

FIAP GRADUAÇÃO

# ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Desenvolvimento Avançado iOS

PROF. GUSTAVO CALIXTO

# Introdução ao Swift

# INTRODUÇÃO AO SWIFT

- Aula de Hoje
  - Entendimento geral sobre classes, enumerações, protocolos, extensões, tratamento de erro e generics.
- Git com códigos de exemplo
  - <https://github.com/gmcalixto/ios>

# INTRODUÇÃO AO SWIFT

A classe segue um padrão de implementação com os seguintes membros

- atributos
- propriedades
- funções (métodos)

```
class Poligono{  
  
    //atributos  
    var lados:Int = 0  
  
    //construtor  
    init(lados: Int){  
        self.lados = lados  
    }  
  
    //metodos  
    func getDescription() -> String {  
        return "Este poligono tem \(self.lados) lados"  
    }  
  
}  
  
//instancia do objeto  
var triangulo = Poligono(lados: 3)  
print(triangulo.getDescription())
```

"class" é a palavra reservada para classes

Atributo com valor inicial

Palavra reservada "init" para definir o construtor  
Uso da palavra self para fazer auto referência aos  
membros da classe. Sem necessidade de retornar  
uma cópia do objeto como no Objective-C

Binding não necessita de palavra reservada,  
tal como new ou alloc, por exemplo.

# INTRODUÇÃO AO SWIFT

Herança - Classe base

```
class Shape{  
  
    var numberOfSides:Int = 0  
    var sideSize:Double = 0  
    var name:String? = nil  
  
    //uso de propriedades  
    var perimeter:Double{  
        get{  
            return 0;  
        }  
    }  
  
    init(sides: Int, size: Double, name: String){  
        self.numberOfSides = sides  
        self.sideSize = size  
        self.name = name  
    }  
  
    func getArea() -> Double{  
        return 0;  
    }  
}
```

Não há classe abstrata em Swift

Uso de propriedade, definido pré-processamento no uso do get.

# INTRODUÇÃO AO SWIFT

## Herança - Classe derivada

```
class Triangle : Shape{  
  
    //uso da palavra super para acessar membros da classe base  
    init(sizeT: Double, nameT: String) {  
        super.init(sides: 3, size: sizeT, name: nameT)  
    }  
  
    //veja que propriedades podem ser reescritas e formar  
    //polimorfismo  
    //newValue: palavra reservada para valor atribuído no set  
    override var perimeter:Double{  
        get{  
            return super.sideSize * 3  
        }  
        set{  
            super.sideSize = newValue/3  
        }  
    }  
  
    //reescrita da função  
    override func getArea() -> Double {  
        return (pow(sideSize,2) * sqrt(3))/4  
    }  
}
```

Classe derivada : Classe base

"super" acessa os elementos da classe base

"override" indica a re-escrita do método

newValue é uma palavra reservada para indicar o valor atribuído para a propriedade

# INTRODUÇÃO AO SWIFT

Herança - Geração dos objetos

```
var trig1 = Triangle(sizeT: 10, nameT: "trig1")
trig1.perimeter = 27
print("Lado: \(trig1.sideSize)")
print("Perímetro: \(trig1.perimeter)")
print("Área: \(trig1.getArea())")
```

# INTRODUÇÃO AO SWIFT

## Enumerações

```
enum Mes{
    case Janeiro, Fevereiro, Marco, Abril, Maio
    case Junho, Julho, Agosto, Setembro, Outubro
    case Novembro, Dezembro

    // veja que é possível definir funções para enumerações
    // o próprio valor de uma variável pode ser usado com a
    // palavra "self"
    func getNumeroDias(ano: Int) -> Int{
        switch self{
            case .Janeiro, .Marco, .Maio, .Julho, .Agosto,
                Outubro, .Dezembro:
                return 31
            case .Abril, .Junho, .Setembro, .Novembro:
                return 30
            case .Fevereiro:
                if (ano % 4) == 0{
                    return 29
                }
                else{
                    return 28
                }
        }
    }
}
```

Testa elemento  
que está  
atribuído na  
variável enum.

