

CURSO DE RUBY PURO

OneBitCode

[INTRODUÇÃO DO CURSO 6](#_Toc19268839)

[Funcionamento do Curso 6](#_Toc19268840)

[Aulas do curso: 6](#_Toc19268841)

[INTRODUÇÃO AO RUBY 8](#_Toc19268842)

[História e características 8](#_Toc19268843)

[Por que aprender Ruby? 8](#_Toc19268844)

[Instalação 8](#_Toc19268845)

[Rbenv - RVM 8](#_Toc19268846)

[Codeanywhere 8](#_Toc19268847)

[Primeiro Hello World 8](#_Toc19268848)

[IRB 8](#_Toc19268849)

[Missões especiais 01 9](#_Toc19268850)

[CONCEITOS BÁSICOS 10](#_Toc19268851)

[Tipos de dados 10](#_Toc19268852)

[O que são 10](#_Toc19268853)

[Integer 10](#_Toc19268854)

[Float 10](#_Toc19268855)

[Boolean 10](#_Toc19268856)

[String 10](#_Toc19268857)

[Array 10](#_Toc19268858)

[Symbol 10](#_Toc19268860)

[Hash 11](#_Toc19268862)

[Tipagem dinâmica 11](#_Toc19268864)

[Operadores Matemáticos 11](#_Toc19268865)

[Entrada/Saída 11](#_Toc19268866)

[Print: 11](file:///C:\Users\Felipe%20Avelar\Studies\Ruby\rubyPuro\ApostilaRubyPuro.docx#_Toc19268867)

[Convertendo para inteiro 11](#_Toc19268868)

[Missões especiais 02 11](#_Toc19268869)

[Estruturas de Controle 12](#_Toc19268870)

[Condicionais 12](#_Toc19268871)

[If 12](#_Toc19268872)

[Else 12](#_Toc19268873)

[Elsif 12](#_Toc19268874)

[Unless 12](#_Toc19268875)

[Case 13](#_Toc19268876)

[Iteração 13](#_Toc19268877)

[For 13](#_Toc19268878)

[Times 13](#_Toc19268879)

[While 13](#_Toc19268880)

[Loop (Do While) 14](#_Toc19268881)

[Missões especiais 03 14](#_Toc19268882)

[Collections 15](#_Toc19268883)

[O que são? 15](#_Toc19268884)

[Array 15](#_Toc19268885)

[Criando array 15](#_Toc19268886)

[Inserindo dados no final do array 15](#_Toc19268887)

[Inserindo dados no início do Array 15](#_Toc19268888)

[Acessando dados no Array 15](#_Toc19268889)

[Tamanho do Array 15](#_Toc19268890)

[Verificando se é/está vazio 15](#_Toc19268891)

[Verificando se tem um item 16](#_Toc19268892)

[Removendo itens do Array 16](#_Toc19268893)

[Hash 16](#_Toc19268894)

[Criando um Hash 16](#_Toc19268895)

[Definindo valores em um Hash 16](#_Toc19268896)

[Adicionando novos dados no Hash 16](#_Toc19268897)

[Pegando todas a chaves do Hash 16](#_Toc19268898)

[Acessando dados do Hash 16](#_Toc19268899)

[Tamanho do Hash 16](#_Toc19268900)

[Verificando se é vazio 17](#_Toc19268901)

[Removendo um valor do Hash 17](#_Toc19268902)

[Iterações 17](#_Toc19268903)

[Each 17](#_Toc19268904)

[Map 17](#_Toc19268905)

[Select 17](#_Toc19268906)

[Missões especiais 04 18](#_Toc19268907)

[Métodos 19](#_Toc19268908)

[O que são? 19](#_Toc19268909)

[Gems 19](#_Toc19268910)

[Instalando uma gem 19](#_Toc19268911)

[Removendo uma Gem 19](#_Toc19268912)

[Listando Gems 19](#_Toc19268913)

[Bundler 19](#_Toc19268914)

[Instalando 19](#_Toc19268915)

[Criando 19](#_Toc19268916)

[Executando 19](#_Toc19268917)

[Missões especiais 05 19](#_Toc19268918)

[Programação Orientado a Objetos 20](#_Toc19268919)

[O que é 20](#_Toc19268920)

[Pilares da POO (3 ou 4) 20](#_Toc19268921)

[Abstração 20](#_Toc19268922)

[Encapsulamento 20](#_Toc19268923)

[Herança 20](#_Toc19268924)

[Polimorfismo 20](#_Toc19268925)

[Classe 20](#_Toc19268926)

[Criando uma classe: 20](#_Toc19268927)

[Objeto 20](#_Toc19268928)

[Exemplo de classes: 21](#_Toc19268929)

[Missões especiais 06 21](#_Toc19268930)

[POO – Parte 2 22](#_Toc19268931)

[Require 22](#_Toc19268932)

[Escopo das variáveis 22](#_Toc19268933)

[Variável Local 22](#_Toc19268934)

[Variável Global 22](#_Toc19268935)

[Variável de Classe 22](#_Toc19268936)

[Variável de Instancia 22](#_Toc19268937)

[Atributos 22](#_Toc19268938)

[attr\_accessor 22](#_Toc19268939)

[Construtores 22](#_Toc19268940)

[Missões especiais 07 23](#_Toc19268941)

[Ruby Avançado I 24](#_Toc19268942)

[Blocks 24](#_Toc19268943)

[Lambda 24](#_Toc19268944)

[Modules 25](#_Toc19268945)

[Namespace 25](#_Toc19268946)

[Mixins 26](#_Toc19268947)

[Missões especiais 08 27](#_Toc19268948)

INTRODUÇÃO DO CURSO

Funcionamento do Curso

* Aulas teóricas e práticas;
* Missões especiais (tarefas) no final de cada aula;
* Projeto de conclusão ao final do curso;

Aulas do curso:

* Introdução ao Ruby
  + História e Características;
  + Instalação;
  + Seu primeiro “Hello World!”;
  + Ruby Irb;
* Conceitos Básicos
  + Tipos de dados;
  + Operadores matemáticos;
  + Strings;
  + Entrada/Saída;
* Estrutura de Controle
  + If/else/elif;
  + Unless;
  + Case;
  + Iteração;
  + For;
  + While;
  + Times;
  + Do while;
* Collections
  + Arrays;
  + Hashes;
  + Each;
  + Map;
  + Select;
* Métodos e Gems
  + Como criar métodos;
  + O que são e como usar Gems;
* POO 1 (Programação Orientada a Objetos)
  + Classes – pt1;
* POO 2
  + Classes – pt2;
  + Construtor
* Ruby Avançado
  + Blocks;
  + Lambda;
  + Namespacing;
  + Modules;
* Ruby Avançado II
  + Regex;
  + Math and time;
  + Methodos misssing;
  + Procs;
* Projeto final
  + Leitura de arquivos;
  + Chamadas web;
  + Web scraping;
  + Projeto final.

