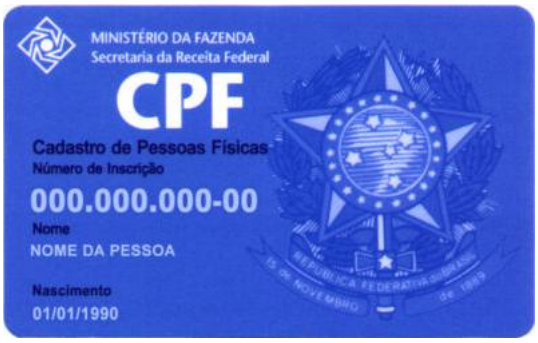
Algoritmo para Validar CPF

Gustavo Furtado de Oliveira Alves Iniciante em programação 26 Comentários

Quando se está trabalhando em um sistema corporativo, é comum a necessidade de validar CPF. Muita gente não sabe que um CPF para ser válido não basta apenas atender à máscara "###.###.###-##" (o caractere '#' representa um número), existe uma regra matemática que também deve ser verificada para um CPF ser considerado válido. Se você acha que é complicado verificar se um CPF é válido ou não, você vai se surpreender!



REGRA PARA VALIDAR CPF

O cálculo para validar um CPF é especificado pelo Ministério da Fazenda, que disponibiliza no próprio site as funções (em javascript) para validação de CPF. Vamos entender como funciona.

O CPF é formado por 11 dígitos numéricos que seguem a máscara "###.###.###-##", a verificação do CPF acontece utilizando os 9 primeiros dígitos e, com um cálculo simples, verificando se o resultado corresponde aos dois últimos dígitos (depois do sinal "-").

> *Vamos usar como exemplo, um CPF fictício "529.982.247-25".*

Validação do primeiro dígito

Primeiramente multiplica-se os 9 primeiros dígitos pela sequência decrescente de números de 10 à 2 e soma os resultados. Assim:

> *5 * 10 + 2 * 9 + 9 * 8 + 9 * 7 + 8 * 6 + 2 * 5 + 2 * 4 + 4 * 3 + 7 * 2*

O resultado do nosso exemplo é:

> *295*

O próximo passo da verificação também é simples, basta multiplicarmos esse resultado por 10 e dividirmos por 11.

> *295 * 10 / 11*

O resultado que nos interessa na verdade é o RESTO da divisão. Se ele for igual ao **primeiro dígito verificador** (primeiro dígito depois do '-'), a primeira parte da validação está correta.

Observação Importante: Se o resto da divisão for igual a 10, nós o consideramos como 0.

Categorias

Banco de Dados (10)
{ Dicas de Java } (53)
{ Dicas de Javascript } (2
{ Dicas de Programação
: Dicas de Python : (21)
Dicionário de programa
Iniciante em programaç

Artigos Recentes

Como transforma separada por vírg
Marcelo Santos de Oliv
Sem comentários ainda

Como criar um ar Python
Marcelo Santos de Oliv
Sem comentários ainda

Como criar uma V um arquivo de requirements.txt
Marcelo Santos de Oliv
Sem comentários ainda

Qual a diferença e File.pathSeparato File.separator
Gustavo Furtado de Oli
Sem comentários ainda

Javascript: Como r valores repetidos
Gustavo Furtado de Oli
Sem comentários ainda

Spring-boot: Com SQL nativo no bar
Gustavo Furtado de Oli
1 Comentário

Vamos conferir o primeiro dígito verificador do nosso exemplo:

*O resultado da divisão acima é '268' e o RESTO é **2***

Isso significa que o nosso CPF exemplo passou na validação do primeiro dígito.

Validação do segundo dígito

A validação do segundo dígito é semelhante à primeira, porém vamos considerar os 9 primeiros dígitos, mais o primeiro dígito verificador, e vamos multiplicar esses 10 números pela sequencia decrescente de 11 a 2. Vejamos:

*5 * **11** + 2 * **10** + 9 * **9** + 9 * **8** + 8 * **7** + 2 * **6** + 2 * **5** + 4 * **4** + 7 * **3** + 2 * **2***

O resultado é:

347

Seguindo o mesmo processo da primeira verificação, multiplicamos por 10 e dividimos por 11.