"enum" é a palavra reservada para enumerações  
Deve ser vinculado o tipo de dados

Elementos da enumeração são  
criados usando a palavra  
reservada "case"

A enumeração no Swift permite a  
implementação de funções internas

# INTRODUÇÃO AO SWIFT

Enumerações – Uso como variável

"enum" é a palavra reservada para enumerações  
Deve ser vinculado o tipo de dados

```
var mesAniversario = Mes.Fevereiro
var mesCasamento = Mes.Dezembro
```

```
print("O mes de aniversario tem \$(mesAniversario.getNumeroDias
      (ano: 2016))")
print("O mes de casamento tem \$(mesCasamento.getNumeroDias(ano:
      2016))")
```

# INTRODUÇÃO AO SWIFT

Enumerações com elementos parametrizáveis

```
enum Mensagem{  
    case Sucesso(String, String)  
    case Falha(String, String)  
  
    func getStatus() -> String{  
        switch self{  
            case let .Falha(msg, time):  
                return "Falha: \(msg), \(time)"  
            case let .Sucesso(msg, time):  
                return "Sucesso: \(msg), \(time)"  
        }  
    }  
}
```

```
var msg1 = Mensagem.Sucesso("Mensagem Entregue", "8:00AM")  
var msg2 = Mensagem.Falha("Erro", "9:30PM")
```

```
print(msg1.getStatus())  
print(msg2.getStatus())
```

Elementos podem portar tupla de dados

Uso do let para trabalhar elementos parametrizáveis como objetos

Sempre no uso dos elementos da enumeração o fornecimento dos dados da tupla são obrigatórios.

# INTRODUÇÃO AO SWIFT

Protocolo - Obriga a implementação de propriedades em uma classe ou estrutura

"protocol" é a palavra reservada para protocolos

```
protocol Imovel{  
    var areaTotal:Float {get set}  
    var valorVenal:Float {get set}  
    var ano:Int {get set}  
    mutating func getTipo() -> String  
}
```

Pode ser definido que a propriedade seja somente leitura {get} ou leitura e escrita {get set}

A palavra mutating permite uma assinatura abstrata da função, obrigando a implementação na classe ou estrutura a qual a herdar.

# INTRODUÇÃO AO SWIFT

Protocolo - Obriga a implementação de propriedades em uma classe ou estrutura

"protocol" é a palavra reservada para protocolos

```
protocol Imovel{  
    var areaTotal:Float {get set}  
    var valorVenal:Float {get set}  
    var ano:Int {get set}  
    mutating func getTipo() -> String  
}
```

Pode ser definido que a propriedade seja somente leitura {get} ou leitura e escrita {get set}

A palavra mutating permite uma assinatura abstrata da função, obrigando a implementação na classe ou estrutura a qual a herdar.

# INTRODUÇÃO AO SWIFT

Protocolo - Obriga a implementação de propriedades em uma classe ou estrutura

Mesma sintaxe utilizada no caso de herança

```
class Apartamento: Imovel{
    var areaTotal:Float
    var valorVenal: Float
    var ano: Int

    func getTipo() -> String {
        return "Apartamento"
    }

    init(){
        self.ano = 0
        self.valorVenal = 0
        self.areaTotal = 0
    }
}
```

# INTRODUÇÃO AO SWIFT

## Protocolo - Uso de objetos

```
var imovel1 = Apartamento()  
  
imovel1.ano = 2009  
imovel1.areaTotal = 60  
imovel1.valorVenal = 102346.44  
  
//uso da função da classe NS String para formatar números em  
// ponto flutuante  
let twoDecimalPlaces = NSString(format: "%.2f", imovel1.  
    valorVenal)  
  
print("\(imovel1.getTipo()), Ano: \(imovel1.ano), Area: \  
    (imovel1.areaTotal), Valor: \(twoDecimalPlaces)")  
  
var imovel2 = imovel1  
  
imovel2.areaTotal = 90  
  
print("\(imovel1.getTipo()), Ano: \(imovel1.ano), Area: \  
    (imovel1.areaTotal), Valor: \(twoDecimalPlaces)")
```

Uso da classe NSString e do construtor "format" para formar a saída de números em ponto flutuante

Variáveis imovel1 e imovel2 possuem a mesma referência  
tratando-se de instância de objetos

# INTRODUÇÃO AO SWIFT

Struct – Estrutura que gera binding estático

"struct" é a palavra reservada para estruturas, podendo herdar protocolos e outras estruturas.

```
struct Casa : Imovel{
    var areaTotal:Float = 0
    var valorVenal: Float = 0
    var ano: Int = 0

    func getTipo() -> String {
        return "Casa"
    }
}
```