INTRODUÇÃO AO RUBY

História e características

* Criado na década de 90 pelo Japonês Yukihiro Matz Matsumoto;
* Lançado ao publico em 1995;
* Orientado a objetos;
* Tipagem dinâmica e forte;
* Interpretada;
* Expansão ligada a criação do Ruby on Rails;

Por que aprender Ruby?

* É uma linguagem produtiva e divertida;
* É fácil aprender;
* Tem uma grande comunidade;
* É usada por grandes empresas;
* Está presente em várias áreas;
* Top 10 nos índices de popularidades;

Instalação

Rbenv - RVM

Ferramenta para gerenciar várias versões do Ruby.

* + - * 1. <https://gorails.com/setup>;
        2. Sistema Operacional desejado;

Subsidiando o Linux no Windows – (<https://olhardigital.com.br/dicas_e_tutoriais/noticia/como-instalar-o-ubuntu-e-outras-distribuicoes-linux-no-windows-10/71845>).

Codeanywhere

Site onde pode usar o Ruby online. Também existe outros sites.

Primeiro Hello World

puts “Hello World”;

p “Hello World”;

puts(“Hello World”);

IRB

Irb é um Ruby executável dentro do próprio terminal, onde não se cria nenhum arquivo. Porém com isso, acaba sendo bem limitado.

Missões especiais 01

1. Instalar o Ruby na sua máquina ou preparar o Codeanywhere.
2. Exiba no console a seguinte mensagem: “Curso de Ruby do ONEBITCODE”.
3. Leia o seguinte artigo: Ruby (linguagem de programação) – Wikipédia, A enciclopédia livre. ( <https://pt.wikipedia.org/wiki/Ruby_(linguagem_de_programa%C3%A7%C3%A3o)> ).

CONCEITOS BÁSICOS

Tipos de dados

O que são

É o conceito que usamos para classificar dados, possibilitando a definição de regras para cada tipo. Com eles o Ruby sabe como lidar com os dados de nosso programa. Para sabermos o tipo usamos “.class” exemplo: var.class;

String

var = “Hello World”

var.class

Integer

Como na matemática, inteiro é o tipo de dado que representa o conjunto de números positivos, negativos e 0.

var = 10;

Float

Tipo que representa os números reais inexatos. De forma abreviada, são números decimais, números que contem ponto flutuante.

var = 3.5;

Boolean

Tipo de dado usado para informar a veracidade de algo. Possui apenas dois estados, sendo eles true que é uma instância da TrueClass e false que é uma instância da classe FalseClass.

var = true;

var = false;

String

Tipo que representa um texto. Um conjunto de letras, símbolos ou números. Pode ser definido de várias formas, porém, as mais comuns são dentro de aspas simples ou duplas.

var = “Hello World”

Array

Um tipo que nos permite armazenar uma lista ordenada de dados em único objeto. Para definir um array você deve utilizar colchetes. ([ ]).

var = [“Felipe”,”Mariangela”,”Carlos”,”Mauricio”];

var[1]

Mariangela

Symbol

Símbolo é um tipo de dados semelhantes a String, com a diferença de que ele é um objeto imutável. Duas strings idênticas podem ser objetos diferentes, mas um símbolo é apenas um objeto, ocupado sempre o mesmo espaço na memória. Um símbolo sempre é definido começando com dois pontos : seguido pelo seu nome

sim = :”Hello”;

sim.object\_id;

00000000 (id do objeto na memoria)

Hash

Tipo que representa uma coleção de dados organizados por chaves únicas e seus respectivos valores. Enquanto Arrays são definidos com colchetes. Hash são definidos com chaves ({ }).

var = {curso :”ADS”, língua :”PT-BR”, duracao\_Semestres = :”6”, local = :”Estacio”};

var[:curso];

ADS

Tipagem dinâmica

No Ruby não é preciso definir o tipo de dado antes de atribuir um valor à uma variável. O tipo é dinâmico, ou seja, ele é definido de acordo com o dado atribuído.

Operadores Matemáticos

* + ► Adição (10+10 = 20)
* - ► Subtração (20-10 = 10)
* \* ► Multiplicação (2\*5 = 10)
* / ► Divisão (20/4 = 5)
* % ► Modulo (9%4 = 1)
* \*\* ► Expoente (5\*\*2 = 25)

Entrada/Saída

Essas duas operações manipulam dado.

Com a diferença que a entrada ocorre quando o programa lê dados que podem ter sido recebidos de um teclado, de um arquivo, ou então de outro programa.

Já a saída é um dado produzido pelo programa que pode ser exibido em uma tela, enviado para um arquivo ou então para outro programa.

Print:

Uma diferença do print é que ele não pula linha automática como o puts.

#Output(Saida)

print (“Digite seu nome”);

#Input(Entrada)

name = gets.chomp;

#Saida usando puts

puts (“Hello #{name}!”);

Convertendo para inteiro

gets.chomp.to\_i;

Missões especiais 02

1. No IRB, crie todos os tipos especiais de dados mencionados na aula com valores diferentes como exemplo.
2. Crie um programa que receba o nome e idade de uma pessoa e no final exiba estes dois dados em uma única frase.
3. Crie um programa que receba dois números inteiros e no final exiba a soma, a subtração, a multiplicação, e a divisão entre eles.