*347 * **10** / **11***

Verificando o RESTO, como fizemos anteriormente, temos:

*O resultado da divisão é '315' e o RESTO é **5***

Verificamos, se o resto corresponde ao segundo dígito verificador.

Com essa verificação, constatamos que o CPF 529.982.247-25 é válido.

CPFS INVÁLIDOS CONHECIDOS

Existe alguns casos de CPFs que passam nessa validação que expliquei, mas que ainda são inválidos. É os caso dos CPFs com dígitos repetidos (111.111.111-11, 222.222.222-22, ...)

Esses CPF atendem à validação, mas ainda são considerados inválidos.

No nosso algoritmo, vamos verificar se todos os dígitos do CPF são iguais e, neste caso, considerar que ele é inválido.

ALGORITMO PARA VALIDAR CPF

Agora que já aprendemos como acontece a validação de um CPF, vamos ver como ficaria um algoritmo para validar CPF. Vamos escrever o algoritmo em Portugal utilizando o Visualg.

No algoritmo abaixo, eu criei uma função chamada **validaCPF(cpf:CARACTER)** que retorna **verdadeiro** ou **falso** se o CPF for ou não válido.

Se você não sabe o que é uma função, leia este artigo.

```
algoritmo "Validação de CPF"

funcao validaCPF(cpf:CARACTER) : LOGICO
var
   num1, num2, num3, num4, num5, num6, num7, num8, num9, num10, num11, soma1, soma2 : inteiro
   resto1, resto2 : REAL
inicio

      //extrai os dígitos do CPF
      num1 := Caracpnum( Copia(cpf, 1, 1) )
      num2 := Caracpnum( Copia(cpf, 2, 1) )
      num3 := Caracpnum( Copia(cpf, 3, 1) )
      num4 := Caracpnum( Copia(cpf, 5, 1) )
      num5 := Caracpnum( Copia(cpf, 6, 1) )
      num6 := Caracpnum( Copia(cpf, 7, 1) )
      num7 := Caracpnum( Copia(cpf, 9, 1) )
```


Tags

virtualenv console
maven request stri
Default Method inteir
servidor Junit super
conversão de dados Ja
Oracle File InputStr
Sort Apache Common
Comparator converter
npm Encriptação n
variável de ambiente
Map let ordenação
linha de comando erro
Eclipse javascript IC
datetime MySQL ju
Collectors instalação
classe abstrata Scanne
Java Básico IteratorUti

```
        num8 := Caracpnum( Copia(cpf, 10, 1) )
        num9 := Caracpnum( Copia(cpf, 11, 1) )
        num10 := Caracpnum( Copia(cpf, 13, 1) )
        num11 := Caracpnum( Copia(cpf, 14, 1) )

        //Validação dos CPFs inválidos conhecidos
        SE (num1 = num2) E (num2 = num3) E (num3 = num4) E (num4 = num5) E (num5 = num6) E (num6 = num7)
           RETORNE FALSO
        SENAO

           soma1 := num1 * 10 + num2 * 9 + num3 * 8 + num4 * 7 + num5 * 6 + num6 * 5 + num7 * 4 + num8 *

           resto1 := (soma1 * 10) mod 11

           SE resto1 = 10 ENTAO
               resto1 := 0
           FIMSE

           soma2 := num1 * 11 + num2 * 10 + num3 * 9 + num4 * 8 + num5 * 7 + num6 * 6 + num7 * 5 + num8

           resto2 := (soma2 * 10) mod 11

           SE resto2 = 10 ENTAO
               resto2 := 0
           FIMSE

           SE ( resto1 = num10) E (resto2 = num11) ENTAO
              RETORNE VERDADEIRO
           SENAO
              RETORNE FALSO
           FIMSE

        FIMSE

fimfuncao

var
   cpf : CARACTER
inicio

        //Verificação de um CPF inválido
        cpf := "123.456.789-12"
        SE validaCPF(cpf) = VERDADEIRO ENTAO
           ESCREVAL("O CPF ", cpf, " é válido!")
        SENAO
           ESCREVAL("O CPF ", cpf, " é inválido!")
        FIMSE

        //Verificação de um CPF válido
        cpf := "529.982.247-25"
        SE validaCPF(cpf) = VERDADEIRO ENTAO
           ESCREVAL("O CPF ", cpf, " é válido!")
        SENAO
           ESCREVAL("O CPF ", cpf, " é inválido!")
        FIMSE

        //Verificação de CPF com dígitos iguais
        cpf := "777.777.777-77"
        SE validaCPF(cpf) = VERDADEIRO ENTAO
           ESCREVAL("O CPF ", cpf, " é válido!")
        SENAO
           ESCREVAL("O CPF ", cpf, " é inválido!")
        FIMSE

fimalgoritmo
```

