### Algoritmo Adition Chaining

Descrito no excelente livro *Applied Cryptography* de Bruce Schneier de 1996, (pág. 244) ele permite fazer exponenciações enormes como por exemplo $546789^{23543}$ = ..... . cuja resposta não aparece aqui, mas certamente é um número além de qualquer capacidade de representação usual. Obviamente os algoritmos usuais do ambiente computacional são claramente inadequados (incapazes) de obter este número.

O algoritmo que vai ser estudado toma partido do fato de estarmos em matemática modular, (também conhecida na escola como aritmética do relógio) ou seja ao lado da expressão há algum módulo finito.

```
longo função XELEVADOAY (longo X, Y, N)
   longo TMP
   se Y = 1 então
      devolva (X mod N) {em C ou Python: X % N}
   senão
      se (Y mod 2) = 0 então
         TMP ← XELEVADOAY (X, (Y/2), N)
         devolva (TMP * TMP) mod N
      senão
         TMP ← XELEVADOAY (X, ((Y-1)/2), N)
         TMP ← (TMP * TMP) mod N
         TMP ← (TMP * X) mod N
         devolva TMP
      fim se
   fim se
fim função
```

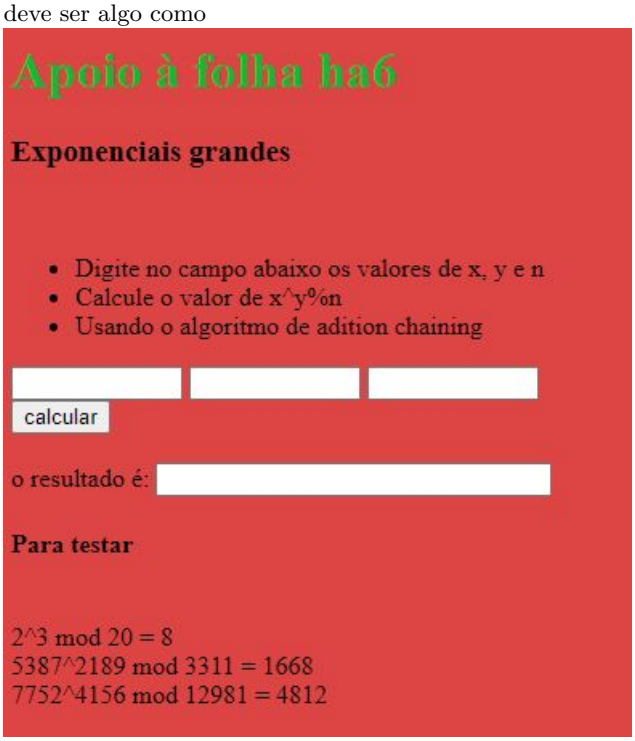
Depois de digitar, use os seguintes exemplos para testar o programa
$(2^3)$ *mod* 20 = 8
$(5387^{2189})$ *mod* 3311 = 1668
$(7752^{4156})$ *mod* 12981 = 4812

### Código

Nesta folha não há código, você deve desenvolvê-lo.

### A tela

deve ser algo como




### ☞ Para você fazer

Primeiro, mostre para o professor sua implementação

A seguir, Ache $15653^{19458}$ *mod* 16510
Responda aqui:

### Algoritmo Adition Chaining

Descrito no excelente livro *Applied Cryptography* de Bruce Schneier de 1996, (pág. 244) ele permite fazer exponenciações enormes como por exemplo $546789^{23543} =$ ..... . cuja resposta não aparece aqui, mas certamente é um número além de qualquer capacidade de representação usual. Obviamente os algoritmos usuais do ambiente computacional são claramente inadequados (incapazes) de obter este número.

O algoritmo que vai ser estudado toma partido do fato de estarmos em matemática modular, (também conhecida na escola como aritmética do relógio) ou seja ao lado da expressão há algum módulo finito.

```
longo função XELEVADOAY (longo X, Y, N)
   longo TMP
   se Y = 1 então
      devolva (X mod N) {em C ou Python: X % N}
   senão
      se (Y mod 2) = 0 então
         TMP ← XELEVADOAY (X, (Y/2), N)
         devolva (TMP * TMP) mod N
      senão
         TMP ← XELEVADOAY (X, ((Y-1)/2), N)
         TMP ← (TMP * TMP) mod N
         TMP ← (TMP * X) mod N
         devolva TMP
      fim se
   fim se
fim função
```

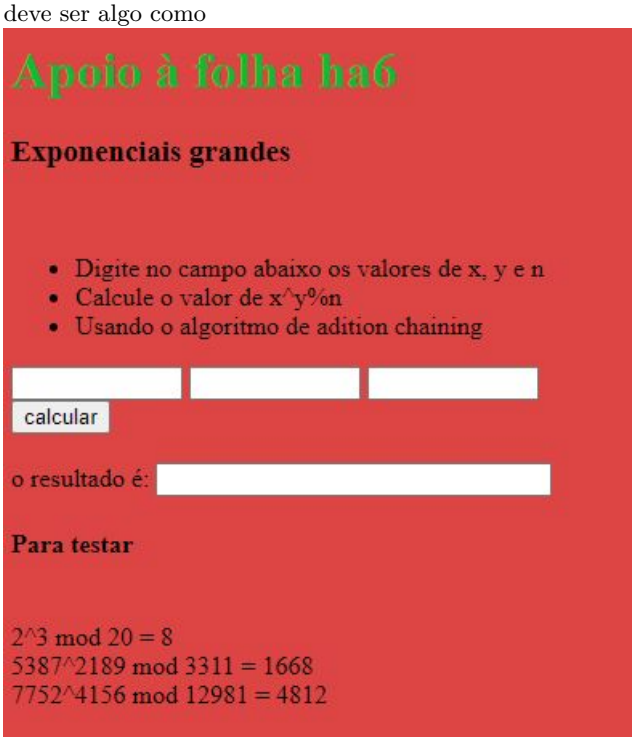
Depois de digitar, use os seguintes exemplos para testar o programa
$(2^3)$ *mod* $20 = 8$
$(5387^{2189})$ *mod* $3311 = 1668$
$(7752^{4156})$ *mod* $12981 = 4812$

### Código

Nesta folha não há código, você deve desenvolvê-lo.

### A tela

deve ser algo como




### ☞ Para você fazer

Primeiro, mostre para o professor sua implementação

A seguir, Ache $12687^{14283}$ *mod* 18981
Responda aqui:

### Algoritmo Adition Chaining

Descrito no excelente livro *Applied Cryptography* de Bruce Schneier de 1996, (pág. 244) ele permite fazer exponenciações enormes como por exemplo $546789^{23543}$ = ..... . cuja resposta não aparece aqui, mas certamente é um número além de qualquer capacidade de representação usual. Obviamente os algoritmos usuais do ambiente computacional são claramente inadequados (incapazes) de obter este número.

