For example, an app that helps people perform a serious task can keep them focused by using subtle, unobtrusive graphics, standard controls, and predictable behaviors. On the other hand, an immersive app, such as a game, can deliver a captivating appearance that promises fun and excitement, while encouraging discovery.

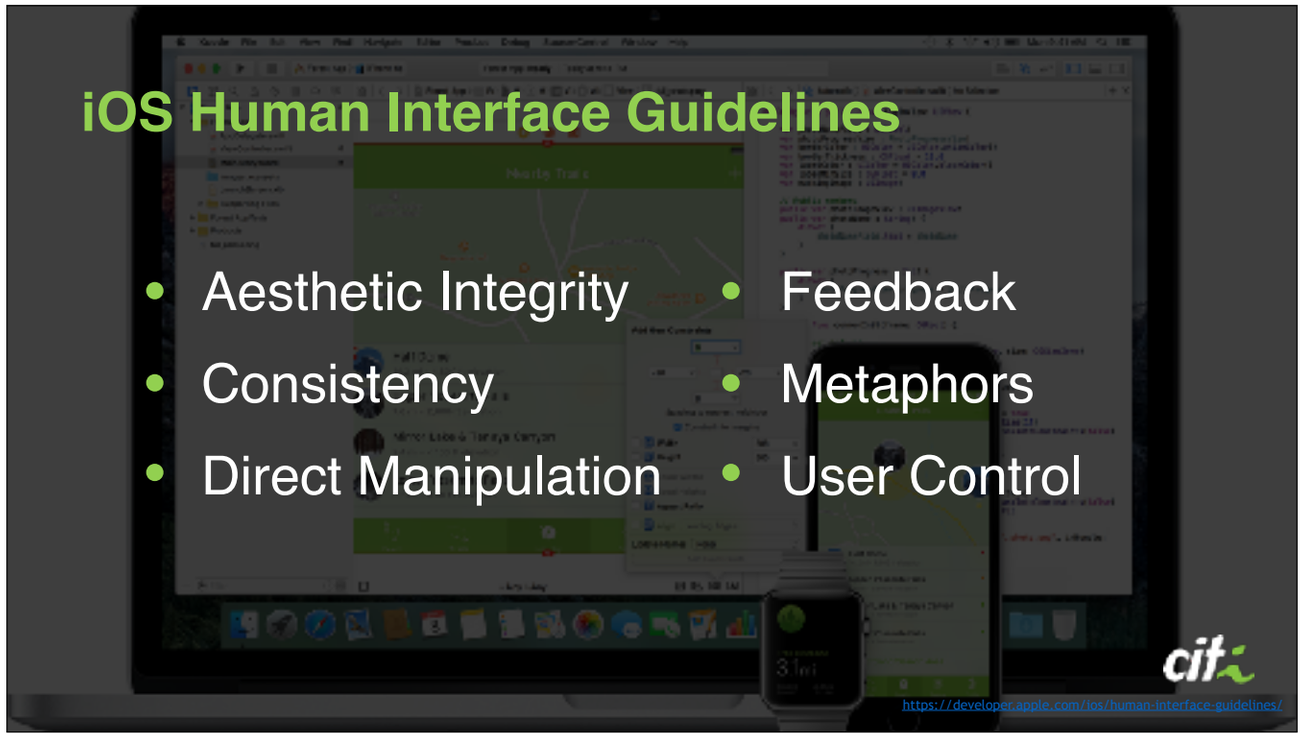
Consistency
A consistent app implements familiar standards and paradigms by using system-provided interface elements, well-known icons, standard text styles, and uniform terminology. The app incorporates features and behaviors in ways people expect.

Direct Manipulation
The direct manipulation of onscreen content engages people and facilitates understanding. Users experience direct manipulation when they rotate the device or use gestures to affect onscreen content. Through direct manipulation, they can see the immediate, visible results of their actions.

Feedback
Feedback acknowledges actions and shows results to keep people informed. The built-in iOS apps provide perceptible feedback in response to every user action. Interactive elements are highlighted briefly when tapped, progress indicators communicate the status of long-running operations, and animation and sound help clarify the results of actions.

Metaphors
People learn more quickly when an app's virtual objects and actions are metaphors for familiar experiences—whether rooted in the real or digital world. Metaphors work well in iOS because people physically interact with the screen. They move views out of the way to expose content beneath. They drag and swipe content. They toggle switches, move sliders, and scroll through picker values. They even flick through pages of books and magazines.

User Control
Throughout iOS, people—not apps—are in control. An app can suggest a course of action or warn about dangerous consequences, but it's usually a mistake for the app to take over the decision-making. The best apps find the correct balance between enabling users and avoiding unwanted outcomes. An app can make people feel like they're in control by keeping interactive elements familiar and predictable, confirming destructive actions, and making it easy to cancel operations, even when they're already underway.



Aesthetic Integrity
Aesthetic integrity represents how well an app's appearance and behavior integrate with its function. For example, an app that helps people perform a serious task can keep them focused by using subtle, unobtrusive graphics, standard controls, and predictable behaviors. On the other hand, an immersive app, such as a game, can deliver a captivating appearance that promises fun and excitement, while encouraging discovery.

Consistency
A consistent app implements familiar standards and paradigms by using system-provided interface elements, well-known icons, standard text styles, and uniform terminology. The app incorporates features and behaviors in ways people expect.

Direct Manipulation
The direct manipulation of onscreen content engages people and facilitates understanding. Users experience direct manipulation when they rotate the device or use gestures to affect onscreen content. Through direct manipulation, they can see the immediate, visible results of their actions.

Feedback
Feedback acknowledges actions and shows results to keep people informed. The built-in iOS apps provide perceptible feedback in response to every user action. Interactive elements are highlighted briefly when tapped, progress indicators communicate the status of long-running operations, and animation and sound help clarify the results of actions.

Metaphors
People learn more quickly when an app's virtual objects and actions are metaphors for familiar experiences—whether rooted in the real or digital world. Metaphors work well in iOS because people physically interact with the screen. They move views out of the way to expose content beneath. They drag and swipe content. They toggle switches, move sliders, and scroll through picker values. They even flick through pages of books and magazines.

User Control
Throughout iOS, people—not apps—are in control. An app can suggest a course of action or warn about dangerous consequences, but it's usually a mistake for the app to take over the decision-making. The best apps find the correct balance between enabling users and avoiding unwanted outcomes. An app can make people feel like they're in control by keeping interactive elements familiar and predictable, confirming destructive actions, and making it easy to cancel operations, even when they're already underway.