Estruturas de Controle

Códigos que escrevemos em nossos programas para analisar dados e decidir algo a ocorrer. Divide-se em dois tipos:

* Condicional.
* Iteração.

Condicionais

Estrutura que executa um trecho do código dependendo do resultado da condição.

If

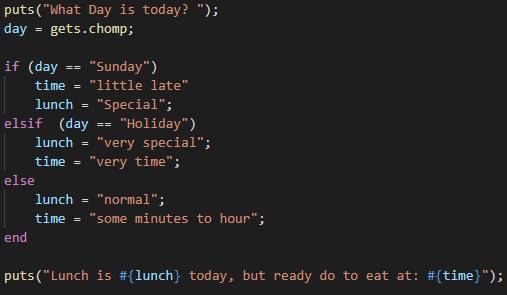
Verifica se uma condição é verdadeira (true), e a partir desse resultado determina se as instruções dentro de seu escopo serão ou não executadas.

Else

Informa o que fazer quando a verificação do if for falsa.

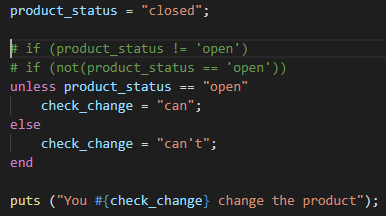
Elsif

Utilizado quando há necessidade de verificar mais de uma condição em um if.



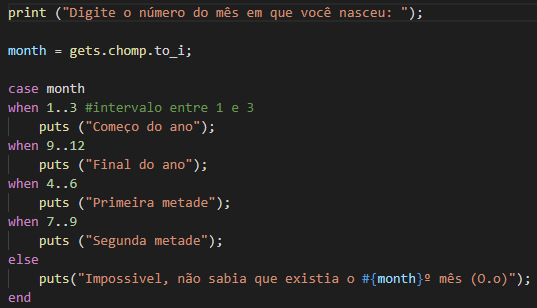
Unless

Enquanto o if é executado quando a condição for verdadeira, o unless ocorre de forma contrária. É executado apenas quando é falsa.



Case

Instrução muito parecida com o if. Ele se enquadra muito bem a situações com diversas condições.

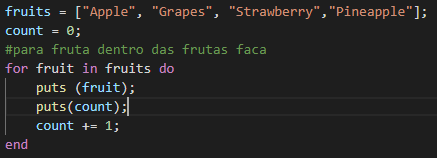


Iteração

Estrutura de controle que gerencia quantas vezes um trecho de código será executado.

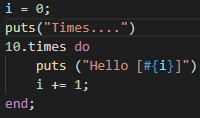
For

Usado para percorrer uma coleção de elementos.

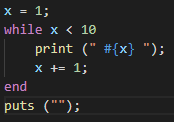


Times

Executa um trecho de código, por um determinado número de vezes.

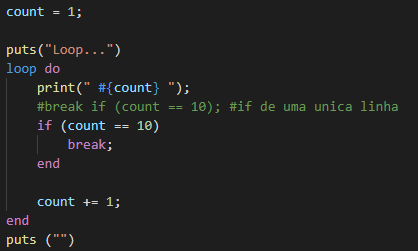


While

Instrução que repete o bloco de código enquanto sua condição for verdadeira.

Loop (Do While)

Cria um laço de repetição que só é parado quando instrução break for verdadeira.



Missões especiais 03

Utilizando as estruturas de iteração e de condição, crie uma calculadora que ofereça ao usuário a opção de multiplicar, dividir, adicionar ou subtrair números. Não se esqueça de também permitir que usuário feche o programa.

Collections

O que são?

Na programação, collection representa um conjunto de dados semelhantes em uma única unidade.

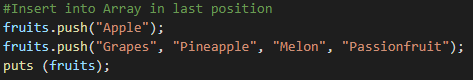
Fora: Um conjunto de livros agrupados, uns conjuntos numéricos, um conjunto de espécies, etc...

Array

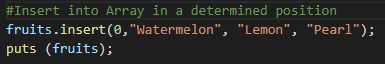
Criando array



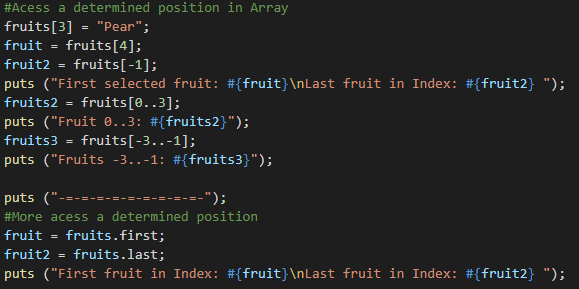
Inserindo dados no final do array



Inserindo dados no início do Array



Acessando dados no Array



Tamanho do Array



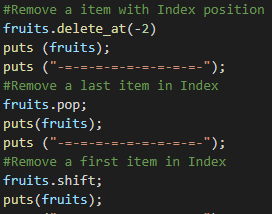
Verificando se é/está vazio



Verificando se tem um item



Removendo itens do Array



Hash

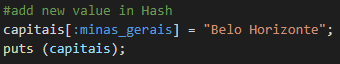
Criando um Hash



Definindo valores em um Hash



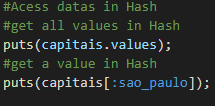
Adicionando novos dados no Hash



Pegando todas a chaves do Hash



Acessando dados do Hash



Tamanho do Hash



Verificando se é vazio



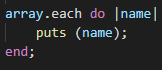
Removendo um valor do Hash



Iterações

Each

Percorre uma coleção de forma parecida ao For, porém, não sobrescreve o valor de variáveis fora da estrutura de repetição.

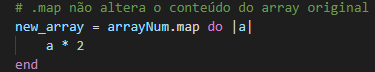




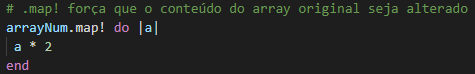
Map

Cria um Array baseando-se em valores de um Array já existente.

De maneira que mantém o conteúdo original intacto.