Perceba que testamosa nossa função com três CPFs, um inválido e outro válido e um inválido conhecido. O resultado da execução deste algoritmo é esse.

```
O CPF 123.456.789-12 é inválido!
O CPF 529.982.247-25 é válido!
O CPF 777.777.777-77 é inválido!
```

Também utilizei algumas funções pré-definidas pelo Visualg para extrair cada caracter da variável *cpf* e para convertê-los em números inteiros. As funções que utilizei foram:

- **Caracpnum (c : caracter): inteiro**
  - Esta função serve para converter um valor do tipo texto em um valor do tipo inteiro
- **Copia (c : caracter ; p, n : inteiro): caracter**
  - Esta função serve para extrair sub-textos de variáveis texto.
  - Ela recebe três parâmetros, o primeiro é o texto de onde vamos extrair o sub-texto, o segundo é a posição de inicio do sub-texto e o terceiro parâmetro é a quantidade de caracteres que vamos extrair.

No nosso caso, nós extraímos os dígitos do cpf através da função **copia** e convertemos o resultado desta função em inteiro através da função **caracpnum**.

Por exemplo, para o cpf *"529.982.247-25"* a linha abaixo atribui o valor inteiro 8 à variável num5, pois este é o caracter da posição 6 (contando o caracter ponto ".").

```
num5 := Caracpnum( Copia(cpf, 6, 1) )
```

Outro detalhe interessante é o operador *mod* que retorna o resto da divisão.

Claro que pode-se implementar de outras formas, com **Vetores**, **LOOPs**, etc. Entretanto eu tentei implementar de uma forma mais simples de entender a regra.

Se quiser entender cada recurso utilizado neste algoritmo leia os artigos abaixo.

Estrutura de decisão SE-ENTAO-SENAO

O que são Funções e Procedimentos?

Você sabe usar os Operadores Aritméticos em programação?

Conheça os Operadores Relacionais!

O que são tipos de dados primitivos?

Quer aprender programação? Saiba qual a melhor linguagem!

Entendendo como funciona o algoritmo, você torna-se capaz de validar CPF em qualquer linguagem.

Sobre Gustavo Furtado de Oliveira Alves

É mestre em computação aplicada pelo Institudo Nacional de Pesquisas Espaciais, Engenheiro da Computação pela ETEP Faculdades e Técnico em Informática pela Escola Técnica Pandiá Calógeras. Possui as certificações AWS Architect Associate, AWS Cloud Practitioner, SCJP-6, SCWCD-5 e Agile Scrum Foundation e trabalha com desenvolvimento de softwares desde 2007.

Veja todos os artigos de Gustavo Furtado de Oliveira Alves →

TAMBÉM EM { DICAS DE PROGRAMAÇÃO }

Java, Python ou Javascript?

5 anos atrás · 10 comentários

A dúvida que fica martelando na cabeça da maioria das pessoas que ...

Qual a diferença entre JDK, JRE e JVM

3 meses atrás · 1 comentário

Uma grande confusão que paira sobre quem está começando a aprender ...

O mínimo que você precisa saber sobre ...

5 anos atrás · 6 comentários

Por ser um dos formatos mais utilizados para comunicação entre ...

Programação a Objetos: ...

7 anos atrás · 3 c

A Programação Objetos (POO, p íntimos) não é u

27 Comentários

1 Entrar

Participe da discussão...

FAZER LOGIN COM OU REGISTRE-SE NO DISQUS ?

Nome

3 Compartilhar

Mais votados Mais recentes Mais antigos

giottosjn
8 anos atrás

e como fazer esse algoritmo de validação de cpf sem usar função??