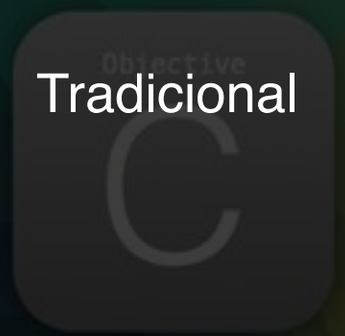

Objective
C

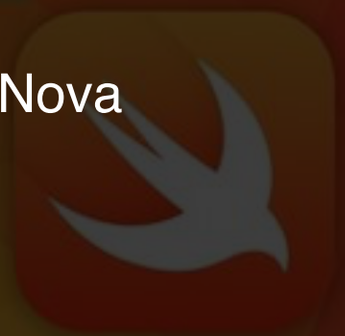
Objective-C vs Swift

citi

Objective-C vs Swift

- Tradicional
- Nova





citi

Objective-C vs Swift

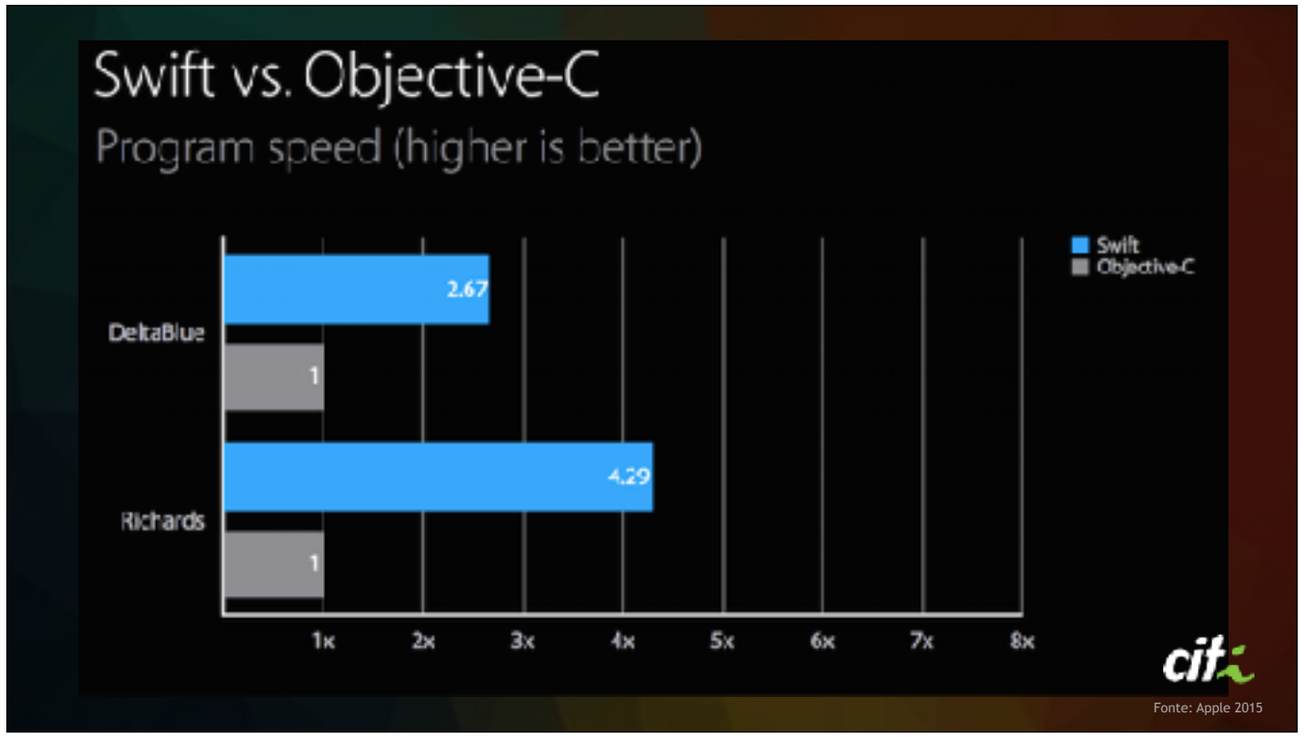
- Tradicional
- Menos Mudanças

- Nova
- Open source

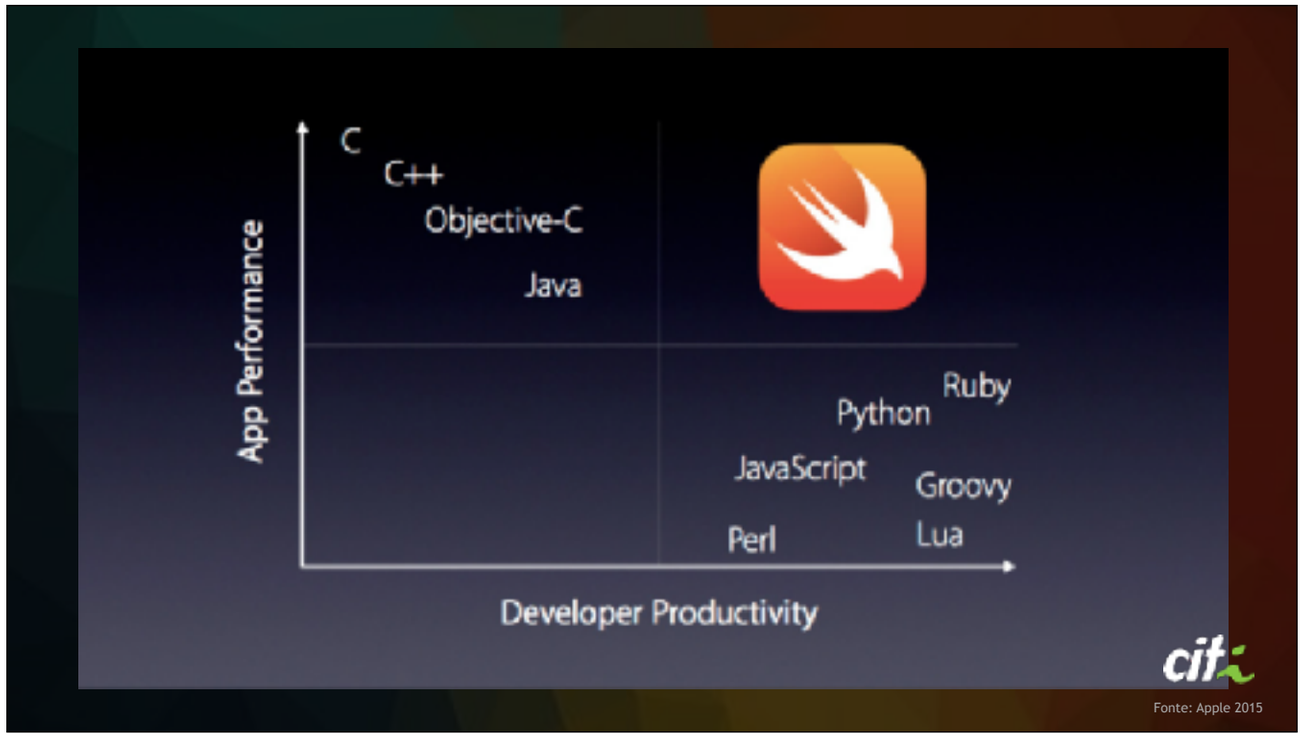
Objective-C vs Swift

- Tradicional
- Menos Mudanças
- Mais Adotada