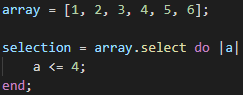


De maneira que alterar os valores do conteúdo original.



Select

Realiza uma seleção de elementos presentes em uma collection através de uma condição pré-definida. Traz como resultado somente os valores que passam nesta condição.





Missões especiais 04

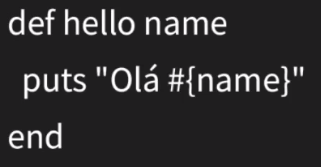
1. Utilizando uma collection do tipo Array, escreva um programa que receba 3 números e no final exiba o resultado de cada um deles elevado a segunda potência.
2. Crie uma collection do tipo Hash e permita que o usuário crie três elementos informando a chave e o valor. No final do programa para cada um desses elementos imprima a frase “Uma das chaves é \*\*\*\* e o seu valor é \*\*\*\*”
3. Dado o seguinte hash: Numbers = {a: 10, b: 30, c: 20, d: 25, e: 15}

Crie uma instrução que seleciona o maior valor deste hash e no final imprima a chave e valor do elemento resultante

Métodos

O que são?

São uma forma organizada de instruções em um programa, permitindo que trechos de códigos sejam reutilizáveis.



Gems

Gem é um pacote que oferece funcionalidade a fim de resolver funcionalidades especificas de um programa Ruby.

Pense como o conceito de bibliotecas em outras linguagens de programação.

Instalando uma gem

Gem Install name

Gem Install os

Removendo uma Gem

Gem uninstall name

Gem uninstall os

Listando Gems

Gem list

Bundler

Agrega varias gems em um arquivo e depois instala todas elas.

Instalando

Gem install bundler

Criando

Criar arquivo chamado GemFile

Executando

Abrir pasta do GemFile, executar o seguinte comando:

bundle

Missões especiais 05

1. Crie um programa que possua um método que resolva a potência dado um número base e seu expoente. Esses dois valores devem ser informados pelo usuário.
2. Siga a documentação da gem CPF\_CNPJ para criar um programa que recebe como entrada um numero de CPF e em um método verifique se esse número é válido.

Programação Orientado a Objetos

O que é

É um paradigma de programação criado para lidar com software grandes e complexos. É um conceito seguido não só pelo Ruby, mas varias outras linguagens como: Python, Java, C#, etc.

Pilares da POO (3 ou 4)

Abstração

Ação de abstrair uma entidade do mundo real e transformá-la em uma classe, também visa trazer o que for necessário do objeto real

Encapsulamento

É o ato de dividir um programa em diversas partes tornando-o flexível, fácil de modificar e incluir novas funcionalidade. Também é responsável por garantir o funcionamento do programa.

Herança

Habilidade de criar uma classe com características de outra existente. A herança promove o reuso e reaproveitamento do código.

Classe < SuperClasse (classe extends SuperClasse – Java/C#)

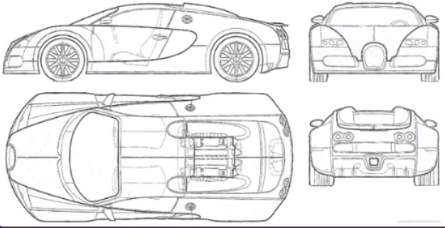
Polimorfismo

Capacidade de utilizar um método de diferentes formas para diferentes objetos.

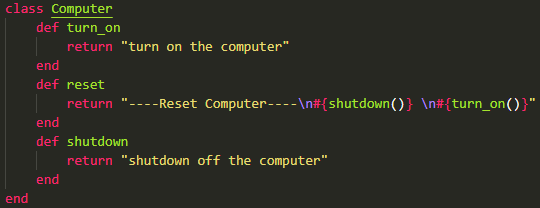
Classe

Tendo como exemplo o objeto carro, pense em uma classe como a “planta” desse carro. Com ela é possível construir vários carros.

Uma classe é como a planta de um objeto



Criando uma classe:

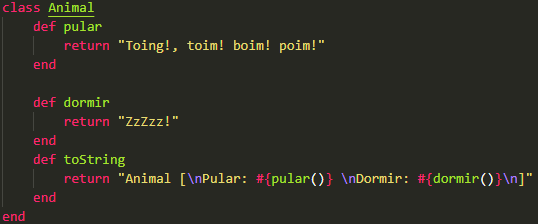


Objeto

Por definição, o objeto é toda coisa material que pode ser percebida pelos sentidos, como por exemplo um carro, livro, caneta, avião, etc. Na programação um objeto é representação de um objeto do mundo real.