2 0 Responder • Compartilhar ›

Ciro Monteiro giottosjn
6 meses atrás

para quer fazer isso sem função mermão, rs

0 0 Responder • Compartilhar ›

Daniel Capua
6 anos atrás

gente, pra quem precisar de javascript mando abaixo o código que escrevi. No formulário dê o name="formulario1" para as tags form que contem o campo cpf e definam name=cpf_verifica para o input que pega o cpf. Além disso, aconselho deixar o javascript num arquivo separado, e deixar o form com a ação onSubmit="return validacpf();" para buscar a função corretamente.

```
function validacpf () {

var cpf = document.formulario1.cpf_verifica.value;
cpf = cpf.replace( "." , "" );
cpf = cpf.replace( "-" , "" );
cpf = cpf.replace( "." , "" );

if (cpf.length != 11 ||
cpf == "00000000000" ||
cpf == "11111111111" ||
cpf == "22222222222" ||
cpf == "33333333333" ||
cpf == "44444444444" ||
cpf == "55555555555" ||
cpf == "66666666666" ||
cpf == "77777777777" ||
cpf == "88888888888" ||
cpf == "99999999999"){
document.getElementById("cpf_verifica").style.backgroundColor = "#faa"; //isso deixa o campo
avermelhado
document.formulario1.cpf_verifica.focus();
return false;
} else {
var soma = 0;
soma = soma + (parseInt(cpf.substring( 0 , 1))) * 10;
soma = soma + (parseInt(cpf.substring( 1 , 2))) * 9;
soma = soma + (parseInt(cpf.substring( 2 , 3))) * 8;
soma = soma + (parseInt(cpf.substring( 3 , 4))) * 7;
soma = soma + (parseInt(cpf.substring( 4 , 5))) * 6;
soma = soma + (parseInt(cpf.substring( 5 , 6))) * 5;
soma = soma + (parseInt(cpf.substring( 6 , 7))) * 4;
soma = soma + (parseInt(cpf.substring( 7 , 8))) * 3;
soma = soma + (parseInt(cpf.substring( 8 , 9))) * 2;
}

var resto1 = (soma * 10) % 11;

if ((resto1 == 10) || (resto1 == 11)) {
```

```
resto1 = 0;
}

var soma = 0;
soma = soma + (parseInt(cpf.substring( 0 , 1))) * 11;
soma = soma + (parseInt(cpf.substring( 1 , 2))) * 10;
soma = soma + (parseInt(cpf.substring( 2 , 3))) * 9;
soma = soma + (parseInt(cpf.substring( 3 , 4))) * 8;
soma = soma + (parseInt(cpf.substring( 4 , 5))) * 7;
soma = soma + (parseInt(cpf.substring( 5 , 6))) * 6;
soma = soma + (parseInt(cpf.substring( 6 , 7))) * 5;
soma = soma + (parseInt(cpf.substring( 7 , 8))) * 4;
soma = soma + (parseInt(cpf.substring( 8 , 9))) * 3;
soma = soma + (parseInt(cpf.substring( 9 , 10))) * 2;

var resto2 = (soma *10) % 11;
if ((resto2 == 10) || (resto2 == 11)) {
resto2 = 0;
}

if (
(resto1 == (parseInt(cpf.substring( 9 , 10)))) &&
(resto2 == (parseInt(cpf.substring( 10 , 11))))  ) {
alert("deuserto");
return true;
} else {
alert ("CPF inválido")
document.getElementById("cpf_verifica").style.backgroundColor = "#faa";
document.formulario1.cpf_verifica.focus();
return false;
}

}
```

1 0 Responder • Compartilhar ›

Paulo Quadros
8 anos atrás

Eu fiz exatamente assim, mas achei um "furo" na formula.
Sempre que o CPF for numeros repetidos (111111111-11, 222222222-22 ... 999999999-99) ele passa como valido, mas nos sites de validacao aparecem como invalidos, entao deve ter algum erro na formula, imagino eu. Ja fiz na mao o teste (papel e caneta) e realmente, nesta formula passada no algoritimo, o CPF 111111111-11 seria valido.
Qual a formula correta? Encontrei algo parecido em outro site.
Nesse aqui: http://www.geradorcpf.com/a...
Testem ai, valeu abraco!

1 0 Responder • Compartilhar ›

Gustavo Furtado → Paulo Quadros
8 anos atrás

Paulo, muito obrigado pelo comentário.

Realmente eu descobri que esses casos são os CPFs inválidos conhecidos.
Vi o algoritmo de validação do site que você indicou. http://www.geradorcpf.com/s...