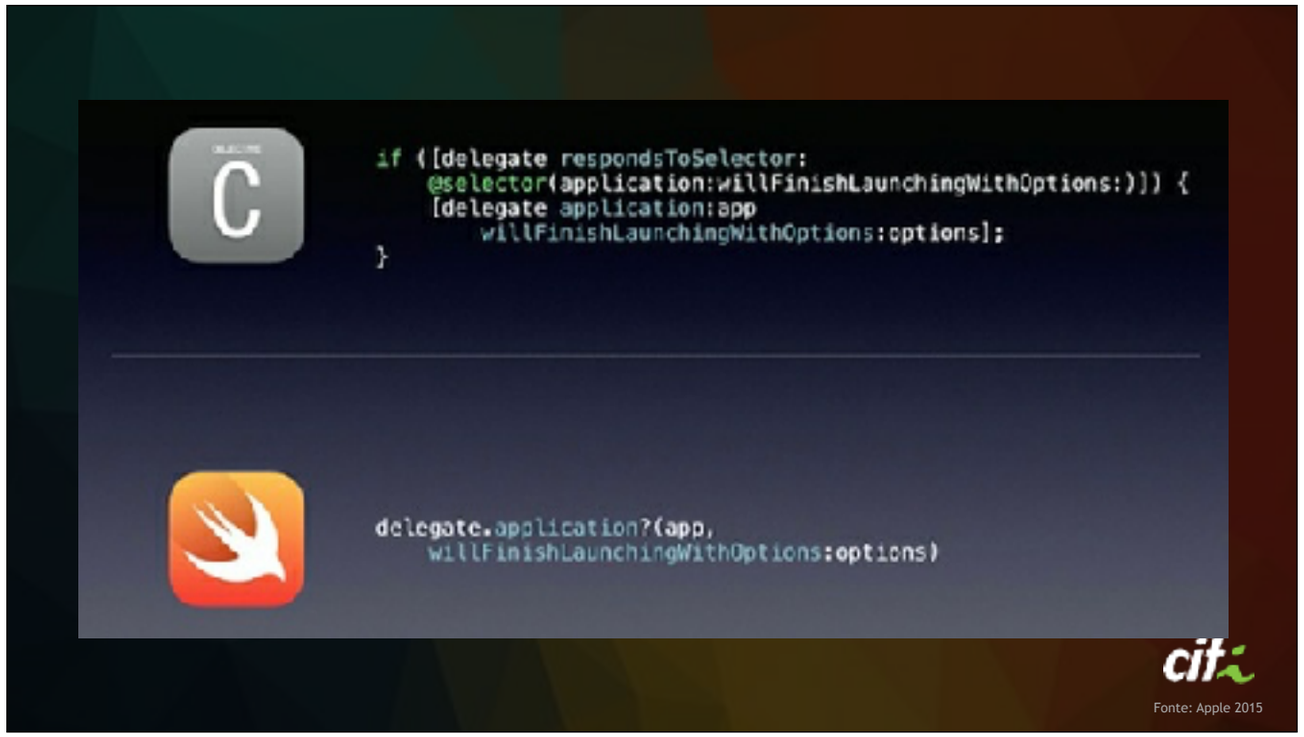
- Nova
- Open source
- Adotada pela Apple



Fonte: Apple 2015

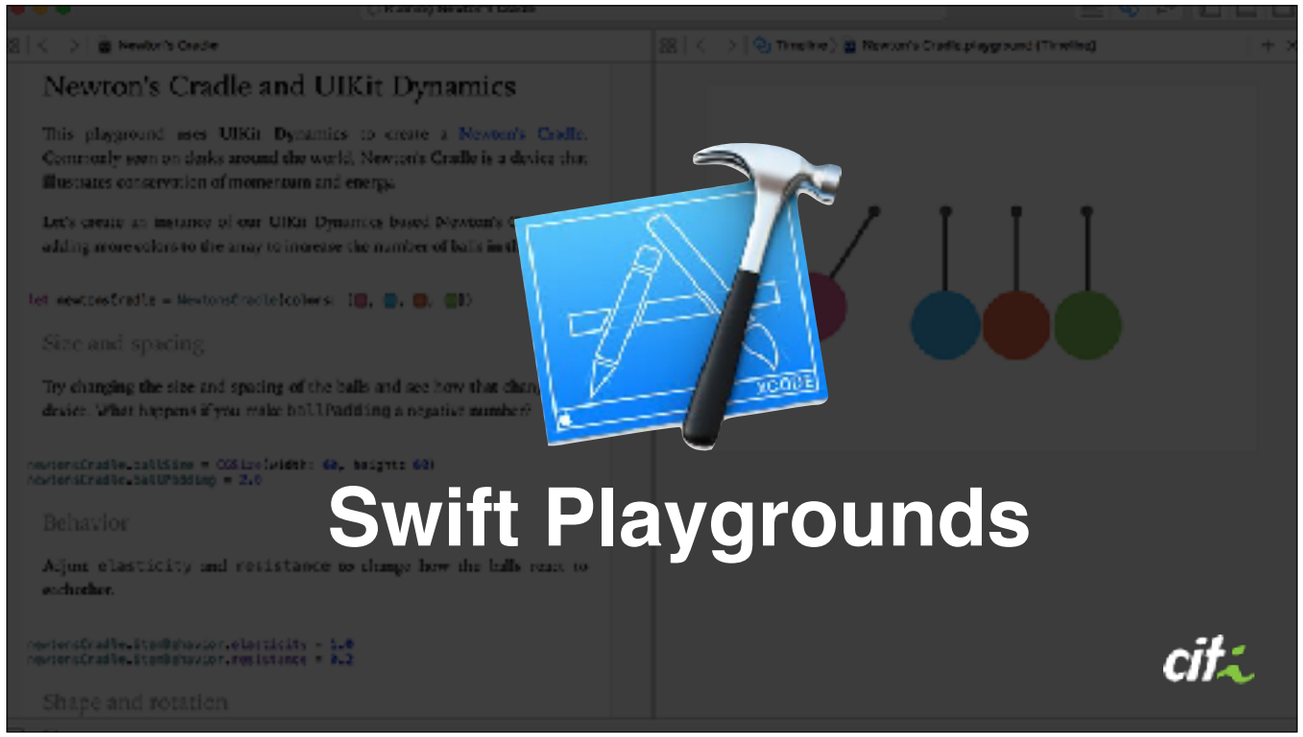


Fonte: Apple 2015



Fonte: Apple 2015


Swift
Xcode
citi


Newton's Cradle
Newton's Cradle playground (Timeline)
Newton's Cradle and UIKit Dynamics
Size and spacing
Behavior
Shape and rotation
Swift Playgrounds
citi

```
// Comentários em Swift

/*
   Comentários
   com várias linhas
*/
```