# INTRODUÇÃO AO SWIFT

Struct – Uso da estrutura

```
var imovel3 = Casa()  
imovel3.ano = 2010  
imovel3.areaTotal = 100  
imovel3.valorVenal = 202346.44  
  
//teste para ver a diferença entre class e struct  
print("\(imovel3.getTipo()), Ano: \(imovel3.ano), Area: \  
      (imovel3.areaTotal), Valor: \($0.00)")  
  
var imovel4 = imovel3  
imovel4.ano = 2011  
  
print("\(imovel3.getTipo()), Ano: \(imovel3.ano), Area: \  
      (imovel3.areaTotal), Valor: \($0.00)")
```

Perceba que a atribuição de imovel3 para imovel4, gera uma cópia  
da estrutura, característico de atribuições em tipos primitivos.

# INTRODUÇÃO AO SWIFT

Extensão - Adicionar membros a uma classe componentizado

```
protocol Descricao{  
    mutating func getDescricao() -> String  
}  
  
//a extensão é utilizada quando se deseja ampliar os membros em  
uma classe já componentizada.  
extension Int:Descricao{  
    func getDescricao() -> String {  
        return "Sou o numero \(self)"  
    }  
}  
  
let numero = 10  
  
print(10.getDescricao())
```

Uma extensão pode usar um protocolo como modelo

Uso do objeto do tipo Int com a extensão.

# INTRODUÇÃO AO SWIFT

Captura de erros – Enumeração com protocolo Error

```
enum TransactionError: Error{  
    case NoFunds  
    case NoDevice  
}
```

Toda a enumeração com os tipos de erro deve estar acompanhada do protocolo “Error”

Cada caso de erro deve ser descrito com a palavra reservada “case”

# INTRODUÇÃO AO SWIFT

Captura de erros – Uso das enumerações de erro em funções

```
func doTransaction(deviceName: String?, funds: Float) throws ->
String{
    if deviceName == nil{
        throw TransactionError.NoDevice
    }
    else if funds <= 0{
        throw TransactionError.NoFunds
    }

    return "done"
}
```

Para retornar o erro, basta usar a palavra  
"throw" seguindo do elemento de  
enumeração a ser retornado.

Palavra "throws" indica que  
a função pode retornar um  
erro

# INTRODUÇÃO AO SWIFT

Captura de erros – Uso das enumerações de erro em funções

```
let device:String? = nil  
let _funds:Float = 0.1  
  
do{  
    print(try doTransaction(deviceName: device, funds: _funds))  
}  
catch TransactionError.NoFunds {  
    print("Sem fundos!")  
}  
catch TransactionError.NoDevice{  
    print("Sem dispositivo!")  
}
```

Bloco "do" indicar que o trecho de código permite tratamento de erro

"try" antes da execução da função indica que a mesma pode gerar um erro

"catch" apresenta o evento a ser executado para cada tipo de erro gerado.

# INTRODUÇÃO AO SWIFT

Generics – Permite o uso de argumentos de função ou retorno de qualquer tipo de dado.

```
func repetePalavra<Palavra> (palavra: Palavra, vezes: Int) ->
    [Palavra]{
    var resposta = [Palavra]()
    for _ in 0..<vezes{
        resposta.append(palavra)
    }
    return resposta
}

print(repetePalavra(palavra: "Toc", vezes: 10))
print(repetePalavra(palavra: 10, vezes: 10))
```

Palavra genérica é declarada depois do nome da função entre "<>"

Permite adaptar qualquer tipo de dado

# INTRODUÇÃO AO SWIFT

## Exercícios

- Elabore o seguintes mecanismos
  - Reajuste dos preços de uma lista de Itens disponíveis em um array (classe) (nome do produto, preço, tipo do produto).
    - O reajuste deve ser feito por um índice de aumento (maior que 0,0). Índices menores que 1,0 reduzem o preço. Índices menores que 1,0 aumentam o preço.
    - O reajuste pode ser feito por tipo de produto.
    - Prever erros (montar uma enumeração com Error) quando a lista de itens está vazia ou o índice de correção é menor que 0,0.
  - Consulta o preço de um item em específico.
  - Lista os produtos por tipo
  - Lista os produtos em uma faixa de preço definida

# INTRODUÇÃO AO SWIFT

## Próxima aula

- Experiência inicial com o iOS
  - ViewController
  - StoryBoard
  - Actions, Outlets e Segues
  - Simulador iOS

Copyright © 2019 Prof. Gustavo Moreira Calixto

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).