A validação adicional que eles fazem é essa:

```
if (cpf.length != 11 || cpf == "00000000000" || cpf == "11111111111" || cpf == "22222222222" || cpf == "33333333333" || cpf == "44444444444" || cpf == "55555555555" || cpf == "66666666666" || cpf == "77777777777" || cpf == "88888888888" || cpf == "99999999999")
return false;
```

Vou acrescentar essa validação no algoritmo deste post.

Novamente muito obrigado.

3 0 Responder • Compartilhar ›

Mr. Blue Sky
2 meses atrás edited

```
def checar_cpf(cpf : str):
l = lambda N,x : sum([N[i] * (x - i) for i in range(x - 1)]) * 10 % 11 % 10
```

```
if len(N := [int(n) for n in cpf if n.isdigit()]) == N.count(N[0]): return False # Cpfs inválidos (OPCIONAL)
return False if len(N) != 11 else [I(N,10), I(N,11)] == N[9:11]
```

Menor versão possível em Python. Claro que ainda é possível colocar lambda dentro da última linha e fazer a validação da segunda na última como alternativa, entretanto ficaria ilegível.

0 0 Responder • Compartilhar ›

C **Ciro Monteiro**
6 meses atrás

uma form de fazer dentro paradigma funcional para executar esse algortimo no javascript:

```
const checarValidadeCpf = (cpf) => {
const nCpf = cpf.split("");
const igual = n.reduce( (a,b) => a==b? true : false);
if (igual) {
console.log("cpf inválido. Números todos iguais")
} else {
const restoDivisao1 = checarDigito(nCpf, nCpf[9], 10);
const restoDivisao2 = checarDigito(nCpf, nCpf[10], 11);

if (restoDivisao1 && restoDivisao2) {
console.log("número de cpf válido")
} else {
console.log("Atenção!!! número de cpf inválido")
}
}
}

function checarDigito (array, digito, dec){
let acc = 0;
const contador = dec - 1;
for (let i = 0; i < contador; i++) {
acc += array[i] * dec;
dec--;
}
let restoDivisao = (acc * 10) % 11;
restoDivisao = restoDivisao == 10 ? 0 : restoDivisao;
return restoDivisao == digito;
}
```

0 0 Responder • Compartilhar ›

P **Paulo Rodrigues da Silva**
10 meses atrás

Mestre Gustavo, ao verificar se a soma dos dígitos é diferente de 44 no CPF já elimita completamente a necessidade de verificar os números considerados "não aceitos" pela Receita Federal. Afinal, tais números "bloqueados" dão como soma sempre diferente de 44. Ou há algo que eu não entendi?

0 0 Responder • Compartilhar ›

Marcos Augusto Da Silva Rodrig
um ano atrás

```
import java.util.Scanner;
public class cpf {

public static void main(String[] args) {
Scanner tl = new Scanner(System.in);

int n1,n2,n3,n4,n5,n6,n7,n8,n9,n10,n11,resultado1,resultado2,resultado3,resultado4;

System.out.println(" informe ");
n1 = tl.nextInt();
System.out.println(" informe ");
n2 = tl.nextInt();
System.out.println(" informe ");
n3 = tl.nextInt();
System.out.println(" informe ");
n4 = tl.nextInt();
System.out.println(" informe ");
n5 = tl.nextInt();
System.out.println(" informe ");
n6 = tl.nextInt();
```

```
n6 = tl.nextInt();
System.out.println(" informe ");
n7 = tl.nextInt();
System.out.println(" informe ");
n8 = tl.nextInt();
System.out.println(" informe ");
n9 = tl.nextInt();
System.out.println(" informe ");
n10 = tl.nextInt();
System.out.println(" informe ");
n11 = tl.nextInt();

resultado1 = (n1*10)+(n2*9)+(n3*8)+(n4*7)+(n5*6)+(n6*5)+(n7*4)+(n8*3)+(n9*2);

resultado2 = resultado1*10%11;

//tratamento caso o resto da divisão for igual a 10

if (resultado2 ==10) {

resultado2=0;
} else {
resultado2 = n10 ;
}

resultado3= (n1*11)+(n2*10)+(n3*9)+(n4*8)+(n5*7)+(n6*6)+(n7*5)+(n8*4)+(n9*3)+(n10*2);

resultado4 = resultado3*10%11;

if (resultado4 ==10) {

resultado4=0;
} else {
resultado4 = n11 ;
}

if((resultado2 == 10) && (resultado2==10)){
resultado2=0;
resultado4=0;

} else {
boolean b = n11==n11;
if((n1==n2) && (n2==n3)&&(n3==n4)&&(n4==n5)&&(n5==n6)&&(n6==n7)&&(n7==n8)&&(n8==n9)&&
(n9==n10)&&(n10==n11)&&b){
System.out.println("O CPF informado " + n1+n2+n3+n4+n5+n6+n7+n8+n9+" - " + n10+n11 + " não valido
");

}else if ((resultado2==n10)&& (resultado4==n11)) {
System.out.println("O CPF informado " + n1+n2+n3+n4+n5+n6+n7+n8+n9+" - " + n10+n11 + " valido ");

} else {
System.out.println("O CPF informado " + n1+n2+n3+n4+n5+n6+n7+n8+n9+" - " + n10+n11 + " não valido
");

}
}

}
}
```