O algoritmo que vai ser estudado toma partido do fato de estarmos em matemática modular, (também conhecida na escola como aritmética do relógio) ou seja ao lado da expressão há algum módulo finito.

```
longo função XELEVADOAY (longo X, Y, N)
   longo TMP
   se Y = 1 então
      devolva (X mod N) {em C ou Python: X % N}
   senão
      se (Y mod 2) = 0 então
         TMP ← XELEVADOAY (X, (Y/2), N)
         devolva (TMP * TMP) mod N
      senão
         TMP ← XELEVADOAY (X, ((Y-1)/2), N)
         TMP ← (TMP * TMP) mod N
         TMP ← (TMP * X) mod N
         devolva TMP
      fim se
   fim se
fim função
```

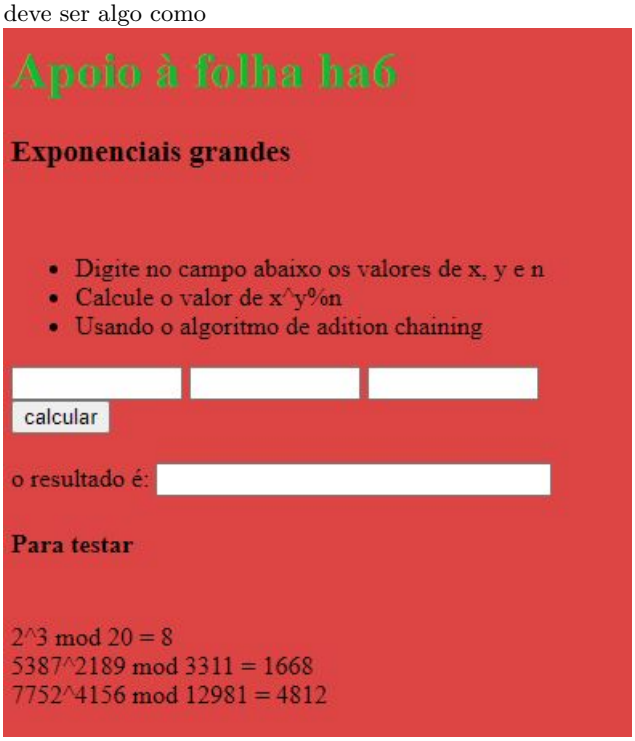
Depois de digitar, use os seguintes exemplos para testar o programa
$(2^3)$ *mod* $20 = 8$
$(5387^{2189})$ *mod* $3311 = 1668$
$(7752^{4156})$ *mod* $12981 = 4812$

### Código

Nesta folha não há código, você deve desenvolvê-lo.

### A tela

deve ser algo como




### ☞ Para você fazer

Primeiro, mostre para o professor sua implementação

A seguir, Ache $13804^{17131}$ *mod* 15645
Responda aqui:

### Algoritmo Adition Chaining

Descrito no excelente livro *Applied Cryptography* de Bruce Schneier de 1996, (pág. 244) ele permite fazer exponenciações enormes como por exemplo $546789^{23543} =$ ..... . cuja resposta não aparece aqui, mas certamente é um número além de qualquer capacidade de representação usual. Obviamente os algoritmos usuais do ambiente computacional são claramente inadequados (incapazes) de obter este número.

O algoritmo que vai ser estudado toma partido do fato de estarmos em matemática modular, (também conhecida na escola como aritmética do relógio) ou seja ao lado da expressão há algum módulo finito.

```
longo função XELEVADOAY (longo X, Y, N)
   longo TMP
   se Y = 1 então
      devolva (X mod N) {em C ou Python: X % N}
   senão
      se (Y mod 2) = 0 então
         TMP ← XELEVADOAY (X, (Y/2), N)
         devolva (TMP * TMP) mod N
      senão
         TMP ← XELEVADOAY (X, ((Y-1)/2), N)
         TMP ← (TMP * TMP) mod N
         TMP ← (TMP * X) mod N
         devolva TMP
      fim se
   fim se
fim função
```

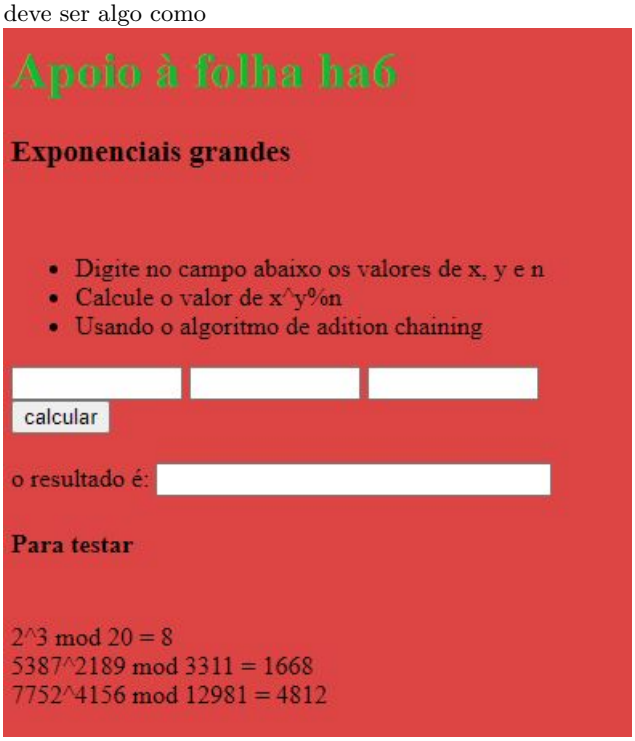
Depois de digitar, use os seguintes exemplos para testar o programa
$(2^3)$ *mod* $20 = 8$
$(5387^{2189})$ *mod* $3311 = 1668$
$(7752^{4156})$ *mod* $12981 = 4812$

### Código

Nesta folha não há código, você deve desenvolvê-lo.

### A tela

deve ser algo como




### ☞ Para você fazer

Primeiro, mostre para o professor sua implementação

A seguir, Ache $11105^{15168}$ *mod* 17810
Responda aqui:

### Algoritmo Adition Chaining

Descrito no excelente livro *Applied Cryptography* de Bruce Schneier de 1996, (pág. 244) ele permite fazer exponenciações enormes como por exemplo $546789^{23543} =$ ..... . cuja resposta não aparece aqui, mas certamente é um número além de qualquer capacidade de representação usual. Obviamente os algoritmos usuais do ambiente computacional são claramente inadequados (incapazes) de obter este número.

O algoritmo que vai ser estudado toma partido do fato de estarmos em matemática modular, (também conhecida na escola como aritmética do relógio) ou seja ao lado da expressão há algum módulo finito.

```
longo função XELEVADOAY (longo X, Y, N)
   longo TMP
   se Y = 1 então
      devolva (X mod N) {em C ou Python: X % N}
   senão
      se (Y mod 2) = 0 então
         TMP ← XELEVADOAY (X, (Y/2), N)
         devolva (TMP * TMP) mod N
      senão
         TMP ← XELEVADOAY (X, ((Y-1)/2), N)
         TMP ← (TMP * TMP) mod N
         TMP ← (TMP * X) mod N
         devolva TMP
      fim se
   fim se
fim função
```

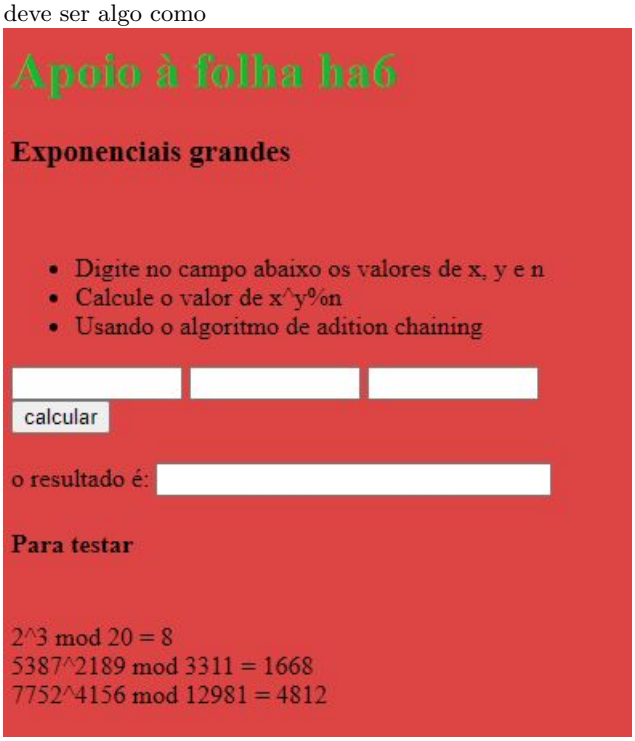
Depois de digitar, use os seguintes exemplos para testar o programa
$(2^3)$ *mod* $20 = 8$
$(5387^{2189})$ *mod* $3311 = 1668$
$(7752^{4156})$ *mod* $12981 = 4812$

### Código

Nesta folha não há código, você deve desenvolvê-lo.