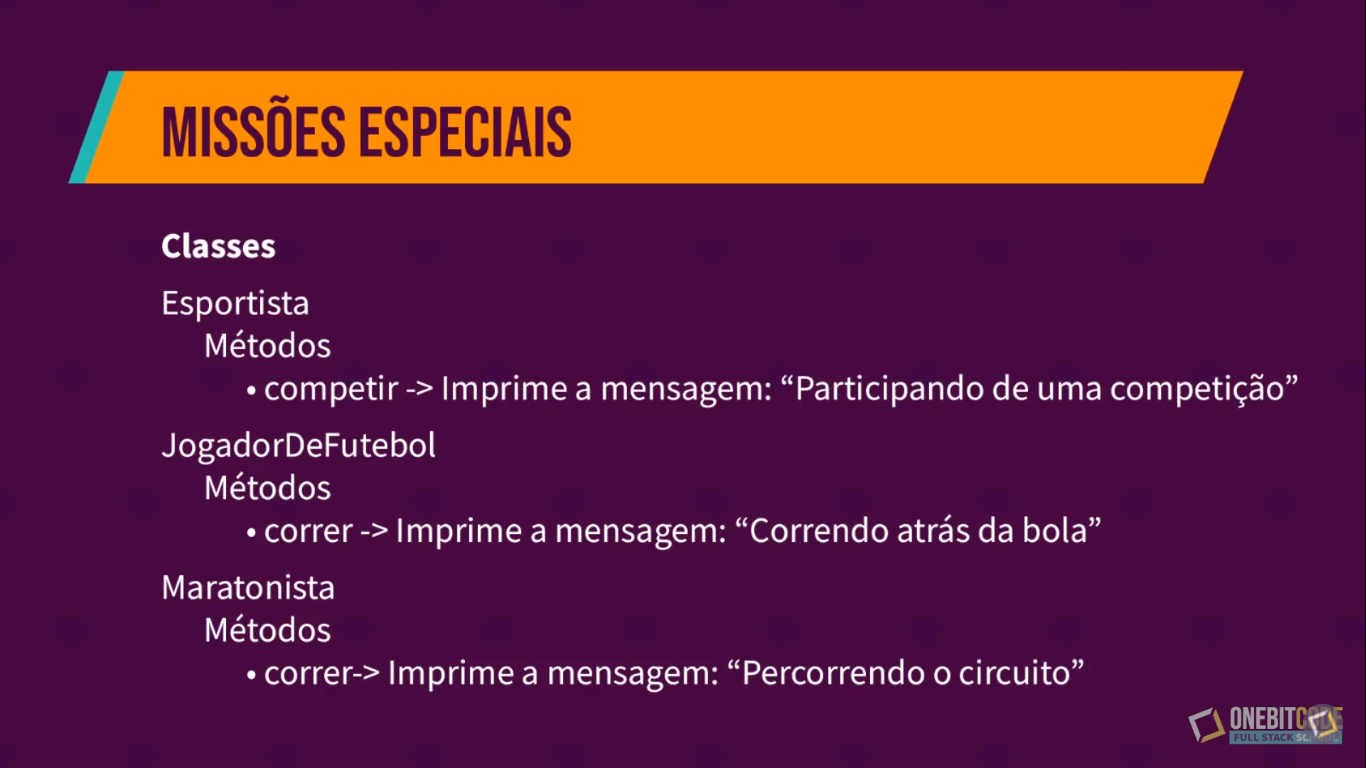
Todos esses objetos apresentam duas características em comum: informações (atributos) e comportamentos (métodos).

Exemplo de classes:





Missões especiais 06



As classes JogadorDeFutebol e Maratonista devem herdar os comportamentos de esportista.

No final do programa execute os métodos competir e correr em objetos do tipo JogadorDeFutebol e Maratonista

POO – Parte 2

Require

Na aula05 (Gems), já o utilizamos para carregar as gems.

Com ele também é possível arquivos Ruby que foram escritos por você. Para exemplifica como isso acontece, você criara um projeto onde alguns arquivos irão conter apenas uma classe.

Escopo das variáveis

O escopo define onde a variável vai estar disponível dentro do programa

No Ruby existem 4 tipos:

Variável Local

É declarada com a primeira letra do seu nome sendo uma letra minúscula ou \_. Pode ser acessada apenas onde foi criada

Por exemplo, se você definir ela dentro de uma classe, ela estará disponível apenas dentro desta classe, se a definiu dentro de um método, conseguirá acessá-la apenas no método.

Variável Global

Declarada com prefixo de $. Pode ser acessada em qualquer lugar do programa. Seu uso é fortemente desencorajado pois além de ser visível em qualquer lugar do código, também pode ser alterada em inúmeros locais ocasionando em dificuldades em rastreamento de bugs

Variável de Classe

Declarada com prefixo de @@. Pode ser acessada em qualquer lugar da classe onde foi declarada e seu valor é compartilhado entre todas as instâncias de sua classe.

Variável de Instancia

Seu nome começa com @. Semelhante a variável de classe, porém o valor não é compartilhado entre todas as instâncias de sua classe.

Atributos

Atributos (variáveis de instância) no Ruby são sempre privados e começam com @, eles só podem ser alterados pelos métodos de um objeto.

attr\_accessor

Cria os gets e sets automaticamente.

attr\_accessor :name, :age

dog = *Dog*.new

dog.name = 'Marlon'

dog.age = 3

puts (dog.name)

puts ("Tem: #{dog.age} ano(s)")

Construtores

Toda vez que a instância de uma classe é criada, o Ruby procura por um método chamado initialize.

Você pode criar esse método para especificar valores padrões durante a construção do objeto.

class Person

    attr\_accessor :name, :age

    def initialize (*name*, *age*)

        @name = name

        @age = age

    end

    def toString

        return "Person[\nName: #{@name}\nAge: #{@age}\n]"