0 0 Responder • Compartilhar ›

eliseunetto

2 anos atrás

Galera gostaria de compartilhar com vocês esta validação de CPF e CNPJ em Java que esta no meu git: https://gist.github.com/eli...

0 0 Responder • Compartilhar ›

M

Michael

3 anos atrás

Codigo em Dart

Aceita em formato "111111111-11"

```
bool validateCpf(String bruteCpf) {

RegExp re = new RegExp(r'(\D)');

var cpf = cpfString.replaceAll(re, "");

if (cpf.length < 11) return false;

var cpfList = cpf.split("");

// verifica se todos os digitos sao iguais
bool areAllDigitsTheSame = true;
for (int i = 0; i < cpfList.length; i++) {
if (cpfList[i] != cpfList[0]) {
areAllDigitsTheSame = false;
break;
}
}

if (areAllDigitsTheSame) return false;

// funcao para somar o resultado
int loopCpf(int loops) {
var result = 0;
for (int i = loops; i > 1; i--) {
result += int.parse(cpfList[(i - loops).abs()]) * i;
}
return result;
}

// funcao para vericar os digitos
bool verifyDigit(int loop, int digitToVerify) {
var result = loopCpf(loop);
var remaining = (result * 10) % 11;
if (remaining == 10 || remaining == 11) remaining = 0;
return remaining.toString() == cpf[digitToVerify];
}

if (!verifyDigit(10, 9)) return false;
return (verifyDigit(11, 10));
}
```

0 0 Responder • Compartilhar ›

Moises Silva

4 anos atrás

Galera se alguém quiser em python, eu fiz essa versão. Livre para att e melhoras

https://github.com/MoisesFa...

0 0 Responder • Compartilhar ›

Joca Kulaif Moises Silva

2 anos atrás

Link está quebrado. Mas caso alguém queira ainda em python3, segue !

```
def valida_cpf(cpf: str) -> bool:
"""Função para validação de CPF
Recebe: CPF (formato String)
Retorna: True or False
"""
import re

def calcula_digito(cpf: str, digito_no: int, cont=10) -> int:
if digito_no == 1:
slice = cpf[0:9]
digito = int(cpf[-2])
else:
slice = cpf[1:10]
digito = int(cpf[-1])

calculo = 0
```

```
for i in slice:
calculo += int(i) * cont
cont -= 1
digito_calculado = round((calculo * 10) % 11, 2)

# regra que caso o resto seja 10, o digito passa a ser 0
digito_calculado = round(digito_calculado, 1) if digito_calculado != 10 else 0

return digito_calculado == digito

cpfs_invalidos = [
"11111111111",
"22222222222",
"33333333333",
"44444444444",
"55555555555",
"66666666666",
"77777777777",
"88888888888",
"99999999999",
"00000000000",
]

# limpa cpf deixando somente números
cpf_digitos = "".join(re.findall("\d+", cpf))

if cpf_digitos in cpfs_invalidos:
return False

digito1_verificacao = calcula_digito(cpf_digitos, 1)
digito2_verificacao = calcula_digito(cpf_digitos, 2)

return digito1_verificacao and digito2_verificacao
```

0 0 Responder • Compartilhar ›

Rafael Lobo

5 anos atrás

Vejam um validador feito em python, tanto para cnpj quanto para cpf:

https://github.com/rafahlob...