### A tela

deve ser algo como




### ☞ Para você fazer

Primeiro, mostre para o professor sua implementação

A seguir, Ache $19808^{15656}$ *mod* 12720
Responda aqui:

### Algoritmo Adition Chaining

Descrito no excelente livro *Applied Cryptography* de Bruce Schneier de 1996, (pág. 244) ele permite fazer exponenciações enormes como por exemplo $546789^{23543} =$ ..... . cuja resposta não aparece aqui, mas certamente é um número além de qualquer capacidade de representação usual. Obviamente os algoritmos usuais do ambiente computacional são claramente inadequados (incapazes) de obter este número.

O algoritmo que vai ser estudado toma partido do fato de estarmos em matemática modular, (também conhecida na escola como aritmética do relógio) ou seja ao lado da expressão há algum módulo finito.

```
longo função XELEVADOAY (longo X, Y, N)
   longo TMP
   se Y = 1 então
      devolva (X mod N) {em C ou Python: X % N}
   senão
      se (Y mod 2) = 0 então
         TMP ← XELEVADOAY (X, (Y/2), N)
         devolva (TMP * TMP) mod N
      senão
         TMP ← XELEVADOAY (X, ((Y-1)/2), N)
         TMP ← (TMP * TMP) mod N
         TMP ← (TMP * X) mod N
         devolva TMP
      fim se
   fim se
fim função
```

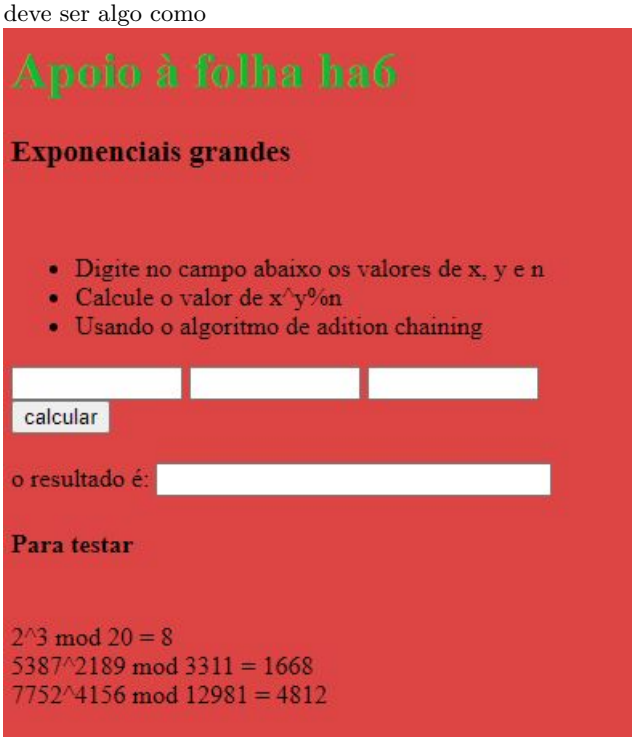
Depois de digitar, use os seguintes exemplos para testar o programa
$(2^3)$ *mod* $20 = 8$
$(5387^{2189})$ *mod* $3311 = 1668$
$(7752^{4156})$ *mod* $12981 = 4812$

### Código

Nesta folha não há código, você deve desenvolvê-lo.

### A tela

deve ser algo como




### ☞ Para você fazer

Primeiro, mostre para o professor sua implementação

A seguir, Ache $10873^{17375}$ *mod* 11515
Responda aqui:

### Algoritmo Adition Chaining

Descrito no excelente livro *Applied Cryptography* de Bruce Schneier de 1996, (pág. 244) ele permite fazer exponenciações enormes como por exemplo $546789^{23543} =$ ..... . cuja resposta não aparece aqui, mas certamente é um número além de qualquer capacidade de representação usual. Obviamente os algoritmos usuais do ambiente computacional são claramente inadequados (incapazes) de obter este número.

O algoritmo que vai ser estudado toma partido do fato de estarmos em matemática modular, (também conhecida na escola como aritmética do relógio) ou seja ao lado da expressão há algum módulo finito.

```
longo função XELEVADOAY (longo X, Y, N)
   longo TMP
   se Y = 1 então
      devolva (X mod N) {em C ou Python: X % N}
   senão
      se (Y mod 2) = 0 então
         TMP ← XELEVADOAY (X, (Y/2), N)
         devolva (TMP * TMP) mod N
      senão
         TMP ← XELEVADOAY (X, ((Y-1)/2), N)
         TMP ← (TMP * TMP) mod N
         TMP ← (TMP * X) mod N
         devolva TMP
      fim se
   fim se
fim função
```

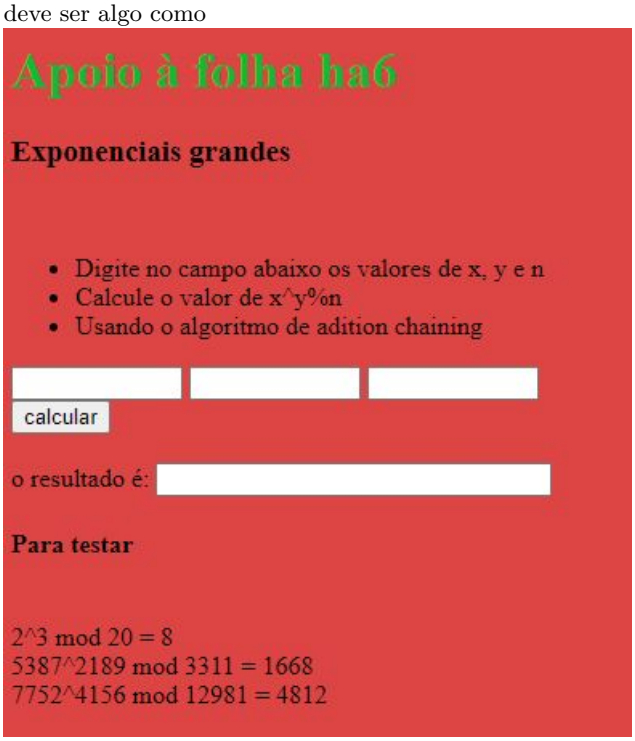
Depois de digitar, use os seguintes exemplos para testar o programa
$(2^3)$ *mod* $20 = 8$
$(5387^{2189})$ *mod* $3311 = 1668$
$(7752^{4156})$ *mod* $12981 = 4812$

### Código

Nesta folha não há código, você deve desenvolvê-lo.

### A tela

deve ser algo como




### ☞ Para você fazer

Primeiro, mostre para o professor sua implementação

A seguir, Ache $12398^{17115}$ *mod* 10678
Responda aqui:

## Algoritmo Adition Chaining

Descrito no excelente livro *Applied Cryptography* de Bruce Schneier de 1996, (pág. 244) ele permite fazer exponenciações enormes como por exemplo $546789^{23543}$ = ..... . cuja resposta não aparece aqui, mas certamente é um número além de qualquer capacidade de representação usual. Obviamente os algoritmos usuais do ambiente computacional são claramente inadequados (incapazes) de obter este número.