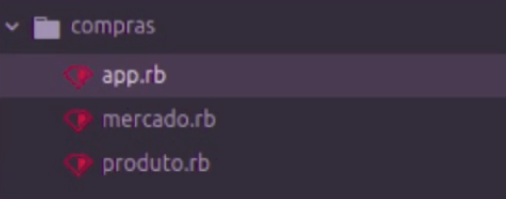
    end

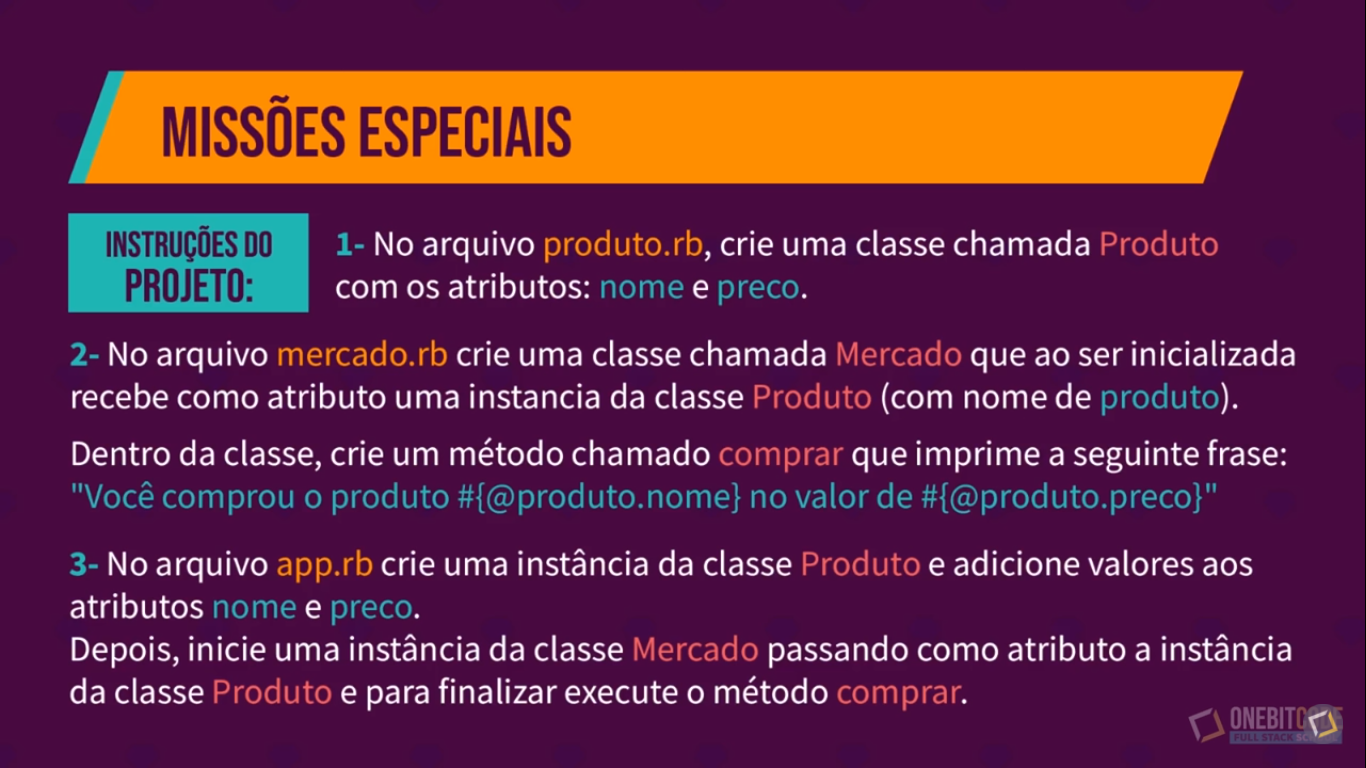
end

Missões especiais 07

Crie um projeto chamado compras com a seguinte estrutura:

Nesse projeto, você simulará o ato de escolher e comprar o produto em um mercado.





Ruby Avançado I

Blocks

Pode ser entendido como uma função anônima, ou seja, uma função sem nome.

É definido entre do...end ou cochetes e da mesma forma que os métodos que você viu até agora, pode receber parâmetros para sua execução

Exemplo 00

3.times { puts ("Executando o block") }

Exemplo 01

def foo

    if block\_given?

        yield

    else

        puts "Sem block de execucao"

    end

end

foo() {puts ("Hello people")}

Exemplo 02

def foo (*name*, *&block*)

    #So pode passar um bloco como parametro

    @name = name

    block.call

end

foo('Felipe'){puts ("Hello #{@name}")}

Exemplo 03

def foo (*numbers*, *&block*)

    if block\_given?

        numbers.each {|key,value|

            block.call(key, value)

        }

    end

end

def main

    numbers = {2 => 2, 3 => 3, 4 => 4}

    foo (numbers) { |key, value|

        puts ("#{key} x #{value} = #{key \* value}")

        puts ("#{key} + #{value} = #{key + value}")

        puts ("-------------")

    }

end

Lambda

São similares aos blocks, mas podem ser salvas em variáveis para serem reutilizadas

Exemplo 00

first\_lambda = lambda { puts ("My first Lambda") }

second\_lambda = -> {puts ("My second Lambda")}

first\_lambda.call

second\_lambda.call

Exemplo 01

aLambda = -> (names){

    names.each { |name|

        puts name

    }

}

names = ["João","Maria","Pedro"]

aLambda.call(names)

Exemplo 02

aLambda = lambda { |numbers|

    index = 0

    puts ("Numero atual + Proximo numero")

    numbers.each { |number|

        return if numbers[index] == numbers.last

        puts ("#{numbers[index]} + #{numbers[index+1]} = ")

        print ("#{numbers[index] +  numbers[index + 1]}")

        index += 1

    }

}

numbers = [1, 2, 3, 4]

aLambda.call(numbers)

Exemplo 03

def main (*first\_lambda*, *second\_lambda*)

    first\_lambda.call

    second\_lambda.call

end

first\_lambda = lambda { puts ("My first Lambda") }

second\_lambda = -> {puts ("My second Lambda")}

main(first\_lambda,second\_lambda)

Modules

Namespace

Serve como um container para agrupar objetos relacionados (classes, constantes, métodos ou outros modules)

Exemplo01

module ReverseWorld

    #Self indica que o metodo pertence ao modulo

    def self.puts (*text*)

        p text.reverse.to\_s

    end

    class Imprimir

        def call *text*

            puts "-=-=-=-=Imprimindo-=-=-=-=-="

            puts text

            puts "-=-=-=-=-=-Fim=-=-=-=-=-=-="

        end

    end

end

puts ("O resultado é: ")

*ReverseWorld*::puts ('O resultado é')

imprimir = *ReverseWorld*::*Imprimir*.new

imprimir.call ('Está funcionando....')

Mixins

Server para incluir funcionalidades extras as classes

module ImpressaoDecorada

    def imprimir (*text*)

        decoracao = '#' \* 20

        puts (decoracao)

        puts (text)

        puts (decoracao)

        puts ()

    end

end

module Pernas

    #Incluindo dos metodos do metodo ImpressaoDecorada

#Ato de mixing

    include ImpressaoDecorada

    def chute\_frontal

        imprimir('Chute frontal')

    end

    def chute\_lateral

        imprimir("Chute lateral")

    end

end

module Bracos

    include ImpressaoDecorada

    def jab\_Direita

        imprimir('Jab de direita')

    end

    def jab\_Esquerda

        imprimir('Jab de esquerda')

    end

    def gancho

        imprimir("Gancho")

    end

end

class LutadorX

    include Pernas

    include Bracos

end

class LutadorY

    include Pernas

end

def main

    lutadorx = *LutadorX*.new

    lutadorx.chute\_frontal()

    lutadorx.jab\_Direita()