0 0 Responder • Compartilhar ›

Erick Alessi

5 anos atrás

Em COBOL, pode testar no http://www.compileonline.co... , essa versão não é blocada, então se for utilizar em algum TSO ou sistema tradicional tem que fazer a tabulação correta. O cpf entra no STDIN, sem pontos.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CPFVALID.

ENVIRONMENT DIVISION.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 CPF PIC 9(11) VALUE ZEROES.

01 INDICES.
05 IND1 PIC 9(2) VALUE ZEROES.
05 IND2 PIC 9(2) VALUE ZEROES.

77 AUX PIC 9(1) VALUE ZEROES.

77 ACUMULADOR PIC 9(4) VALUE ZEROES.

77 VALIDO PIC 9(1) VALUE 1.

PROCEDURE DIVISION.

PRINCIPAL SECTION.

PERFORM INICIALIZAR.
PERFORM PROCESSAR.
PERFORM FINALIZAR.
```

```
PERFORM FINALIZAR.

PRINCIPAL-FIM. EXIT.

INICIALIZAR SECTION.

ACCEPT CPF FROM STDIN.

INICIALIZAR-FIM. EXIT.

PROCESSAR SECTION.

PERFORM TESTE-DEZ-DIGITO.
PERFORM TESTE-ONZE-DIGITO.
PERFORM TESTE-IGUAL.

PROCESSAR-FIM. EXIT.

TESTE-DEZ-DIGITO SECTION.

MOVE 1 TO IND1.
PERFORM VARYING IND2 FROM 10 BY -1 UNTIL IND2 < 2
MOVE CPF(IND1:1) TO AUX
COMPUTE ACUMULADOR = ACUMULADOR + AUX * ind2
ADD 1 TO IND1
END-PERFORM.

COMPUTE ACUMULADOR = ACUMULADOR * 10.
DIVIDE ACUMULADOR BY 11 GIVING AUX REMAINDER AUX

IF AUX EQUAL CPF(10:1)
MOVE 1 TO VALIDO
ELSE
MOVE 0 TO VALIDO
PERFORM FINALIZAR
END-IF.

TESTE-DEZ-DIGITO-FIM. EXIT.

TESTE-ONZE-DIGITO SECTION.

MOVE 1 TO IND1.
MOVE 0 TO ACUMULADOR.
PERFORM VARYING IND2 FROM 11 BY -1 UNTIL IND2 < 2
MOVE CPF(IND1:1) TO AUX
COMPUTE ACUMULADOR = ACUMULADOR + AUX * ind2
ADD 1 TO IND1
END-PERFORM.

COMPUTE ACUMULADOR = ACUMULADOR * 10.
DIVIDE ACUMULADOR BY 11 GIVING AUX REMAINDER AUX

IF AUX EQUAL CPF(11:1)
MOVE 1 TO VALIDO
ELSE
MOVE 0 TO VALIDO
PERFORM FINALIZAR
END-IF.

TESTE-ONZE-DIGITO-FIM. EXIT.

TESTE-IGUAL SECTION.

PERFORM VARYING IND1 FROM 1 BY 1 UNTIL IND1 > 10
COMPUTE IND2 = IND1 + 1
IF CPF(IND1:1) EQUAL CPF(IND2:1)
MOVE 0 TO VALIDO
ELSE
MOVE 1 TO VALIDO
END-IF
END-PERFORM.

TESTE-IGUAL-FIM. EXIT.

FINALIZAR SECTION.

IF VALIDO EQUAL 1
```

DISPLAY CPF ' VALIDO'
ELSE
IF VALIDO EQUAL 0
DISPLAY CPF ' INVALIDO'
END-IF.
STOP RUN.

FINALIZAR-FIM. EXIT.

0 0 Responder • Compartilhar ›

Marcus Eduardo Motta Duarte
5 anos atrás

Escrevi ele em Dart, para quem desenvolve usando Flutter.io.
Segue o gist: https://gist.github.com/mar...

0 0 Responder • Compartilhar ›

M **Mateus Gomes**
6 anos atrás

rapaz tava querendo criar um algoritmo ni visual g pra ler codigos binarios alguem tem alguma informacao ai obrugado!

0 0 Responder • Compartilhar ›

CL **CRISTIANO LOPES BORGES**
8 anos atrás

CARO GUSTAVO QUERO UM LINK PARA ACESSA SUA BIBLIOGRAFIA POR QUE ESTOU FAZENDO UM TRABALHO ACADEMICO PRA UNOPAR

0 0 Responder • Compartilhar ›

CL **CRISTIANO LOPES BORGES**
8 anos atrás

QUERO UM LINK PARA ACESSA SUA BIBLIOGRAFIA POR QUE ESTOU FAZENDO UM TRABALHO ACADEMICO PRA UNOPAR

0 0 Responder • Compartilhar ›

CF **Carlos Fadin**
8 anos atrás

Estava procurando por este algoritmo Gustavo.
Obrigado por posta-lo.

0 0 Responder • Compartilhar ›

GW **Gerson W. Barbosa**
8 anos atrás

O programa que enviei ontem tinha um erro: quando o primeiro dígito verificador era 10 (= 0), o segundo dígito era calculado erradamente. Agora está correto:

```
Program CPF;
Uses Crt;
var d1,d2, i: Byte;
s, t: Integer;
n: Longint;
begin
ClrScr;
Read(n);
s:=0;
t:=0;
for i:=1 to 9 do
begin
s:=s+(10-i)*(n Mod 10);
t:=t+i*(n Mod 10);
n:=n div 10
end;
if (s Mod 11)=10 then
t:=t+9;
d1:=(s Mod 11) Mod 10;
```

```
d2:=(t Mod 11) Mod 10;
GotoXY(10,1);
WriteLn('-',d1:1,d2:1)
end.
```

Run

123456789

123456789-09
Type EXIT to return...

Note-se que o programa não é exatamente um validador de CPF, mas pode ser facilmente modificado para tal. Dados os primeiros nove dígitos do CPF, os dois dígitos verificadores são calculados usando um algoritmo diferente daquele disponível no site da Receita (e mais simples).
Estou sem o Dev-C++ instalado, mas acho que o código Pascal é suficientemente claro para demonstrar o algoritmo.

0 0 Responder • Compartilhar ›

Franklyn Roberto Da Silva → Gerson W. Barbosa

5 anos atrás edited

```
isso é em python 3.7
casos = int(input())
lmfao = 1
for k in range(casos):
lista1 = [10,9,8,7,6,5,4,3,2]
lista3 = [11,10,9,8,7,6,5,4,3,2]
lista_vazia1 = []
lista_vazia2 = []
a, b, c, d, e, f, g, h, i = map(int,input().split())

lista_das_variaveis = [a,b,c,d,e,f,g,h,i]

for i in range(9):
a =lista_das_variaveis[i]* lista1[i]
lista_vazia1.append(a)

o = sum(lista_vazia1)
o = (o *10)%11

if o == 10:
o = 0
lista_das_variaveis.append(o)

else:
lista_das_variaveis.append(o)

for y in range(10):
a1 = lista_das_variaveis[y] * lista3[y]
lista_vazia2.append(a1)

p = sum(lista_vazia2)
p = (p*10)%11
if p == 10:
p = 0
lista_das_variaveis.append(p)
else:
lista_das_variaveis.append(p)

lista_das_variaveis[9:9] = '-'
p = str(lista_das_variaveis).strip("[]")
p1 = p.replace(",","")
p1 = p1.replace(" ","")
p1 = p1.replace("'","")
print("Caso {}: {}".format(lmfao,p1))

lmfao += 1
```

0 0 Responder • Compartilhar ›

MM Marcelo Martins
8 anos atrás

Eu não entendi o porque do 5 * 10 + 2 , porque esse + 2 no final ?
Desde já , agradeço.

0 0 Responder • Compartilhar ›

GF Gustavo Furtado → Marcelo Martins
8 anos atrás

Marcelo, perceba que a expressão ainda não terminou ...
5 * 10 + 2 * 9 + ...
Ou seja, 2 (o segundo número do CPF) * 9.

0 0 Responder • Compartilhar ›

M muitobom@gmail.com → Gustavo Furtado
8 anos atrás

muito bom

0 0 Responder • Compartilhar ›

DG Daniel Gomes
9 anos atrás

Muito bom mesmo.Agora é só praticar.

0 0 Responder • Compartilhar ›

J Join
9 anos atrás

Muito interessante. Vou implementar em C.

0 0 Responder • Compartilhar ›

Inscreva-se Privacidade Política de Proteção de Dados