O algoritmo que vai ser estudado toma partido do fato de estarmos em matemática modular, (também conhecida na escola como aritmética do relógio) ou seja ao lado da expressão há algum módulo finito.

```
longo função XELEVADOAY (longo X, Y, N)
   longo TMP
   se Y = 1 então
      devolva (X mod N) {em C ou Python: X % N}
   senão
      se (Y mod 2) = 0 então
         TMP ← XELEVADOAY (X, (Y/2), N)
         devolva (TMP * TMP) mod N
      senão
         TMP ← XELEVADOAY (X, ((Y-1)/2), N)
         TMP ← (TMP * TMP) mod N
         TMP ← (TMP * X) mod N
         devolva TMP
      fim se
   fim se
fim função
```

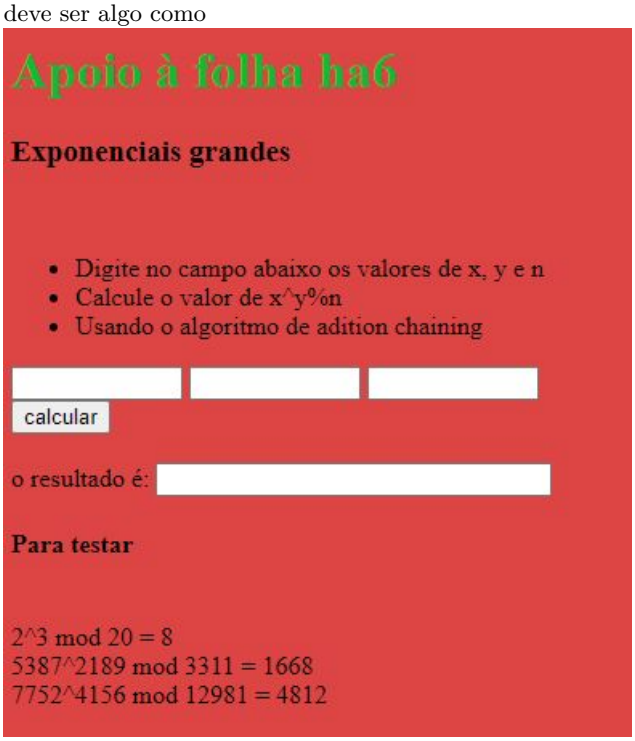
Depois de digitar, use os seguintes exemplos para testar o programa
$(2^3)$ *mod* 20 = 8
$(5387^{2189})$ *mod* 3311 = 1668
$(7752^{4156})$ *mod* 12981 = 4812

## Código

Nesta folha não há código, você deve desenvolvê-lo.

## A tela

deve ser algo como




## ☞ Para você fazer

Primeiro, mostre para o professor sua implementação

A seguir, Ache $19841^{13173}$ *mod* 15712
Responda aqui:

### Algoritmo Adition Chaining

Descrito no excelente livro *Applied Cryptography* de Bruce Schneier de 1996, (pág. 244) ele permite fazer exponenciações enormes como por exemplo $546789^{23543}$ = ..... . cuja resposta não aparece aqui, mas certamente é um número além de qualquer capacidade de representação usual. Obviamente os algoritmos usuais do ambiente computacional são claramente inadequados (incapazes) de obter este número.

O algoritmo que vai ser estudado toma partido do fato de estarmos em matemática modular, (também conhecida na escola como aritmética do relógio) ou seja ao lado da expressão há algum módulo finito.

```
longo função XELEVADOAY (longo X, Y, N)
   longo TMP
   se Y = 1 então
      devolva (X mod N) {em C ou Python: X % N}
   senão
      se (Y mod 2) = 0 então
         TMP ← XELEVADOAY (X, (Y/2), N)
         devolva (TMP * TMP) mod N
      senão
         TMP ← XELEVADOAY (X, ((Y-1)/2), N)
         TMP ← (TMP * TMP) mod N
         TMP ← (TMP * X) mod N
         devolva TMP
      fim se
   fim se
fim função
```

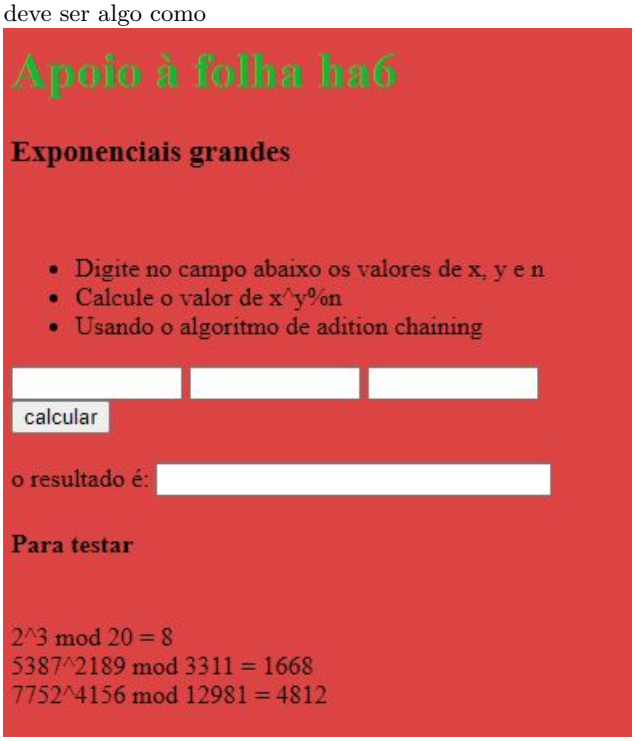
Depois de digitar, use os seguintes exemplos para testar o programa
$(2^3)$ *mod* 20 = 8
$(5387^{2189})$ *mod* 3311 = 1668
$(7752^{4156})$ *mod* 12981 = 4812

### Código

Nesta folha não há código, você deve desenvolvê-lo.

### A tela

deve ser algo como




### ☞ Para você fazer

Primeiro, mostre para o professor sua implementação

A seguir, Ache $12704^{13511}$ *mod* 15248
Responda aqui:

### Algoritmo Adition Chaining

Descrito no excelente livro *Applied Cryptography* de Bruce Schneier de 1996, (pág. 244) ele permite fazer exponenciações enormes como por exemplo $546789^{23543}$ = ..... . cuja resposta não aparece aqui, mas certamente é um número além de qualquer capacidade de representação usual. Obviamente os algoritmos usuais do ambiente computacional são claramente inadequados (incapazes) de obter este número.

O algoritmo que vai ser estudado toma partido do fato de estarmos em matemática modular, (também conhecida na escola como aritmética do relógio) ou seja ao lado da expressão há algum módulo finito.

```
longo função XELEVADOAY (longo X, Y, N)
   longo TMP
   se Y = 1 então
      devolva (X mod N) {em C ou Python: X % N}
   senão
      se (Y mod 2) = 0 então
         TMP ← XELEVADOAY (X, (Y/2), N)
         devolva (TMP * TMP) mod N
      senão
         TMP ← XELEVADOAY (X, ((Y-1)/2), N)
         TMP ← (TMP * TMP) mod N
         TMP ← (TMP * X) mod N
         devolva TMP
      fim se
   fim se
fim função
```

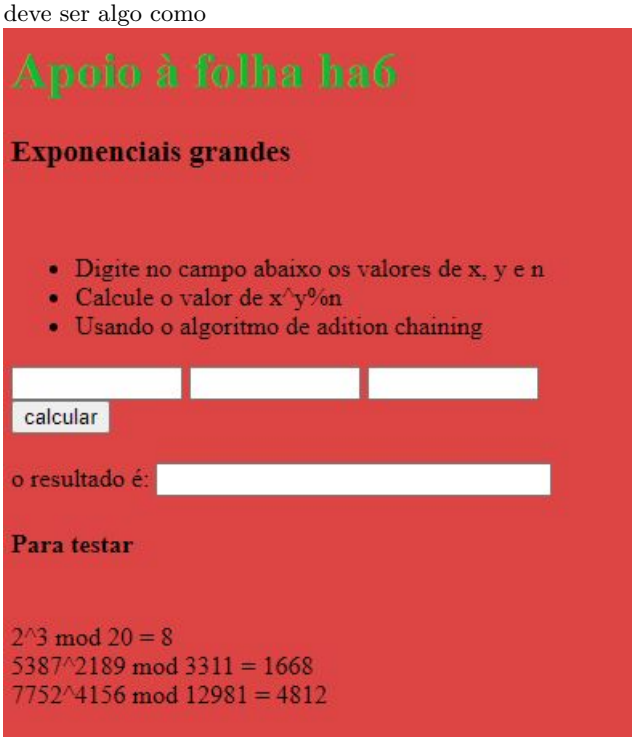
Depois de digitar, use os seguintes exemplos para testar o programa
$(2^3)$ *mod* 20 = 8
$(5387^{2189})$ *mod* 3311 = 1668
$(7752^{4156})$ *mod* 12981 = 4812

### Código

Nesta folha não há código, você deve desenvolvê-lo.