// Memória

```
// Variáveis

var nome = "Hilton"
var idade: Int = 21
```

// Variáveis

```
var nome = "Hilton"
var idade: Int = 21
```

Palavra-reservada
*(**keyword**)*

// Variáveis

```
var nome = "Hilton"
var idade: Int = 21
```

Palavra-reservada *(**keyword**)*

Identificador

// Variáveis

```
var nome = "Hilton"
var idade: Int = 21
```

Palavra-reservada *(**keyword**)* **Identificador** **Tipo**

// Variáveis

```
var nome = "Hilton"
var idade: Int = 21
```

Palavra-reservada *(keyword)* | **Identificador** | **Tipo** | **operador de atribuição**

// Variáveis

```
var nome = "Hilton"
var idade: Int = 21
```

Palavra-reservada *(**keyword**)* **Identificador** **Tipo** Operador de **atribuição** Valor **(String)** Valor **(Int)**

// Atribuição de novo valor

idade = 22

Palavra-reservada *(**keyword**)* **Identificador** **Tipo** Operador de **atribuição** Valor **(Int)**

// Atribuição de novo valor

idade = 22

Palavra reservada (*keyword*) **Identificador** Operador de **atribuição** Valor **(Int)**

```
// Constantes


let diaNascimento = 2
let nacionalidade: String = "Brasileiro"
```

```
// Constantes


let diaNascimento = 2
let nacionalidade: String = "Brasileiro"
```

Palavra-reservada
(keyword)

// Constantes

```
let diaNascimento = 2
let nacionalidade: String = "Brasileiro"
```

Palavra-reservada
(keyword)

Identificador

```
// Constantes


let diaNascimento = 2
let nacionalidade: String = "Brasileiro"
```

Palavra-reservada *(**keyword**)*

Identificador

Tipo

// Constantes

```
let diaNascimento = 2
let nacionalidade: String = "Brasileiro"
```

Palavra-reservada *(**keyword**)*

Identificador

Tipo

Operador de **atribuição**

// Constantes

```
let diaNascimento = 2
let nacionalidade: String = "Brasileiro"
```

Palavra-reservada *(**keyword**)*

Identificador

Tipo

Operador de **atribuição**

Valor **(Int)**

Valor **(String)**

// Tipos

```
// Tipos numéricos


let inteiro: Int = 10
let decimal: Double = 3.14159265359
let pi: Float = 3.14159265359
```

Valor **(Int)**
Valor **(Double)**
Valor **(Float)**

```swift
// Tipos numéricos


let inteiro: Int = 10
let decimal: Double = 3.14159265359
let pi: Float = 3.14159265359
                //arredondado para 3.141593
```

// Booleans

```
let souRecifense: Bool = true
var odeioSwift = false
```

Valor **(Bool)**

// Strings

```
let sobrenome: String = “Pintor"
let cpf = "111-404-222.16"
```

Valor **(String)**

```swift
// Tuplas

let estado: (String, String) = ("Recife", "PE")
```

// Tuplas

```
let estado: (String, String) = ("Recife", "PE")
estado.0 // acessando primeiro valor da tupla estado
```

Identificador **.** **Índice**

Ponto

```swift
// Tuplas

let rg = (numero: 9153865, orgao: "SDS", UF: "PE")
```

```
// Tuplas


let rg = (numero: 9153865, orgao: "SDS", UF: "PE")
rg.numero // acessando valor "numero" da tupla rg
```

Identificador **.** **Label**

Ponto

// Optionals

```
var sentidoVida: Int?
sentidoVida // nesse ponto não tem valor (nil)
```

Tipo **?**
Interrogação

```
// Optionals

sentidoVida = 42
sentidoVida // nesse ponto o valor é 42
```

```
// Forced unwrapping

var titulo: String?
titulo = "PhD"
var unwrapped = titulo! // extrai o valor do opcional
```

titulo é do tipo **String?**
unwrapped é do tipo **String**

```swift
// Optional binding

var opcional: Bool?
opcional = true

if let concreto = opcional {
    concreto // valor extraído: true
}
```

opcional é do tipo **Bool?**
concreto é do tipo **Bool**

// Operadores

```
// Atribuição

nome = “Hilton"
idade = 21
```

```swift
// Aritmética

let dez: Int = 10
let dois: Int = 2
```

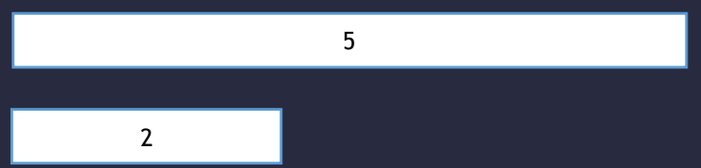
// Aritmética

```
dez + dois   // 12
dez - dois   // 8
dez / dois   // 5
dez * dois   // 20
```

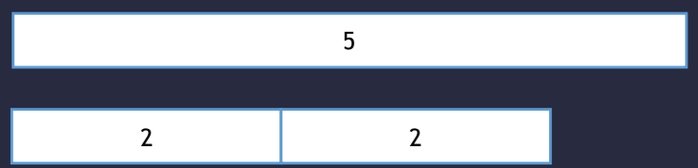
dez e **dois** são do mesmo tipo

```
// Resto

5 % 2  // 1
5 % 5  // 0
2 % 5  // 2
```

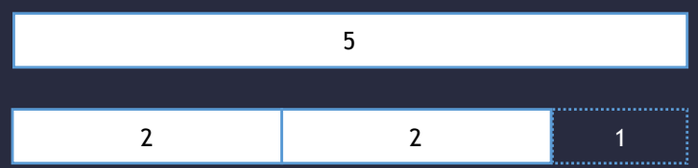
// Resto



// Resto



// Resto



5 % 2 // 1

// Concatenação

```
"olá" + "mundo" // resulta em: "olámundo"
"oi " +  nome   // resulta em: "oi Elton"
```

operador **+** realiza operações diferentes de acordo
com o **tipo dos operandos (Int, String, ...)**

```
// Comparação

2 == 2  // dois é igual a dois? true
10 < 1  // 10 é menor que 1? false
```

```
// Comparação

nome != "Hilton" /* o valor de nome é diferente de
"Hilton"? false */
idade >= 21 /* o valor de idade é maior ou igual a
21? true */
```

```swift
// Negação

odeioSwift = false
!odeioSwift // retorna true
```

```
// Conjunção (And)

true && true    // retorna true
false && true   // retorna false
false && false  // retorna false
```

```
// Disjunção (Or)

true || true    // retorna true
false || true   // retorna true
false || false  // retorna false
```

```
// Intervalos

0...3   // representa 0, 1, 2, 3
0..<3   // representa 0, 1, 2
```