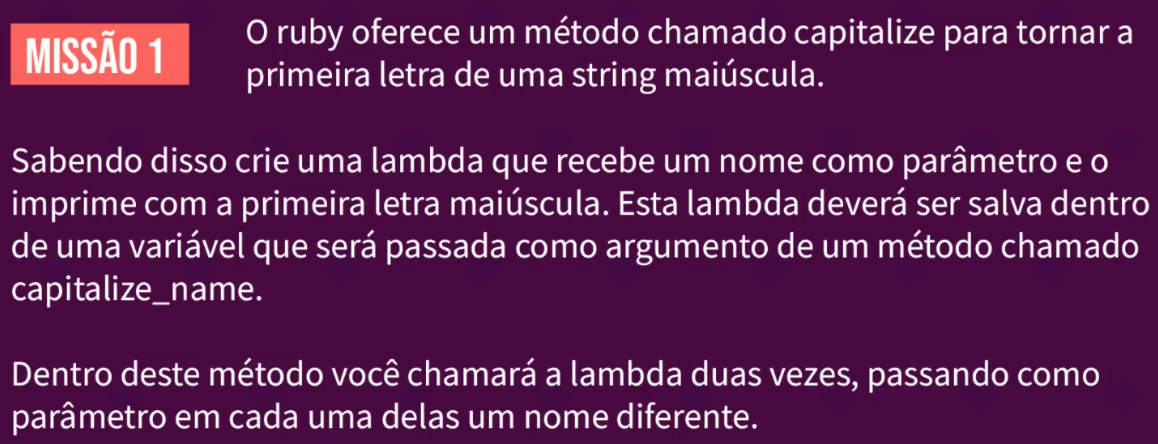
    lutadory = *LutadorY*.new

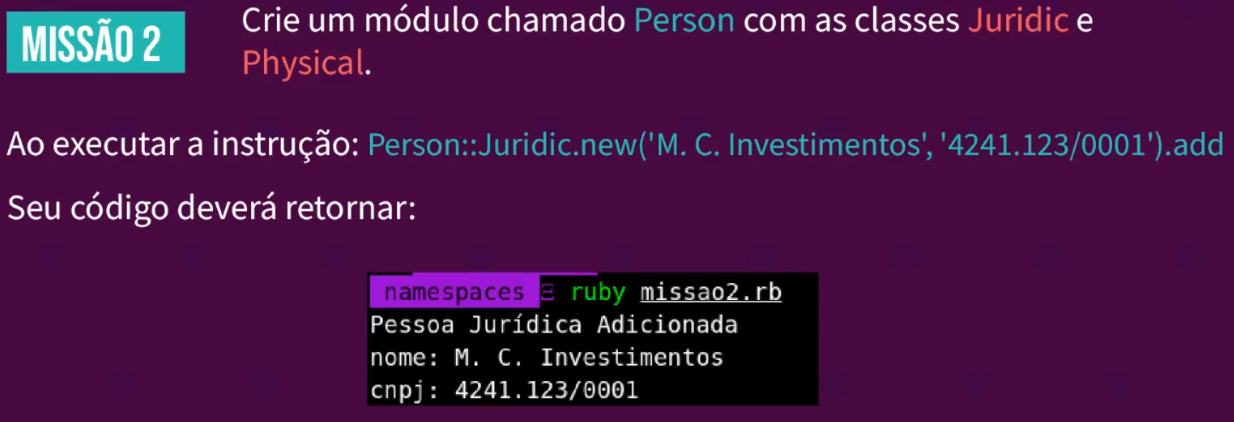
    lutadory.chute\_lateral()

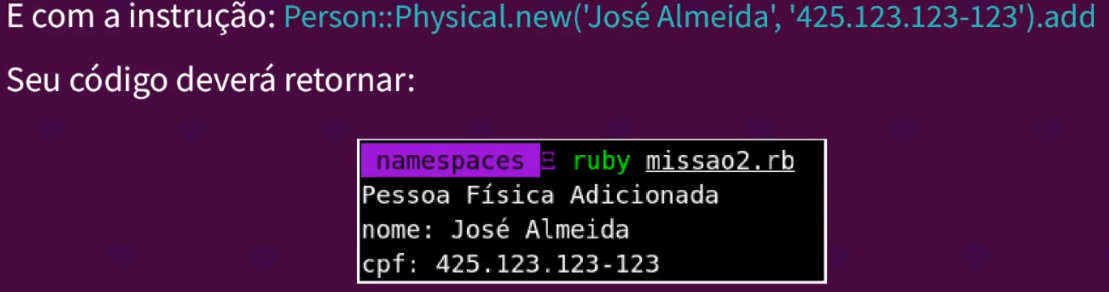
end

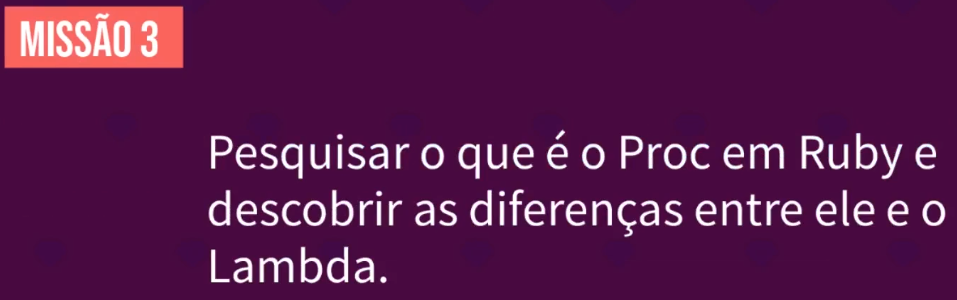
main()

Missões especiais 08









Ruby Avançado II

Regex

As expressões regulares são notações para representar padrões em strings, elas servem para validar entradas de dados, ou fazer buscas e extração de informações em textos.