### A tela

deve ser algo como




### ☞ Para você fazer

Primeiro, mostre para o professor sua implementação

A seguir, Ache $14256^{15634}$ *mod* 17932
Responda aqui:

## Algoritmo Adition Chaining

Descrito no excelente livro *Applied Cryptography* de Bruce Schneier de 1996, (pág. 244) ele permite fazer exponenciações enormes como por exemplo $546789^{23543}$ = ..... . cuja resposta não aparece aqui, mas certamente é um número além de qualquer capacidade de representação usual. Obviamente os algoritmos usuais do ambiente computacional são claramente inadequados (incapazes) de obter este número.

O algoritmo que vai ser estudado toma partido do fato de estarmos em matemática modular, (também conhecida na escola como aritmética do relógio) ou seja ao lado da expressão há algum módulo finito.

```
longo função XELEVADOAY (longo X, Y, N)
   longo TMP
   se Y = 1 então
      devolva (X mod N) {em C ou Python: X % N}
   senão
      se (Y mod 2) = 0 então
         TMP ← XELEVADOAY (X, (Y/2), N)
         devolva (TMP * TMP) mod N
      senão
         TMP ← XELEVADOAY (X, ((Y-1)/2), N)
         TMP ← (TMP * TMP) mod N
         TMP ← (TMP * X) mod N
         devolva TMP
      fim se
   fim se
fim função
```

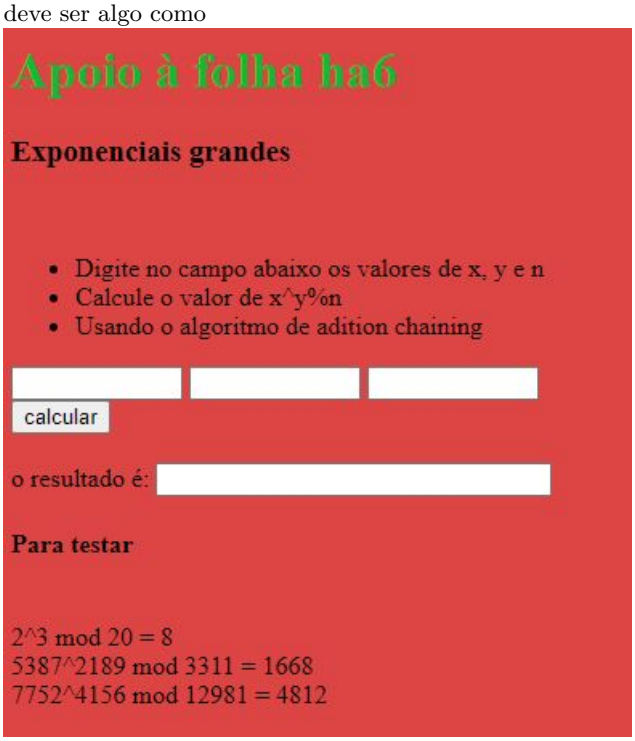
Depois de digitar, use os seguintes exemplos para testar o programa
$(2^3)$ *mod* 20 = 8
$(5387^{2189})$ *mod* 3311 = 1668
$(7752^{4156})$ *mod* 12981 = 4812

## Código

Nesta folha não há código, você deve desenvolvê-lo.

## A tela

deve ser algo como




## ☞ Para você fazer

Primeiro, mostre para o professor sua implementação

A seguir, Ache $18783^{12635}$ *mod* 13829
Responda aqui:

### Algoritmo Adition Chaining

Descrito no excelente livro *Applied Cryptography* de Bruce Schneier de 1996, (pág. 244) ele permite fazer exponenciações enormes como por exemplo $546789^{23543}$ = ..... . cuja resposta não aparece aqui, mas certamente é um número além de qualquer capacidade de representação usual. Obviamente os algoritmos usuais do ambiente computacional são claramente inadequados (incapazes) de obter este número.

O algoritmo que vai ser estudado toma partido do fato de estarmos em matemática modular, (também conhecida na escola como aritmética do relógio) ou seja ao lado da expressão há algum módulo finito.

```
longo função XELEVADOAY (longo X, Y, N)
   longo TMP
   se Y = 1 então
      devolva (X mod N) {em C ou Python: X % N}
   senão
      se (Y mod 2) = 0 então
         TMP ← XELEVADOAY (X, (Y/2), N)
         devolva (TMP * TMP) mod N
      senão
         TMP ← XELEVADOAY (X, ((Y-1)/2), N)
         TMP ← (TMP * TMP) mod N
         TMP ← (TMP * X) mod N
         devolva TMP
      fim se
   fim se
fim função
```

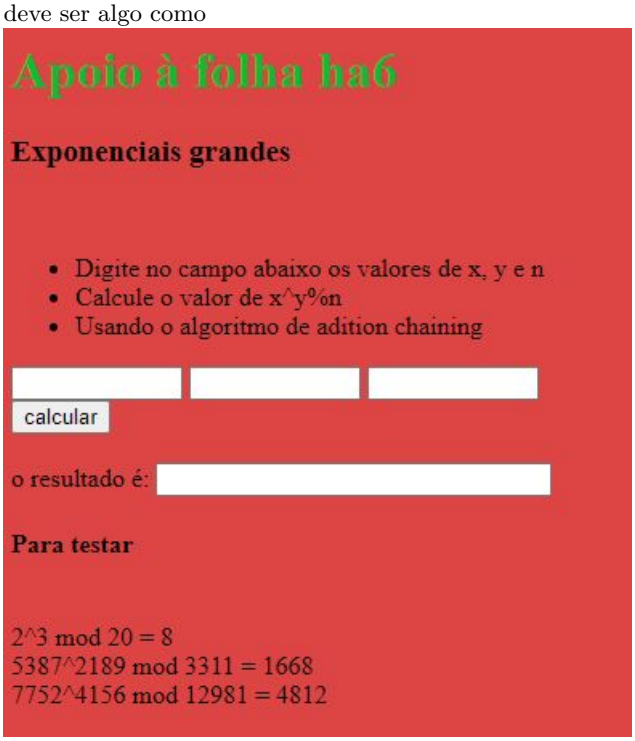
Depois de digitar, use os seguintes exemplos para testar o programa
$(2^3)$ *mod* 20 = 8
$(5387^{2189})$ *mod* 3311 = 1668
$(7752^{4156})$ *mod* 12981 = 4812

### Código

Nesta folha não há código, você deve desenvolvê-lo.

### A tela

deve ser algo como




### ☞ Para você fazer

Primeiro, mostre para o professor sua implementação

A seguir, Ache $17530^{13165}$ *mod* 15470
Responda aqui:

### Algoritmo Adition Chaining

Descrito no excelente livro *Applied Cryptography* de Bruce Schneier de 1996, (pág. 244) ele permite fazer exponenciações enormes como por exemplo $546789^{23543} =$ ..... . cuja resposta não aparece aqui, mas certamente é um número além de qualquer capacidade de representação usual. Obviamente os algoritmos usuais do ambiente computacional são claramente inadequados (incapazes) de obter este número.