// Coleções

```
// Arrays

var cidades = ["Recife", "Olinda"]
cidades[0]  // retorna "Recife"
```

```
// Arrays

var cidades = [0 "Recife" | 1 "Olinda"]
cidades[0] // retorna "Recife"
```

```
// Arrays - adição

cidades.append("Jaboatão")
cidades[2]  // retorna "Jaboatão"
```

```
// Arrays - adição

| 0 | "Recife" | 1 | "Olinda" |.append("Jaboatão")

cidades[2]  // retorna "Jaboatão"
```

```
// Arrays - adição

| 0 | "Recife" | 1 | "Olinda" |  .append( | 2 | "Jaboatão" |  )

cidades[2]  // retorna "Jaboatão"
```

// Arrays - remoção

0	"Recife"	1	"Olinda"	2	"Jaboatão"

```
cidades.removeLast() // remove "Jaboatão"
cidades.removeFirst() // remove "Recife"
cidades.removeAll()  // remove todos os elementos
```

// Arrays - remoção

0	"Recife"	1	"Olinda"

```
cidades.removeLast() // remove "Jaboatão"
cidades.removeFirst() // remove "Recife"
cidades.removeAll()  // remove todos os elementos
```

// Arrays - remoção

0	"Olinda"

```
cidades.removeLast() // remove "Jaboatão"
cidades.removeFirst() // remove "Recife"
cidades.removeAll()  // remove todos os elementos
```

```
// Arrays - remoção


cidades.removeLast() // remove "Jaboatão"
cidades.removeFirst() // remove "Recife"
cidades.removeAll()   // remove todos os elementos
```

```
// Arrays - contagem

var cidadesVisitadas: [String] = []
cidadesVisitadas.count  // retorna 0 (zero)
```

// Arrays - contagem

```
cidadesVisitadas.append("Recife")
cidadesVisitadas.count  // retorna 1
```

0	"Recife"

// Arrays - modificação

```
cidadesVisitadas[0] = "Olinda"
cidadesVisitadas     // ["Olinda"]
```

0	"Recife"

// Arrays - modificação

```
cidadesVisitadas[0] = "Olinda"
cidadesVisitadas    // ["Olinda"]
```

0	"Olinda"

```
// Arrays - acesso inválido


cidadesVisitadas[cidadesVisitadas.count]
cidades[-1]
```

0	"Olinda"

// Arrays - acesso inválido

```
cidadesVisitadas[cidadesVisitadas.count]
cidades[-1]
```

0	"Olinda"

// Dicionários

```
var capitais: [String: String] = ["PE": "Recife"]
capitais["PE"]    // retorna "Recife"
```

Chave	Valor
“PE”	“Recife”
...	nil

// Dicionários - adição

```
capitais["PB"] = "João Pessoa"
capitais    // ["PB": "João Pessoa", "PE": "Recife"]
```

Chave	Valor
“PE”	“Recife”
“PB”	“João Pessoa”
...	nil

// Dicionários - modificação

```
capitais["PB"] = "Maria Pessoa"
capitais    // ["PB": "Maria Pessoa", "PE": "Recife"]
```

Chave	Valor
“PE”	“Recife”
“PB”	“João Pessoa”
...	nil

// Dicionários - modificação

```
capitais["PB"] = "Maria Pessoa"
capitais    // ["PB": "Maria Pessoa", "PE": "Recife"]
```

Chave	Valor
“PE”	“Recife”
“PB”	“Maria Pessoa”
...	nil

// Dicionários - remoção

```
capitais["PB"] = nil
capitais.removeValue(forKey: "PE")
```

Chave	Valor
“PE”	“Recife”
“PB”	“João Pessoa”
...	nil

// Dicionários - remoção

```
capitais["PB"] = nil
capitais.removeValue(forKey: "PE")
```

Chave	Valor
“PE”	“Recife”
“PB”	nil
...	nil

// Dicionários - remoção

```
capitais["PB"] = nil
capitais.removeValue(forKey: "PE")
```

Chave	Valor
“PE”	nil
“PB”	nil
...	nil

// Dicionários - contagem

```
var dictVazio: [Int: String] = [:]
dictVazio.count // retorna 0
```

Chave	Valor
...	nil

// Dicionários - acesso sem valor

```
dictVazio[0]    // retorna nil
dictVazio[-1]   // retorna nil
```