Criando um regex

#Existem 3 maneiras de criar um regex

regex1 = /regex01/

regex2 = %r{regex02}

regex3 = *Regexp*.new("regex03")

Casamento de Padrões

Operador =~

Retorna posicionamento da string

frase = "Felipe Avelar aprende Ruby"

puts (/Ruby/ =~ frase)

puts (frase =~ /Ruby/)

Match

Retorna um objeto do tipo match data.

phrase = "Hello, how are you"

#retorna o objeto match

matchData = /how/.match(phrase)

#retorna o que tem antes do match

puts (matchData.pre\_match)

#retorna o que tem depois do match

puts (matchData.post\_match)

Os símbolos **(, ), [, ], {, }, ., ?, +, \***,  são metacharacters. Eles possuem um significado quando utilizados em expressões regulares.

Caso o padrão que você procura seja um metacharacter, utilize o símbolo de escape **\** para realizar a busca

/\?/.match ('Tudo bem ? Sim claro!')

Character Classes

É uma lista que informa quais caracteres devem aparecer em um ponto do casamento.

/[t]exto/.match('texto começando com t')

#Assim, é procurado um padrão onde o primeiro número pode ser 1 até 5

/[1-5]/.match('123')

#Também pode ser usado para letras

/[a-z]/.match('Oi')

Existem alguns metacharacters que se comportam como character classes

#\d verifica o padrão [0-9]

/A\d/.match('A4')

Repetion

É possível definir a repetição de um mesmo padrão, evitando escrever a mesma coisa diversas vezes.

#verifica se começa com uma letra e 3 números.

"A343".match(/[A-Z]\d{3}/) #match(/\d\d\d/)

#padrão de repetição entre três ou mais vezes

"BBB AAAA".match(/A{3,}/)

Para mais informações, documentos:

<https://ruby-doc.org/core-2.5.1/doc/regexp_rdoc.html>

Time

O Ruby conta com uma classe chamada Time para representar datas e horas

Method Missing

É utilizado para interceptar chamadas a métodos que não existem

class Fish

    def method\_missing (*method\_name*)

        puts ("Fish don't have #{method\_name} behavior")

    end

    def swim ()

        puts ("Fish is swimming")

    end

end

fish = *Fish*.new

fish.walk()

fish.swim()

Self

Self é uma variável especial que aponta para o objeto atual

Métodos privados e protegidos

Por padrão, todos os métodos definidos são públicos. Isso significa que eles podem ser acessados por qualquer um.

Mas além dos métodos públicos, existem outros dois tipos de métodos chamados private e protected

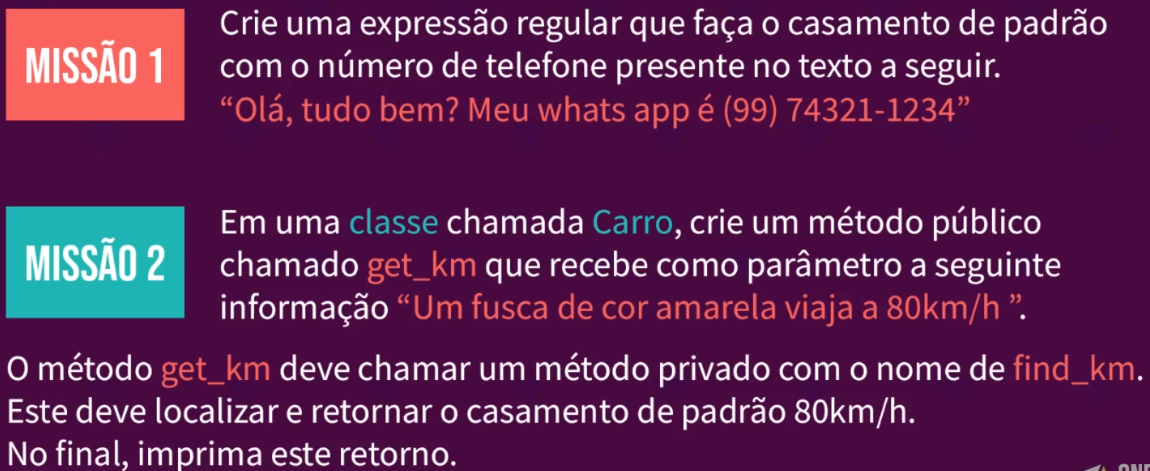
Private

Os métodos privates só podem ser chamados de dentro da própria classe.

Protected

Os métodos protegidos só podem ser chamados dentro da própria classe, suas instancias, e duas subclasses.

Missões Especiais 09



Projeto final

Leitura de arquivos

Ler e escrever em arquivos te dá o poder de criar uma grande gama de software, felizmente fazer isso no Ruby é bem fácil

Chamadas Web

Através de chamadas web podemos conectar o nosso software a outros programas através da internet. Então veremos alguns exemplos como fazer isso com a Lib Net::HTTP do Ruby.

Para aprender mais sobre API, JSons e etc...

<https://onebitcode.com/course/criando-uma-api-completa-com-rails/>

Web Scraping

Através do Web Scraping conseguimos extrair dados de sites, como por exemplo: resultados dos jogos de futebol de um portal de notícias, uma lista de posts de um blog e etc.

Para fazermos isto, usaremos uma biblioteca chamada Nokogiri, que nos ajudará a encontrar as informações dentro da estrutura HTML dos sites.

Projeto Final

Crie um software que permita que o usuário entre com um texto, o idioma em que o texto está escrito e também o idioma para qual ele deseja o texto e receba como resposta na tela o texto traduzido

Critério:

Tudo isto deve ser realizado dentro de uma classe.

O resultado de cada tradução (junto com a frase original) deve ser guardado em um arquivo que possua no nome a data e a hora da tradução (ex: 17-09-19\_10\_30\_30.txt / dd-mm-yyy-HH-MM-SS)

* Para realizar as chamadas Web utilize a Gem Rest Client (documentação): <https://github.com/rest-client/rest-client>

Para realizar a tradução utilize a API do Yandex.

* + Crie sua conta e pegue seu Token de autenticação nesta URL: <https://translate.yandex.com/developers/keys>
  + Veja como realizar a chamada nessa URL: <https://tech.yandex.com/translate/doc/dg/reference/translate-docpage/>

Solução projeto final