O algoritmo que vai ser estudado toma partido do fato de estarmos em matemática modular, (também conhecida na escola como aritmética do relógio) ou seja ao lado da expressão há algum módulo finito.

```
longo função XELEVADOAY (longo X, Y, N)
   longo TMP
   se Y = 1 então
      devolva (X mod N) {em C ou Python: X % N}
   senão
      se (Y mod 2) = 0 então
         TMP ← XELEVADOAY (X, (Y/2), N)
         devolva (TMP * TMP) mod N
      senão
         TMP ← XELEVADOAY (X, ((Y-1)/2), N)
         TMP ← (TMP * TMP) mod N
         TMP ← (TMP * X) mod N
         devolva TMP
      fim se
   fim se
fim função
```

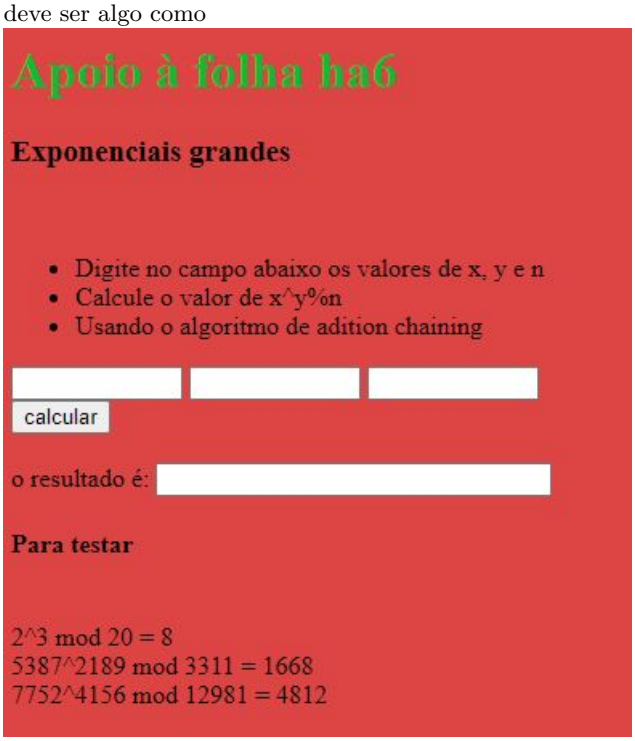
Depois de digitar, use os seguintes exemplos para testar o programa
$(2^3)$ *mod* $20 = 8$
$(5387^{2189})$ *mod* $3311 = 1668$
$(7752^{4156})$ *mod* $12981 = 4812$

### Código

Nesta folha não há código, você deve desenvolvê-lo.

### A tela

deve ser algo como




### ☞ Para você fazer

Primeiro, mostre para o professor sua implementação

A seguir, Ache $13924^{16182}$ *mod* 17669
Responda aqui:

## Algoritmo Adition Chaining

Descrito no excelente livro *Applied Cryptography* de Bruce Schneier de 1996, (pág. 244) ele permite fazer exponenciações enormes como por exemplo $546789^{23543}$ = ..... . cuja resposta não aparece aqui, mas certamente é um número além de qualquer capacidade de representação usual. Obviamente os algoritmos usuais do ambiente computacional são claramente inadequados (incapazes) de obter este número.

O algoritmo que vai ser estudado toma partido do fato de estarmos em matemática modular, (também conhecida na escola como aritmética do relógio) ou seja ao lado da expressão há algum módulo finito.

```
longo função XELEVADOAY (longo X, Y, N)
   longo TMP
   se Y = 1 então
      devolva (X mod N) {em C ou Python: X % N}
   senão
      se (Y mod 2) = 0 então
         TMP ← XELEVADOAY (X, (Y/2), N)
         devolva (TMP * TMP) mod N
      senão
         TMP ← XELEVADOAY (X, ((Y-1)/2), N)
         TMP ← (TMP * TMP) mod N
         TMP ← (TMP * X) mod N
         devolva TMP
      fim se
   fim se
fim função
```

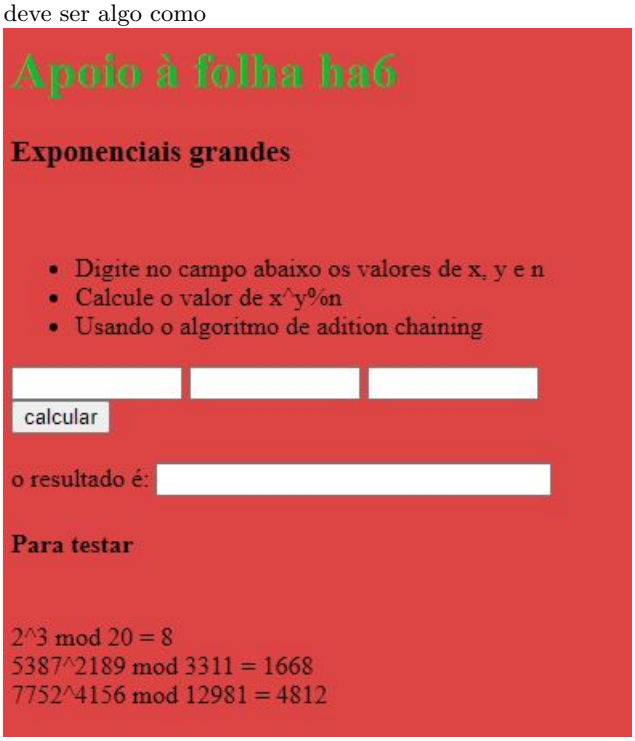
Depois de digitar, use os seguintes exemplos para testar o programa
$(2^3)$ *mod* 20 = 8
$(5387^{2189})$ *mod* 3311 = 1668
$(7752^{4156})$ *mod* 12981 = 4812

## Código

Nesta folha não há código, você deve desenvolvê-lo.

## A tela

deve ser algo como




## ☞ Para você fazer

Primeiro, mostre para o professor sua implementação

A seguir, Ache $14852^{14079}$ *mod* 10750
Responda aqui:

### Algoritmo Adition Chaining

Descrito no excelente livro *Applied Cryptography* de Bruce Schneier de 1996, (pág. 244) ele permite fazer exponenciações enormes como por exemplo $546789^{23543}$ = ..... . cuja resposta não aparece aqui, mas certamente é um número além de qualquer capacidade de representação usual. Obviamente os algoritmos usuais do ambiente computacional são claramente inadequados (incapazes) de obter este número.

O algoritmo que vai ser estudado toma partido do fato de estarmos em matemática modular, (também conhecida na escola como aritmética do relógio) ou seja ao lado da expressão há algum módulo finito.

```
longo função XELEVADOAY (longo X, Y, N)
   longo TMP
   se Y = 1 então
      devolva (X mod N) {em C ou Python: X % N}
   senão
      se (Y mod 2) = 0 então
         TMP ← XELEVADOAY (X, (Y/2), N)
         devolva (TMP * TMP) mod N
      senão
         TMP ← XELEVADOAY (X, ((Y-1)/2), N)
         TMP ← (TMP * TMP) mod N
         TMP ← (TMP * X) mod N
         devolva TMP
      fim se
   fim se
fim função
```

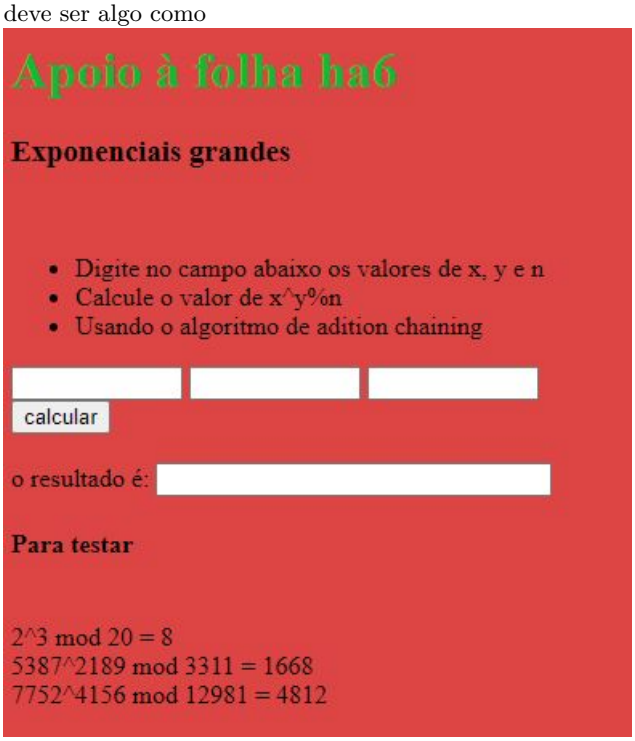
Depois de digitar, use os seguintes exemplos para testar o programa
$(2^3)$ *mod* 20 = 8
$(5387^{2189})$ *mod* 3311 = 1668
$(7752^{4156})$ *mod* 12981 = 4812

### Código

Nesta folha não há código, você deve desenvolvê-lo.