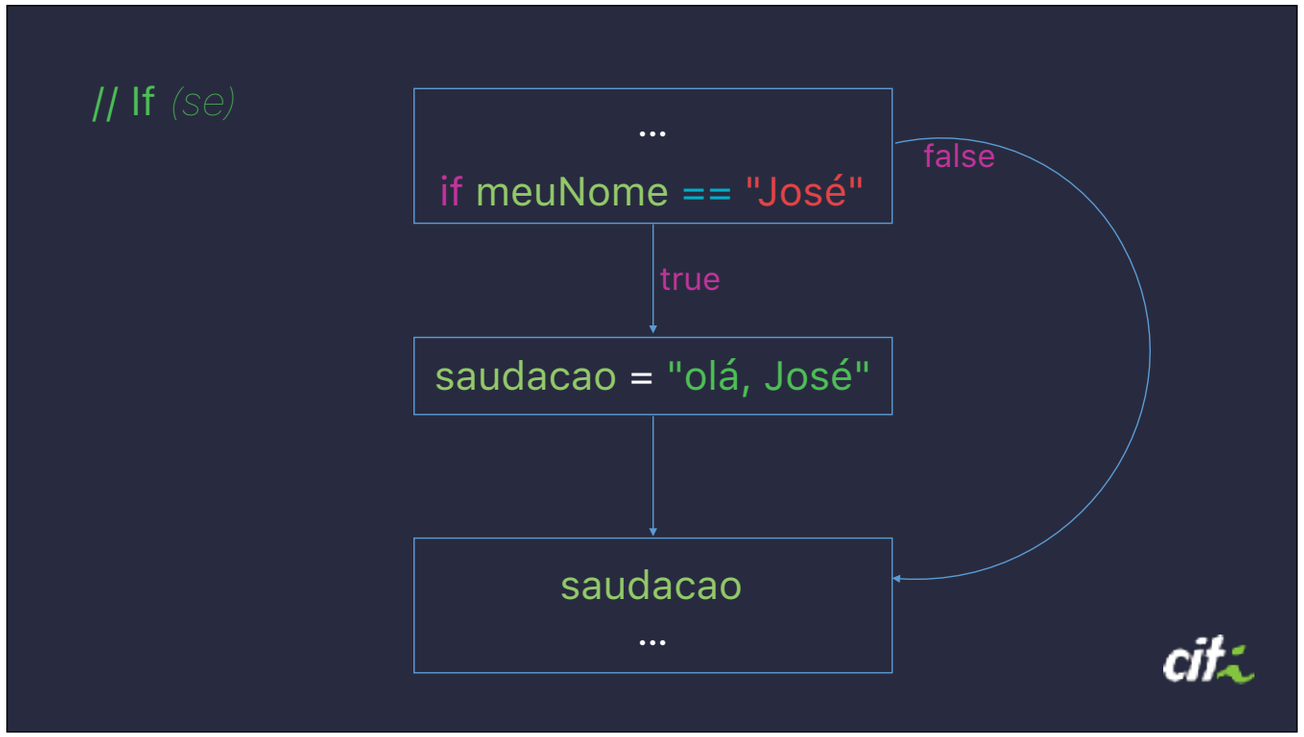
Chave	Valor
0	nil
-1	nil
...	nil

// Controle de Fluxo

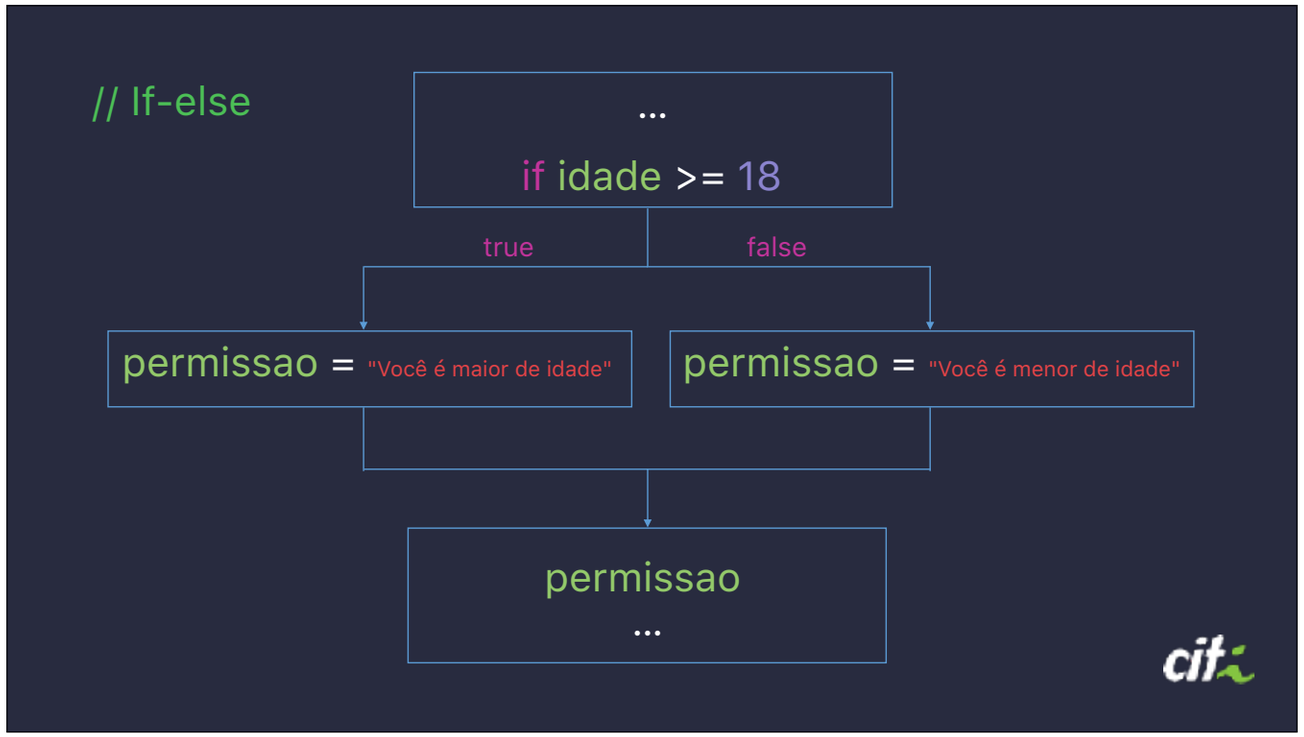
// Desvios Condicionais

```
// If (se)

                var meuNome = "José"
                var saudacao = ""

                if meuNome == "José" {
                    // executar caso verdadeiro:
                    saudacao = "olá, José"
                }

                saudacao    // tem valor de "olá, José"
```

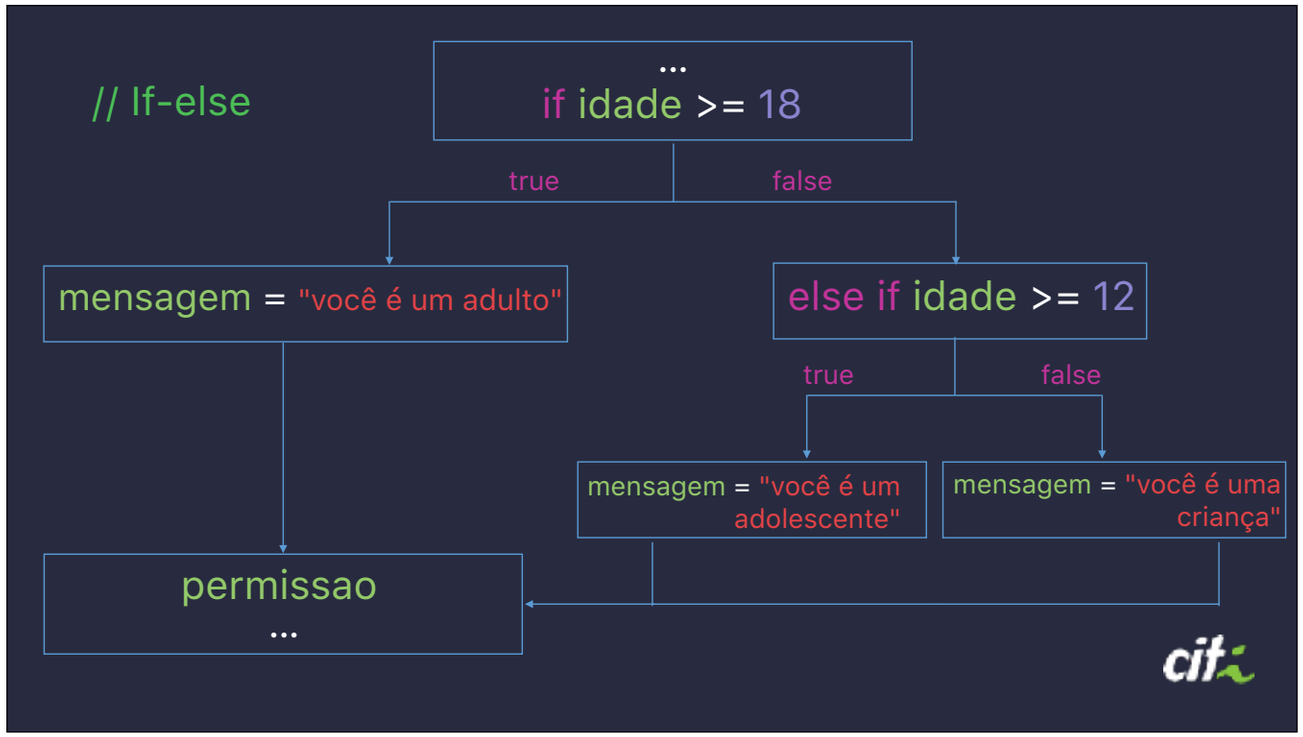
// If *(se)*



```
// If-else      idade = 17
                var permissao = ""

                if idade >= 18 {
                    permissao = "Você é maior de idade"
                } else {
                   // executar caso falso:
                    permissao = "Você é menor de idade"
                }

                permissao // tem valor de "Você é menor
                de idade"
```


// If-else
...
if idade >= 18
true
false
permissao = "Você é maior de idade"
permissao = "Você é menor de idade"
permissao
...

```
// else if

idade = 16
var mensagem = ""

if idade >= 18 {
    mensagem = "você é um adulto"
} else if idade >= 12{
    mensagem = "você é um adolescente"
} else {
    mensagem = "você é uma criança"
}

mensagem // tem valor de "você é um adolescente"
```


// If-else
...
if idade >= 18
true
false
mensagem = "você é um adulto"
else if idade >= 12
true
false
mensagem = "você é um adolescente"
mensagem = "você é uma criança"
permissao
...

// switch

```
var animal = "Cachorro"
var som: String

switch animal {

case "Gato":
    som = "miado"

case "Cachorro":
    som = "latido"

case "Leão":
    som = "rugido"

default:
    // se for qualquer outro animal
    som = "indefinido"
}

som // tem valor de "latido"
```

// switch

```
animal = "Gata"
som = ""
switch animal {
case "Gato", "Gata":
   som = "miado"
case "Cachorro", "Cadela":
   som = "latido"
case "Leão", "Leoa":
   som = "rugido"
default:
  // se for qualquer outro animal
   som = "indefinido"
}
som // tem valor de "miado"
```

// Loops

```
// for in


        cidadesVisitadas = ["Recife", "Olinda", "Jaboatão"]
        mensagem = "nós visitamos:"

        for cidade in cidadesVisitadas {
            mensagem = mensagem + " " + cidade
        }

        mensagem // tem valor de "nós visitamos: Recife Olinda Jaboatão"
```

// for in

```
for cidade in cidadesVisitadas {
    mensagem = mensagem + " " + cidade
}
```

Execução	Valor de cidade	Valor de mensagem
Antes do loop	Não existe	nós visitamos:
Fim da 1a Iteração	Recife	nós visitamos: Recife
Fim da 2a Iteração	Olinda	nós visitamos: Recife Olinda
Fim da 3a Iteração	Jaboatão	nós visitamos: Recife Olinda Jaboatão
Após o loop	Não existe	nós visitamos: Recife Olinda Jaboatão

```
cidadesVisitadas = ["Recife", "Olinda", "Jaboatão"]
```

```swift
// for in range

var quadrados: [Int] = []

for numero in 1...10 {
    let quadrado = numero * numero
    quadrados.append(quadrado)
}

quadrados // tem valor de [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

// while *(enquanto)*

```
var preco = 100
var devoComprar = false

//enquanto o preço for caro não devo comprar
while !devoComprar {

    preco = preco - 1 // preço diminui

     //se o preco chegar em 50, eu devo comprar.
    if preco == 50 {
        devoComprar = true
    }
}

preco        // tem valor de 50
devoComprar // tem valor de true
```

// Funções e Closures

```swift
// funções - declaração

    func maior(primeiro: Int, segundo: Int) -> Int{
        if primeiro > segundo {
            return primeiro

        } else {
            return segundo
        }
    }
```

```swift
// funções - declaração

func maior(primeiro: Int, segundo: Int) -> Int{
    if primeiro > segundo {
        return primeiro
    } else {
        return segundo
    }
}
```

Palavra-reservada
(keyword)

```swift
// funções - declaração

func maior(primeiro: Int, segundo: Int) -> Int{
    if primeiro > segundo {
        return primeiro
    } else {
        return segundo
    }
}
```

Palavra-reservada ***(keyword)***

Identificador

```
// funções - declaração

    func maior(primeiro: Int, segundo: Int) -> Int{
        if primeiro > segundo {
            return primeiro
        } else {
            return segundo
        }
    }
```

Palavra-reservada *(keyword)*

Identificador

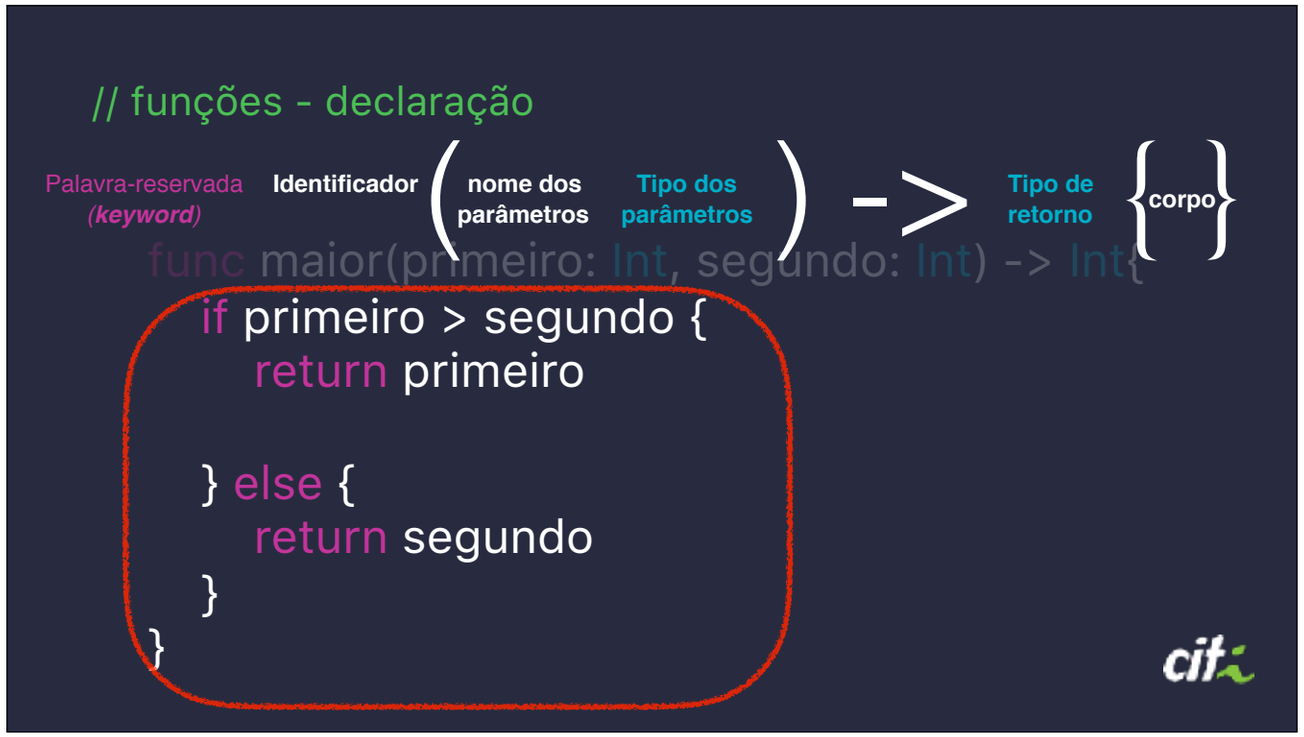
nome dos parâmetros

```
// funções - declaração

func maior(primeiro: Int, segundo: Int) -> Int{
    if primeiro > segundo {
        return primeiro
    } else {
        return segundo
    }
}
```

Palavra-reservada *(keyword)* | **Identificador** | **nome dos parâmetros** | **Tipo dos parâmetros**

```
// funções - declaração

func maior(primeiro: Int, segundo: Int) -> Int {
    if primeiro > segundo {
        return primeiro
    } else {
        return segundo
    }
}
```

Palavra-reservada *(keyword)* | Identificador | (nome dos parâmetros | Tipo dos parâmetros) -> Tipo de retorno


// funções - declaração
Palavra-reservada
(keyword)
Identificador
nome dos parâmetros
Tipo dos parâmetros
->
Tipo de retorno
corpo
func maior(primeiro: Int, segundo: Int) -> Int{
if primeiro > segundo {
return primeiro
} else {
return segundo
}
}

```swift
// funções - chamada

let um = 1
let dois = 2

let maximo = maior(primeiro: um, segundo: dois)

maximo // tem valor de 2
```

```swift
// funções - chamada

let um = 1
let dois = 2

let maximo = maior(primeiro: um, segundo: dois)

maximo // tendo valor de 2
```

Identificador

```
// funções - chamada

let um = 1
let dois = 2

let maximo = maior(primeiro: um, segundo: dois)

maximo // tem valor de 2
```

Identificador

nome dos parâmetros

```
// funções - chamada

let um = 1
let dois = 2

let maximo = maior(primeiro: um, segundo: dois)

maximo // tem valor de 2
```

Identificador (nome dos parâmetros argumentos)

// funções são tipos

O **tipo** de uma função é dado pelo tipo de seus **parâmetros** e de seu **retorno**, separados por uma seta.

```swift
// funções são tipos

func maior(primeiro: Int, segundo: Int) -> Int{...}
```

```swift
// funções são tipos

func maior(primeiro: Int, segundo: Int) -> Int{...}

                (primeiro: Int, segundo: Int) -> Int
```

```swift
// funções são tipos

func maior(primeiro: Int, segundo: Int) -> Int{...}

(primeiro: Int, segundo: Int) -> Int

(Int, Int) -> Int
```

// funções de primeira ordem

Funções que recebem como **parâmetro**, ou **retornam**, outras funções.

```swift
// funções de primeira ordem

var array = [1, 2, 3]

func somaUm(valor: Int) -> Int {
    return valor + 1
}

// map aplica a funcao somaUm a cada elemento de array
var novoArray = array.map(somaUm)

novoArray   // tem valor de [2, 3, 4]
```

```swift
// funções de primeira ordem


func somaUm(valor: Int) -> Int {
    return valor + 1
}


(Int) -> Int
```

```
// map


var array = [1, 2, 3]

var novoArray =  array.map(somaUm)
```

```
// map

var array = [1, 2, 3]

var novoArray = [ somaUm(valor: array[0]),   somaUm(valor: array[1]),   somaUm(valor: array[2]) ]
```

// closures

Funções **anônimas** que podem ser definidas no mesmo lugar em que são chamados

```swift
// map com closure

array = [1, 2, 3]
novoArray = []

novoArray = array.map({ (numero: Int) -> Int in
    return numero + 1
})

novoArray // tem valor de [2, 3, 4]
```

```swift
// closures

    novoArray = array.map({ (numero: Int) -> Int in
        return numero + 1
    })
```

{ }

```
// closures

    novoArray = array.map({ (numero: Int) -> Int in
        return numero + 1
    })
```

{ (nome dos parâmetros)

}

```
// closures

    novoArray = array.map({ (numero: Int) -> Int in
        return numero + 1
    })
```

{ (nome dos parâmetros Tipo dos parâmetros) }

```
// closures

    novoArray = array.map({ (numero: Int) -> Int in
        return numero + 1
    })
```

{ (nome dos parâmetros Tipo dos parâmetros) -> Tipo de retorno }

```
// closures

    novoArray = array.map({ (numero: Int) -> Int in
        return numero + 1
    })
```

{ (nome dos parâmetros Tipo dos parâmetros) -> Tipo de retorno Palavra-reservada *(keyword)* }

```
// closures

    novoArray = array.map({ (numero: Int) -> Int in
        return numero + 1
    })
```

{ (nome dos parâmetros | Tipo dos parâmetros) -> Tipo de retorno | Palavra-reservada *(keyword)* | corpo }

citi

```swift
// função vs closure

func somaUm(valor: Int) -> Int {
    return valor + 1
}

==

{ (numero: Int) -> Int in
    return numero + 1
})
```

```swift
// closure enxuto


novoArray = array.map({ (numero: Int) -> Int in
    return numero + 1
})
```

```swift
// closure enxuto


novoArray = array.map({ (numero: Int) -> Int in
    return numero + 1
})
```

```swift
// closure enxuto


novoArray = array.map({ numero in
    return numero + 1
})
```

```swift
// closure enxuto


  novoArray = array.map({ numero in
      return numero + 1
  })
```

```
// closure enxuto


  novoArray = array.map({
      return numero + 1
  })
```

```
// closure enxuto


 novoArray = array.map({
    return        $0 + 1
 })
```

```
// closure enxuto


 novoArray = array.map({
                    $0 + 1
 })
```

```
// closure enxuto


  novoArray = array.map({ $0 + 1 })
```

```
// closure enxuto


  novoArray = array.map { $0 + 1 }
```

```
// closure enxuto


                array = [1, 2, 3]
                novoArray = []

                novoArray = array.map {$0 + 1}

                novoArray // tem valor de [2, 3, 4]
```

```swift
// função vs closure II

func somaUm(valor: Int) -> Int {
    return valor + 1
}
                ==
{ (numero: Int) -> Int in
    return numero + 1
})
                ==
            {$0 + 1}
```

// Exercícios

// Exercício 01-a

Ano Bissexto

Chama-se ano bissexto o ano ao qual é acrescentado um dia extra, ficando ele com 366 dias.

Ocorrendo a cada **quatro anos** (exceto anos **múltiplos de 100** que não são **múltiplos de 400**).

Dado um ano como entrada, identifique se o ano é bissexto.

// Exercício 01-b

Ano Bissexto

Chama-se ano bissexto o ano ao qual é acrescentado um dia extra, ficando ele com 366 dias.

Ocorrendo a cada **quatro anos** (exceto anos **múltiplos de 100** que não são **múltiplos de 400**).

Calcule os anos bissextos de 1988 a 2017

// Exercício 01-c

Ano Bissexto

Chama-se ano bissexto o ano ao qual é acrescentado um dia extra, ficando ele com 366 dias.

Ocorrendo a cada **quatro anos** (exceto anos **múltiplos de 100** que não são **múltiplos de 400**).

Filtre um array de anos, deixando somente os bissextos