### A tela

deve ser algo como




### ☞ Para você fazer

Primeiro, mostre para o professor sua implementação

A seguir, Ache $14035^{12410}$ *mod* 18357
Responda aqui:

### Algoritmo Adition Chaining

Descrito no excelente livro *Applied Cryptography* de Bruce Schneier de 1996, (pág. 244) ele permite fazer exponenciações enormes como por exemplo $546789^{23543}$ = ..... . cuja resposta não aparece aqui, mas certamente é um número além de qualquer capacidade de representação usual. Obviamente os algoritmos usuais do ambiente computacional são claramente inadequados (incapazes) de obter este número.

O algoritmo que vai ser estudado toma partido do fato de estarmos em matemática modular, (também conhecida na escola como aritmética do relógio) ou seja ao lado da expressão há algum módulo finito.

```
longo função XELEVADOAY (longo X, Y, N)
   longo TMP
   se Y = 1 então
      devolva (X mod N) {em C ou Python: X % N}
   senão
      se (Y mod 2) = 0 então
         TMP ← XELEVADOAY (X, (Y/2), N)
         devolva (TMP * TMP) mod N
      senão
         TMP ← XELEVADOAY (X, ((Y-1)/2), N)
         TMP ← (TMP * TMP) mod N
         TMP ← (TMP * X) mod N
         devolva TMP
      fim se
   fim se
fim função
```

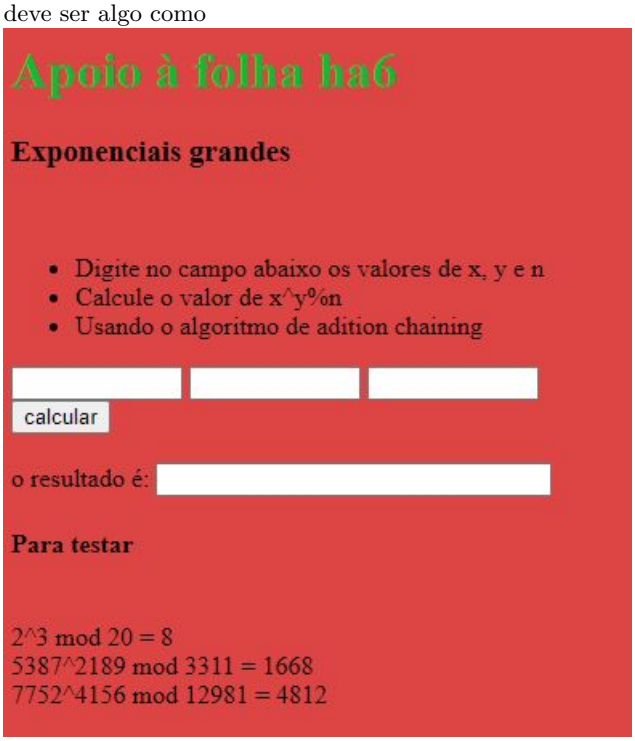
Depois de digitar, use os seguintes exemplos para testar o programa
$(2^3)$ *mod* 20 = 8
$(5387^{2189})$ *mod* 3311 = 1668
$(7752^{4156})$ *mod* 12981 = 4812

### Código

Nesta folha não há código, você deve desenvolvê-lo.

### A tela

deve ser algo como




### ☞ Para você fazer

Primeiro, mostre para o professor sua implementação

A seguir, Ache $15454^{19252}$ *mod* 14337
Responda aqui:

### Algoritmo Adition Chaining

Descrito no excelente livro *Applied Cryptography* de Bruce Schneier de 1996, (pág. 244) ele permite fazer exponenciações enormes como por exemplo $546789^{23543} =$ ..... . cuja resposta não aparece aqui, mas certamente é um número além de qualquer capacidade de representação usual. Obviamente os algoritmos usuais do ambiente computacional são claramente inadequados (incapazes) de obter este número.

O algoritmo que vai ser estudado toma partido do fato de estarmos em matemática modular, (também conhecida na escola como aritmética do relógio) ou seja ao lado da expressão há algum módulo finito.

```
longo função XELEVADOAY (longo X, Y, N)
   longo TMP
   se Y = 1 então
      devolva (X mod N) {em C ou Python: X % N}
   senão
      se (Y mod 2) = 0 então
         TMP ← XELEVADOAY (X, (Y/2), N)
         devolva (TMP * TMP) mod N
      senão
         TMP ← XELEVADOAY (X, ((Y-1)/2), N)
         TMP ← (TMP * TMP) mod N
         TMP ← (TMP * X) mod N
         devolva TMP
      fim se
   fim se
fim função
```

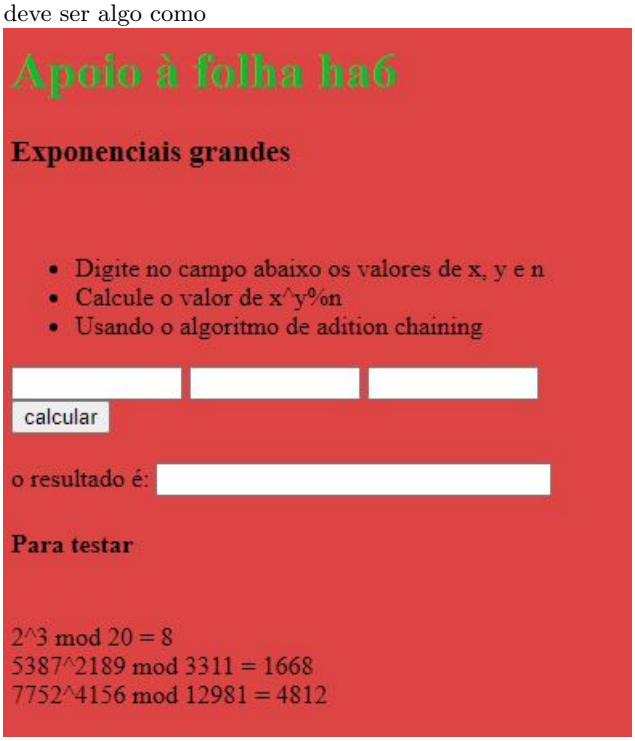
Depois de digitar, use os seguintes exemplos para testar o programa
$(2^3)$ *mod* $20 = 8$
$(5387^{2189})$ *mod* $3311 = 1668$
$(7752^{4156})$ *mod* $12981 = 4812$

### Código

Nesta folha não há código, você deve desenvolvê-lo.

### A tela

deve ser algo como




### ☞ Para você fazer

Primeiro, mostre para o professor sua implementação

A seguir, Ache $12818^{11797}$ *mod* 16935
Responda aqui:

### Algoritmo Adition Chaining

Descrito no excelente livro *Applied Cryptography* de Bruce Schneier de 1996, (pág. 244) ele permite fazer exponenciações enormes como por exemplo $546789^{23543} =$ ..... . cuja resposta não aparece aqui, mas certamente é um número além de qualquer capacidade de representação usual. Obviamente os algoritmos usuais do ambiente computacional são claramente inadequados (incapazes) de obter este número.

O algoritmo que vai ser estudado toma partido do fato de estarmos em matemática modular, (também conhecida na escola como aritmética do relógio) ou seja ao lado da expressão há algum módulo finito.

```
longo função XELEVADOAY (longo X, Y, N)
   longo TMP
   se Y = 1 então
      devolva (X mod N) {em C ou Python: X % N}
   senão
      se (Y mod 2) = 0 então
         TMP ← XELEVADOAY (X, (Y/2), N)
         devolva (TMP * TMP) mod N
      senão
         TMP ← XELEVADOAY (X, ((Y-1)/2), N)
         TMP ← (TMP * TMP) mod N
         TMP ← (TMP * X) mod N
         devolva TMP
      fim se
   fim se
fim função
```

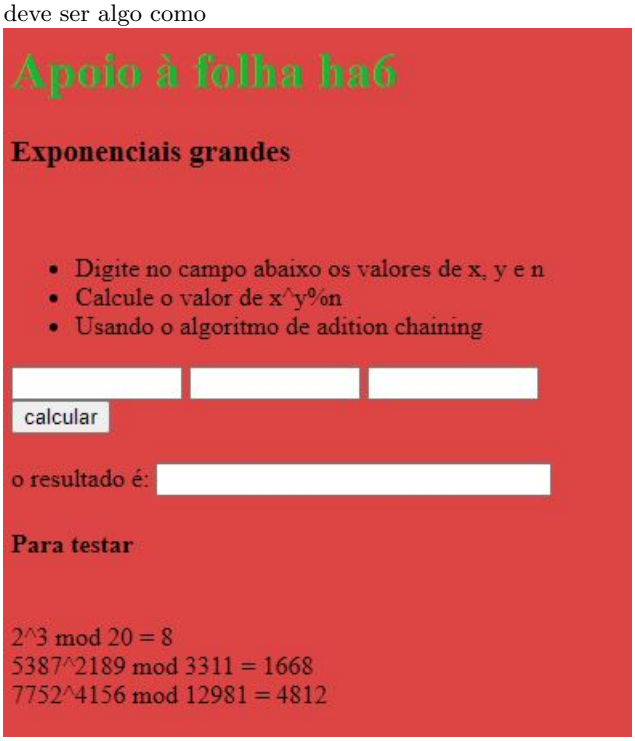
Depois de digitar, use os seguintes exemplos para testar o programa
$(2^3)$ *mod* $20 = 8$
$(5387^{2189})$ *mod* $3311 = 1668$
$(7752^{4156})$ *mod* $12981 = 4812$

### Código

Nesta folha não há código, você deve desenvolvê-lo.

### A tela

deve ser algo como




### ☞ Para você fazer

Primeiro, mostre para o professor sua implementação

A seguir, Ache $11923^{19510}$ *mod* 17362
Responda aqui:

### Algoritmo Adition Chaining

Descrito no excelente livro *Applied Cryptography* de Bruce Schneier de 1996, (pág. 244) ele permite fazer exponenciações enormes como por exemplo $546789^{23543} =$ ..... . cuja resposta não aparece aqui, mas certamente é um número além de qualquer capacidade de representação usual. Obviamente os algoritmos usuais do ambiente computacional são claramente inadequados (incapazes) de obter este número.

O algoritmo que vai ser estudado toma partido do fato de estarmos em matemática modular, (também conhecida na escola como aritmética do relógio) ou seja ao lado da expressão há algum módulo finito.

```
longo função XELEVADOAY (longo X, Y, N)
   longo TMP
   se Y = 1 então
      devolva (X mod N) {em C ou Python: X % N}
   senão
      se (Y mod 2) = 0 então
         TMP ← XELEVADOAY (X, (Y/2), N)
         devolva (TMP * TMP) mod N
      senão
         TMP ← XELEVADOAY (X, ((Y-1)/2), N)
         TMP ← (TMP * TMP) mod N
         TMP ← (TMP * X) mod N
         devolva TMP
      fim se
   fim se
fim função
```

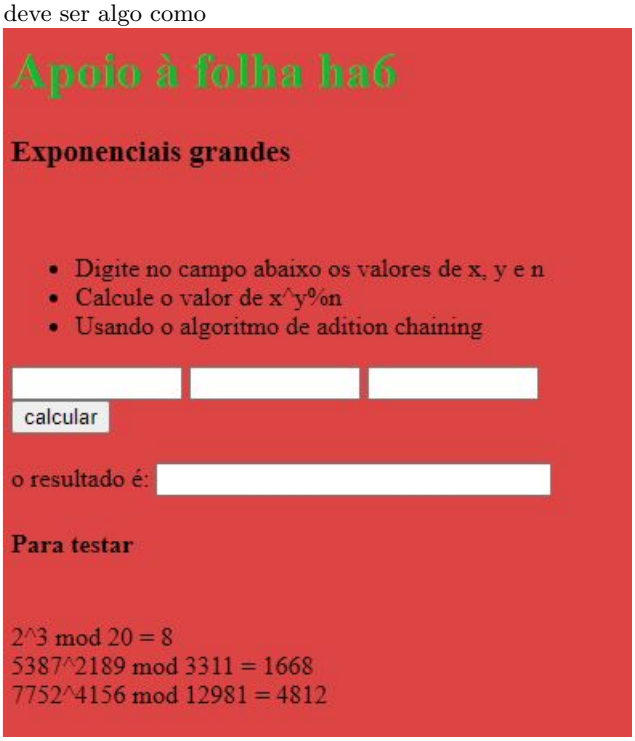
Depois de digitar, use os seguintes exemplos para testar o programa
$(2^3)$ *mod* $20 = 8$
$(5387^{2189})$ *mod* $3311 = 1668$
$(7752^{4156})$ *mod* $12981 = 4812$

### Código

Nesta folha não há código, você deve desenvolvê-lo.

### A tela

deve ser algo como




### ☞ Para você fazer

Primeiro, mostre para o professor sua implementação

A seguir, Ache $11297^{19485}$ *mod* 19447
Responda aqui:

## Algoritmo Adition Chaining

Descrito no excelente livro *Applied Cryptography* de Bruce Schneier de 1996, (pág. 244) ele permite fazer exponenciações enormes como por exemplo $546789^{23543} =$ ..... . cuja resposta não aparece aqui, mas certamente é um número além de qualquer capacidade de representação usual. Obviamente os algoritmos usuais do ambiente computacional são claramente inadequados (incapazes) de obter este número.

O algoritmo que vai ser estudado toma partido do fato de estarmos em matemática modular, (também conhecida na escola como aritmética do relógio) ou seja ao lado da expressão há algum módulo finito.

```
longo função XELEVADOAY (longo X, Y, N)
   longo TMP
   se Y = 1 então
      devolva (X mod N) {em C ou Python: X % N}
   senão
      se (Y mod 2) = 0 então
         TMP ← XELEVADOAY (X, (Y/2), N)
         devolva (TMP * TMP) mod N
      senão
         TMP ← XELEVADOAY (X, ((Y-1)/2), N)
         TMP ← (TMP * TMP) mod N
         TMP ← (TMP * X) mod N
         devolva TMP
      fim se
   fim se
fim função
```

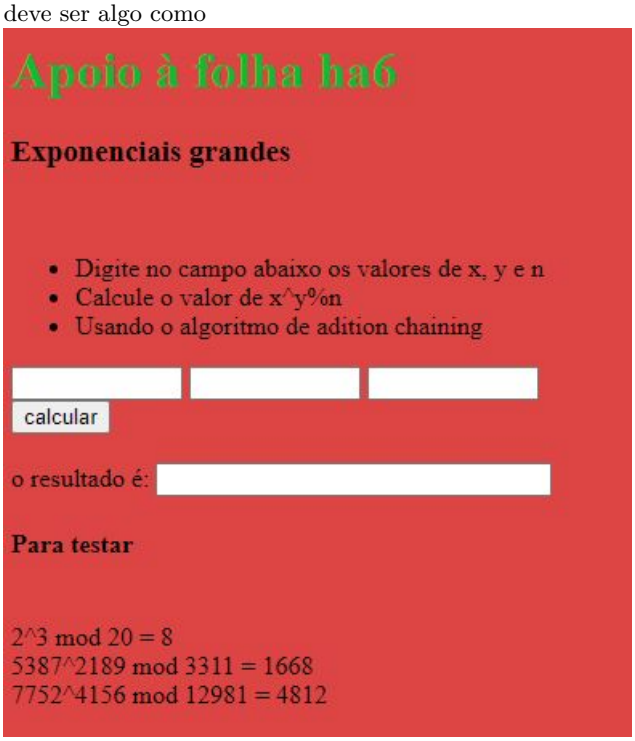
Depois de digitar, use os seguintes exemplos para testar o programa
$(2^3)$ *mod* $20 = 8$
$(5387^{2189})$ *mod* $3311 = 1668$
$(7752^{4156})$ *mod* $12981 = 4812$

## Código

Nesta folha não há código, você deve desenvolvê-lo.

## A tela

deve ser algo como




## ☞ Para você fazer

Primeiro, mostre para o professor sua implementação

A seguir, Ache $14093^{14660}$ *mod* 15209
Responda aqui: