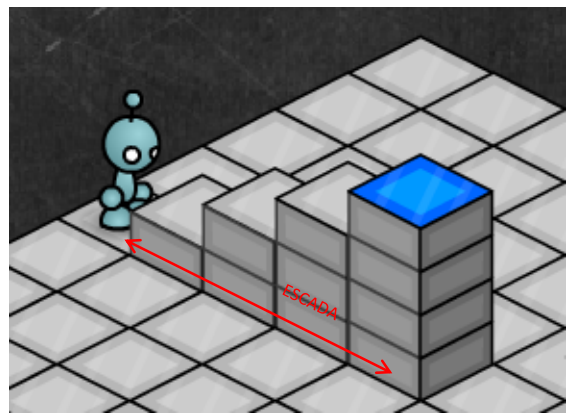
Recursão

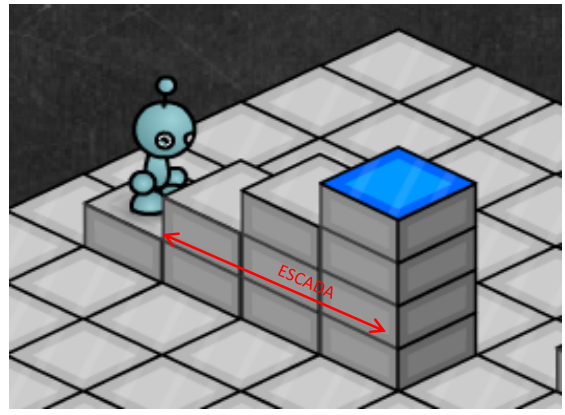
Subindo uma escada

Ação: Subir 1 degrau da escada



Subindo uma escada

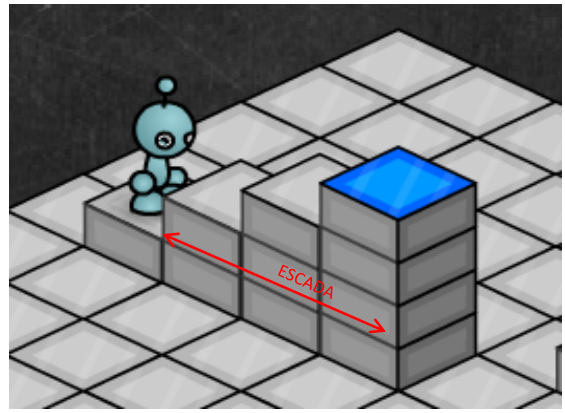
Ação: Subir 1 degrau da escada



3

Subindo uma escada

Ação: **E agora???**



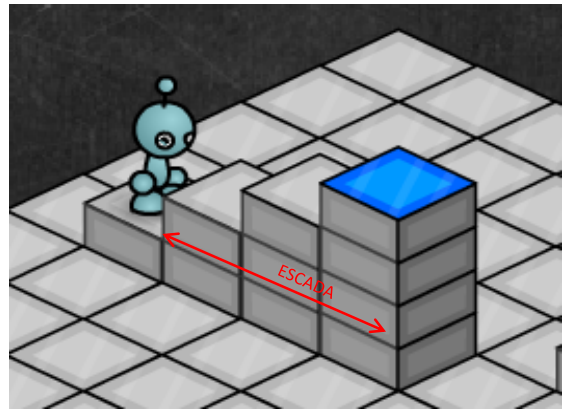
4

Subindo uma escada

Ação:

E agora???

Tem uma escada....

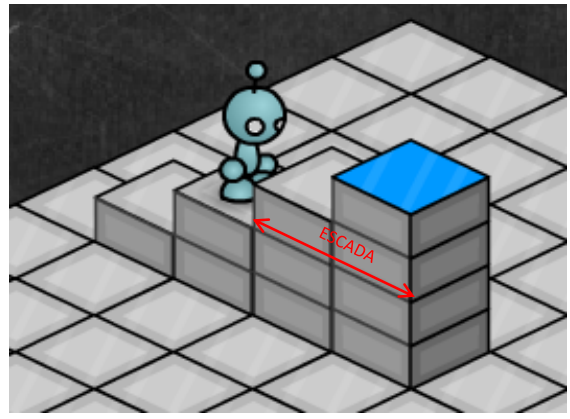


5

Subindo uma escada

Ação:

Subir 1 degrau da escada

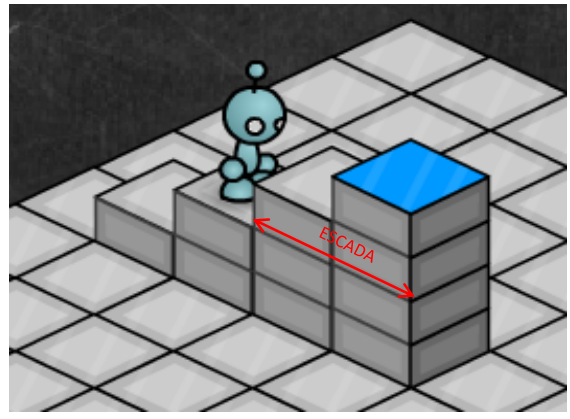


6

Subindo uma escada

Ação:

E agora???



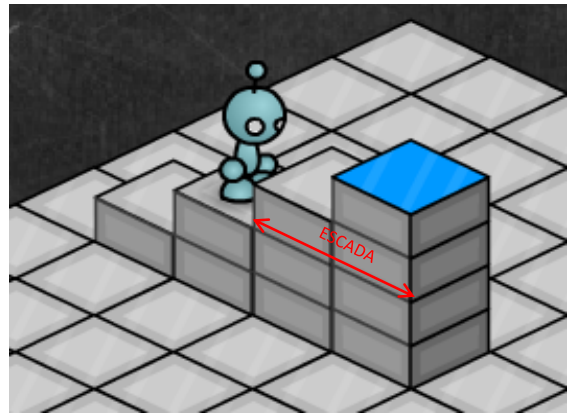
7

Subindo uma escada

Ação:

E agora???

Tem uma escada.... → Subir 1 degrau da escada



8

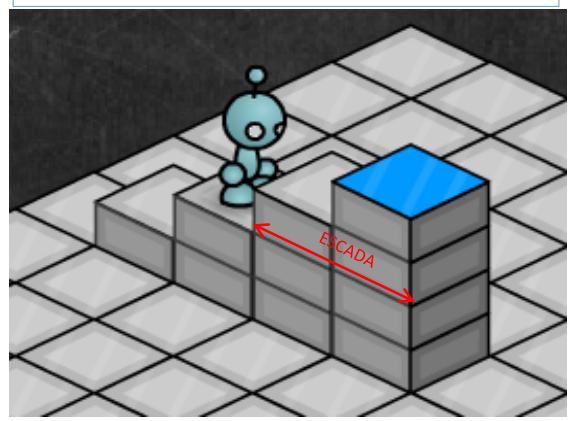
Subindo uma escada

Ação:

E agora???

Tem uma escada.... → Subir 1 degrau da escada

Que sequência de ações o *bot* está realizando?

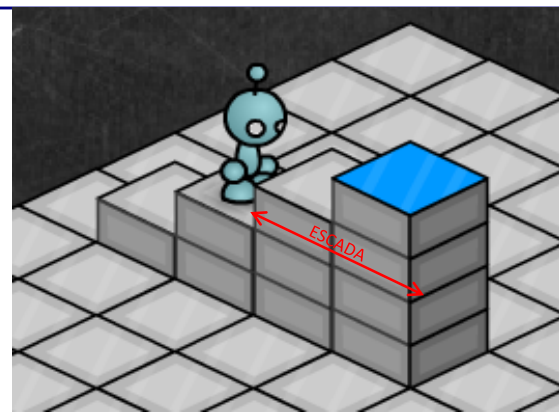


9

Subindo uma escada

Que sequência de ações o *bot* está realizando?

Subir 1 degrau
Subir escada restante (de *n*-1 degraus)

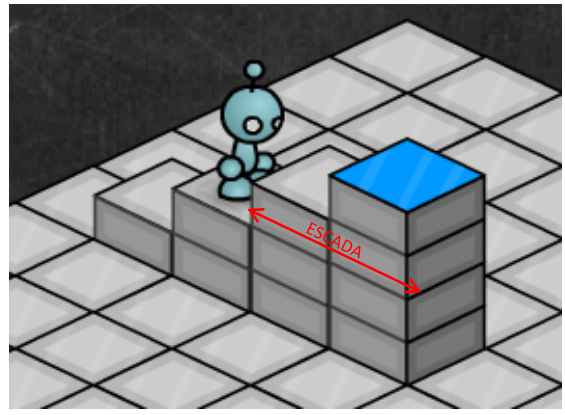


10

Subindo uma escada

Subir uma escada de *n* degraus é:

Subir 1 degrau
Subir uma escada de *n*-1 degraus

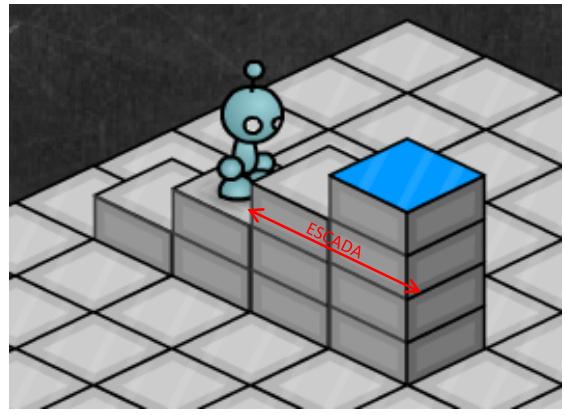


11

Subindo uma escada

Subir uma escada de *n* degraus é:

Subir 1 degrau ⇦
Subir uma escada de *n*-1 degraus

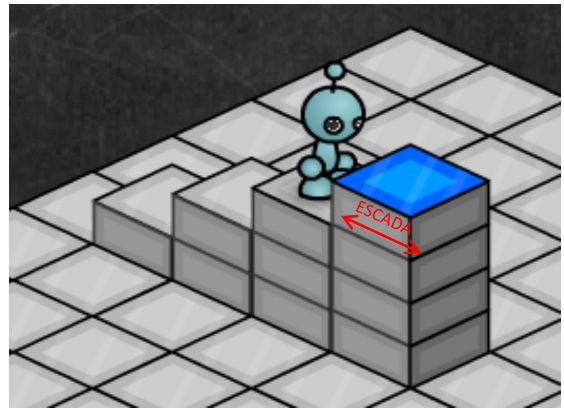


12

Subindo uma escada

Subir uma escada de *n* degraus é:

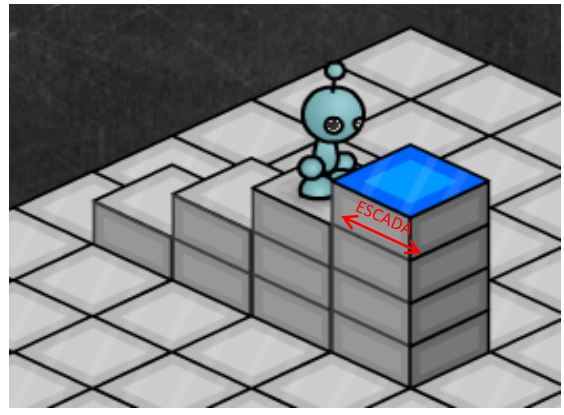
Subir 1 degrau ⇦
Subir uma escada de *n*-1 degraus



Subindo uma escada

Subir uma escada de *n* degraus é:

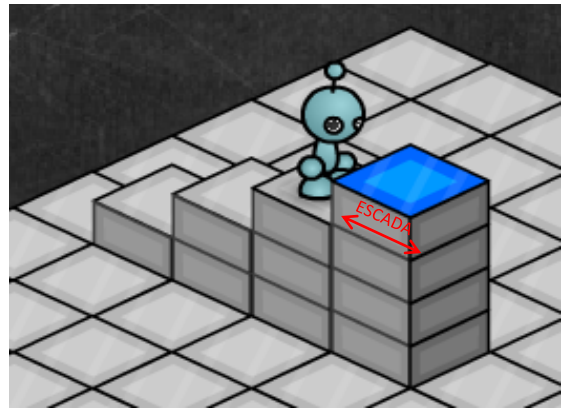
Subir 1 degrau
Subir uma escada de *n*-1 degraus ⇦



Subindo uma escada

Subir uma escada de *n* degraus é:

Subir 1 degrau ⇦
Subir uma escada de *n*-1 degraus

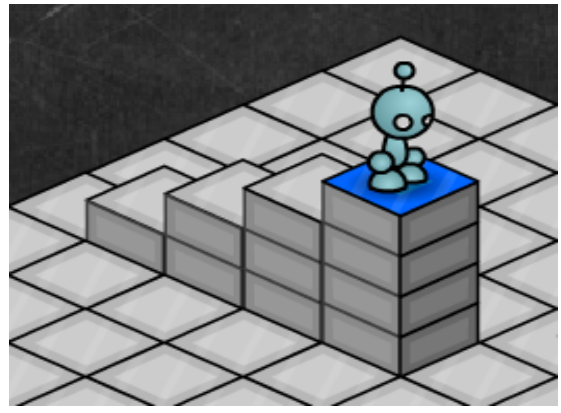


15

Subindo uma escada

Subir uma escada de *n* degraus é:

Subir 1 degrau ⇦
Subir uma escada de *n*-1 degraus

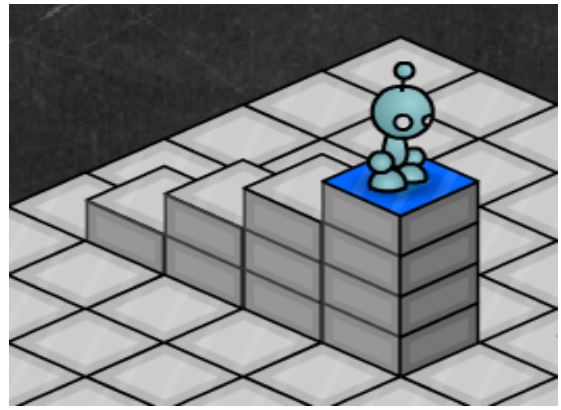


16

Subindo uma escada

Subir uma escada de *n* degraus é:

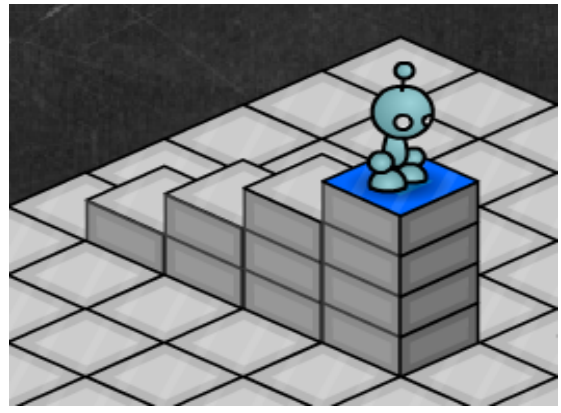
Subir 1 degrau
Subir uma escada de *n*-1 degraus ⇦



Subindo uma escada

Subir uma escada de *n* degraus é:

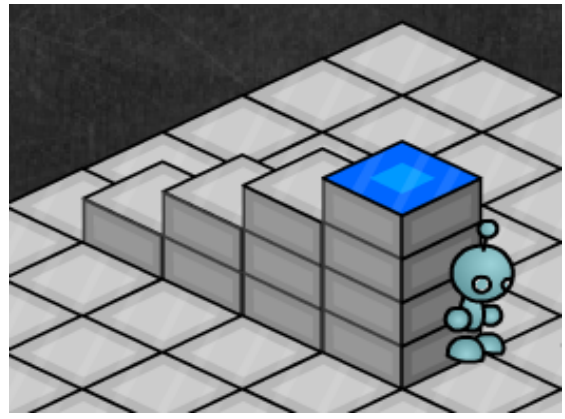
Subir 1 degrau ⇦
Subir uma escada de *n*-1 degraus



Subindo uma escada

Subir uma escada de *n* degraus é:

Subir 1 degrau ⇦
Subir uma escada de *n*-1 degraus



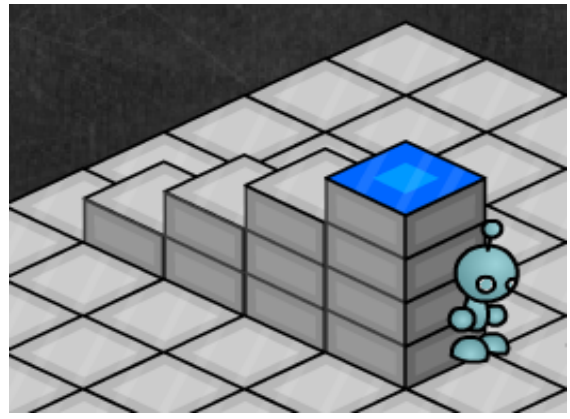
19

Subindo uma escada

Subir uma escada de *n* degraus é:

Subir 1 degrau ⇦
Subir uma escada de *n*-1 degraus

O que aconteceu?????
Por que ele caiu????



20

Subindo uma escada

Subir uma escada de *n* degraus é:

Subir 1 degrau
Subir uma escada de *n*-1 degraus

a ação é definida em termos de si mesma

Subir uma escada aparece como parte da definição de subir uma escada!!

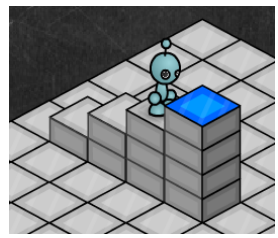
⇓

Forma um ciclo

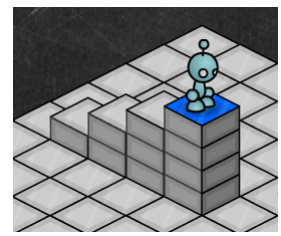
Quando interromper o ciclo para que o *bot* não caia?

Subindo uma escada

Quando interromper o ciclo para que o *bot* não caia?

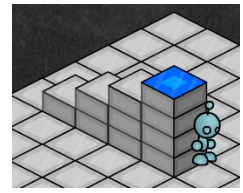


Escada com degraus→ deve subir



Escada sem degraus→ deve parar

Subindo uma escada



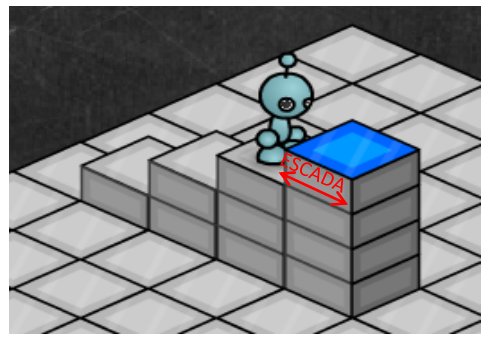
Subir uma escada de *n* degraus é:

Se não há degraus na escada (*n == 0)*
chegou no topo da escada/não há escada
Senão
Subir 1 degrau
Subir uma escada de *n*-1 degraus

Subindo uma escada

Subir uma escada de *n* degraus é:

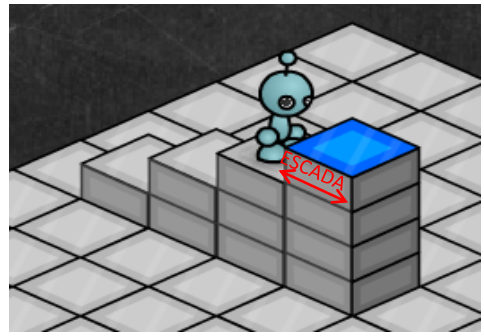
Se não há degraus na escada (*n == 0)*
chegou no topo da escada/não há escada
Senão
Subir 1 degrau
Subir uma escada de *n*-1 degraus



Subindo uma escada

Subir uma escada de *n* degraus é:

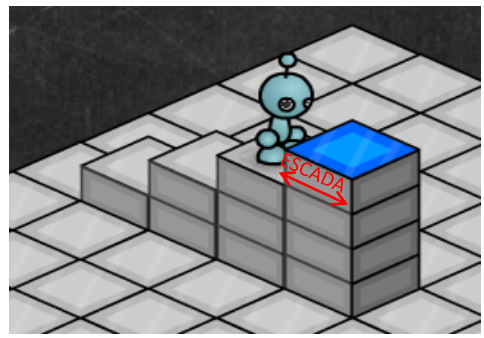
Se não há degraus na escada (*n == 0)* ⬅
    chegou no topo da escada/não há escada
Senão
    Subir 1 degrau
    Subir uma escada de *n*-1 degraus



Subindo uma escada

Subir uma escada de *n* degraus é:

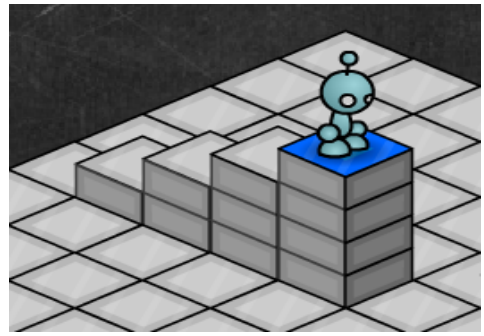
Se não há degraus na escada (*n == 0)*
    chegou no topo da escada/não há escada
Senão
    Subir 1 degrau ⬅
    Subir uma escada de *n*-1 degraus



Subindo uma escada

Subir uma escada de *n* degraus é:

Se não há degraus na escada (*n == 0)*
    chegou no topo da escada/não há escada
Senão
    Subir 1 degrau
    Subir uma escada de *n*-1 degraus

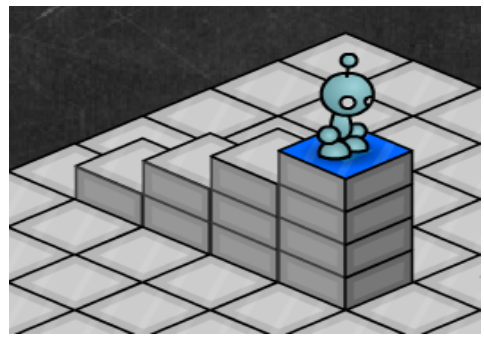


27

Subindo uma escada

Subir uma escada de *n* degraus é:

Se não há degraus na escada (*n == 0)*
    chegou no topo da escada/não há escada
Senão
    Subir 1 degrau
    Subir uma escada de *n*-1 degraus

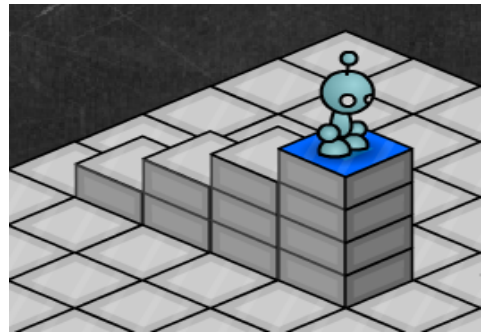


28

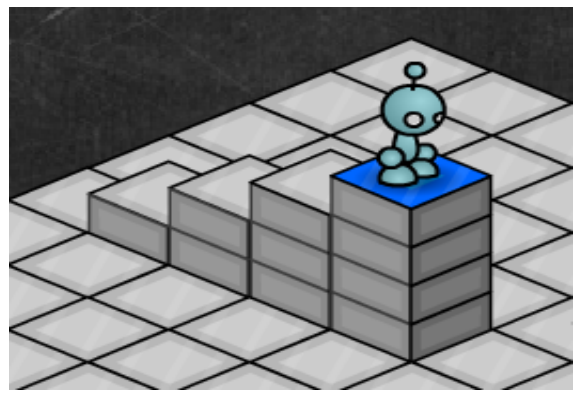
Subindo uma escada

Subir uma escada de *n* degraus é:

Se não há degraus na escada (*n == 0)*
    chegou no topo da escada/não há escada
Senão
    Subir 1 degrau
    Subir uma escada de *n*-1 degraus



Subindo uma escada



Recursividade

Um objeto é dito recursivo se pode ser definido em termos de si próprio:

"*o quê está sendo definido aparece como parte da definição*"

Para fazer iogurte, você precisa de leite e de um pouco de iogurte.

Para subir uma escada você precisa subir um degrau e depois o resto da escada

Em toda definição recursiva, existe um caso trivial (caso base), cuja solução é conhecida e interrompe a recorrência.

Sequências, funções e conjuntos podem ser definidos recursivamente.

Recursividade em conjuntos

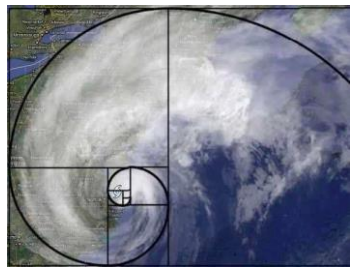
Exemplo: Conjunto dos números naturais (pelos axiomas de Peano)

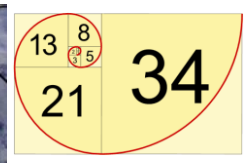
- 0 não é sucessor de nenhum número
- Todo número natural tem um único sucessor que também é um número natural

Observação: *Todo conjunto definido recursivamente tem cardinalidade infinita*

- ✓ 0
- ✓ 1 -- (suc(0))
- ✓ 2 -- (suc(suc(0))
- ✓ 3 -- (suc(suc(suc(0)))

Recursividade na natureza





33

Definições recursivas

Composta de duas partes:

(1) uma base, em que alguns casos simples do item que está sendo definido são dados explicitamente (ponto de partida)

(2) um passo indutivo ou recursivo, em que outros casos do item que está sendo definido são dados em termos de casos anteriores. (novos casos construídos a partir dos casos simples, que são base para construir outros casos e assim por diante)

34

Mãos na massa!

Encontre a definição recursiva de:

1. x*y
   Casos Triviais?
   Passos Recursivos?
2. x^y, com $y \in N$
   Casos Triviais?
   Passos Recursivos?
3. Quantidade de caracteres em uma string
   Casos Triviais?
   Passos Recursivos?
4. x!
   Casos Triviais?
   Passos Recursivos?
5. DESAFIO: x:y, com $x, y \in N$
   Casos Triviais?
   Passos Recursivos?
6. DESAFIO: x%y com $y \in N$
   Casos Triviais?
   Passos Recursivos?

Solução

Encontre a definição recursiva de:

1. x * y

   3 * 4 = 3 + (3*3)

   3 * 3 = 3 + (3*2)

   3 * 2 = 3 + (3*1)

   3 * 1 = 3 + (3*0)

   3 * 0 = 0

   Casos Triviais?

   Passo Recursivo?

Solução

1. x * y

   3 * 4 = 3 + (3*3)

   3 * 3 = 3 + (3*2)

   3 * 2 = 3 + (3*1)

   3 * 1 = 3 + (3*0)

   3 * 0 = 0

Caso Trivial? x*0 = 0

Passo Recursivo? x* y = x + x*(y-1)

$$x * y = \begin{cases} 0, y = 0 \\ x + x * (y-1), y > 0 \end{cases}$$

37

Solução

2. x^y

$3^4 = 3 * 3^3$

$3^3 = 3 * 3^2$

$3^2 = 3 * 3^1$

$3^1 = 3 * 3^0$

$3^0 = 1$

Caso Trivial? $x^0 = 1$

Passo Recursivo? $x^y = x * x^{(y-1)}$

$$x^y = \begin{cases} 1, y = 0 \\ x * x^{y-1}, y > 0 \end{cases}$$

38

Solução

3. Quantidade de caracteres em uma string

Mia = M + ia

ia = i + a

a = a + ''

Caso Trivial? string vazia →0

Passos Recursivos? 1 (do 1º caractere) + quantidade de caracteres na string subsequente

$$comprimento(str) = \begin{cases} 0, se\ str = '' \\ 1 + comprimento(\text{str}[1:]), \text{se str} \neq '' \end{cases}$$

Fatorial de um número

4. Fatorial de um número

n! = n * (n-1) !
(n-1)! = (n-1) * (n-2)!
(n-2)! = (n-2) * (n-3)!
...
(n-(n-1))! = (n-(n-1)) * (n-n)!
(n-n)! =1

$$n! = \begin{cases} 1, se\ n = 0 \\ n \times (n-1)!, se\ n > 0 \end{cases}$$

Esta relação de recorrência é infinita?

Fatorial de um número

$$n! = \begin{cases} 1, se\ n = 0 \\ n \times (n-1)!, se\ n > 0 \end{cases}$$

Caso base (ou trivial): solução conhecida .

↓

não envolve a recursão

↓

Interrompe a recorrência

Fatorial de um número

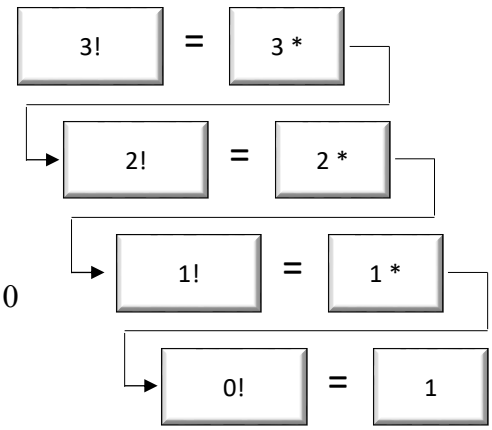
$$n! = \begin{cases} 1, se\ n = 0 \\ n \times (n-1)!, se\ n > 0 \end{cases}$$

Caso BASE

Passo Recursivo

Fatorial de 3

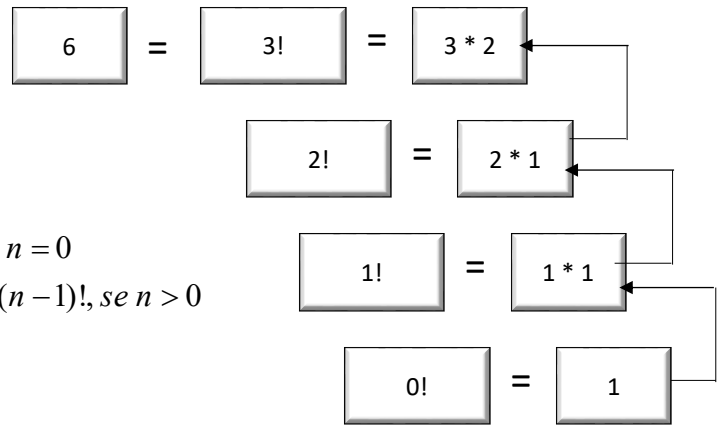
$$n! = \begin{cases} 1, se\ n = 0 \\ n \times (n-1)!, se\ n > 0 \end{cases}$$



43

Fatorial de 3

$$n! = \begin{cases} 1, se\ n = 0 \\ n \times (n-1)!, se\ n > 0 \end{cases}$$

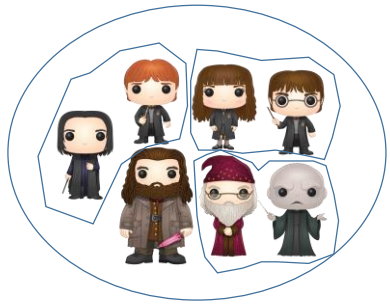


44

Solução

5. Quociente da divisão inteira x : y

Repartir x em grupos de y unidades. Quantas vezes o y cabe inteiro dentro do x?



Caso Trivial? x < y = 0
Passo Recursivo? x ≥ y 1 + (x-y):y

7:2 = 1+ (5:2)
5:2 = 1 + (3:2)
3:2 = 1+ (1:2)
1:2 = 0

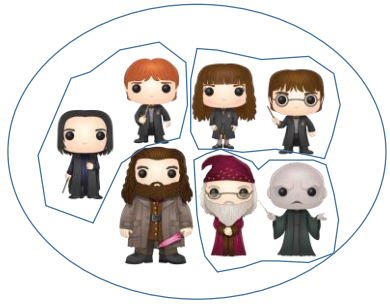
$$x:y = \begin{cases} 0, & x < y \\ 1 + (x-y):y, & x \geq y \end{cases}$$

45

Solução

6. Resto da divisão inteira x % y

O que sobra de x ao reparti-lo em grupos de y unidades.

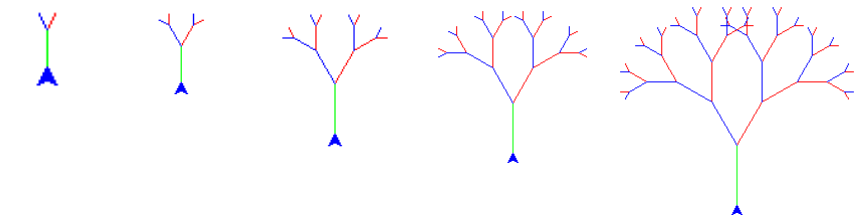


Caso Trivial? x < y = x
Passo Recursivo? x ≥ y, 0 + (x-y) % y

$$x \,\%\, y = \begin{cases} x, & x < y \\ (x-y) \,\%\, y, & x \geq y \end{cases}$$

46

Estruturas recursivas: árvore genealógica



Eu e pais

Eu, pais e avós

pais dos pais

Eu, pais, avós e bisavós

pais dos avós

Eu, pais, avós bisavós e tataravós

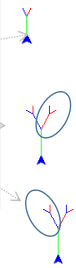
pais dos bisavós

Eu e meus ancestrais

47

Estruturas recursivas: árvore genealógica-principal

Se o nível de ancestral desejado foi atingido
    parar
Senão
    Desenhar galho do nível
    Desenhar árvore genealógica da mãe do nível (c/-30^0)
    Desenhar árvore genealógica do pai do nível (c/+30^0)

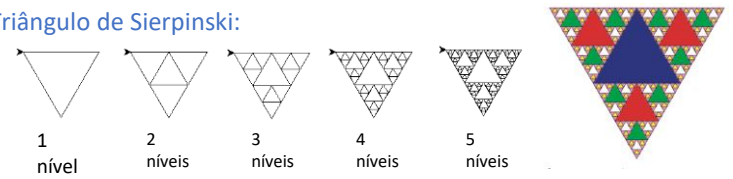


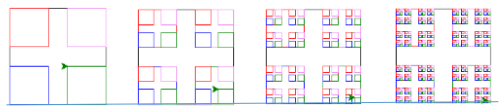
48

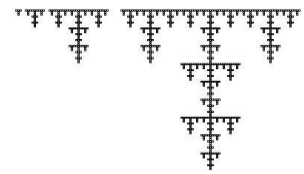
Figuras recursivas (fractais)

Fractais são objetos em que cada parte é semelhante ao objeto como um todo. O objeto é composto por partes reduzidas dele próprio.

Triângulo de Sierpinski:







49

Pensamento recursivo

Como somar estes números?

2

4

\+ 6

8

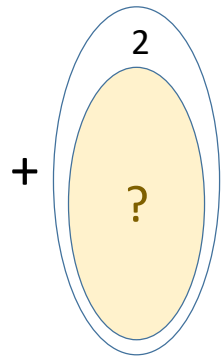
4

????

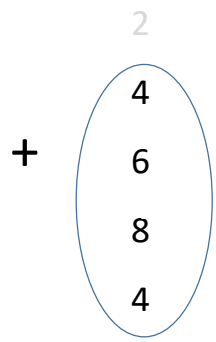
Quebrar o problema em subproblemas menores, de mesma natureza, até alcançar um problema simples o bastante para ser resolvido trivialmente

50

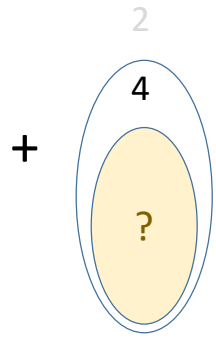
Pensamento recursivo



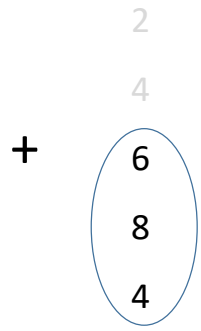
Pensamento recursivo



Pensamento recursivo



Pensamento recursivo



Pensamento recursivo

2

4

\+ 6

?

Pensamento recursivo

2

4

\+ 6

8

4

Pensamento recursivo

2

4

\+ 6

8

?

Pensamento recursivo

2

4

\+ 6

8

4 → 4

Pensamento recursivo

2

4

\+ 6

8

4

Pensamento recursivo

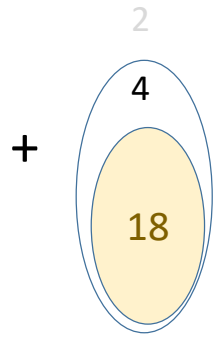
2

4

\+ 6

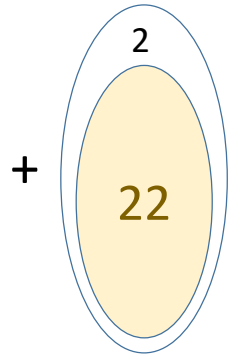
12

Pensamento recursivo



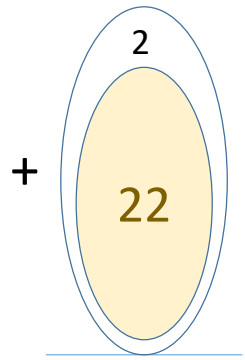
61

Pensamento recursivo



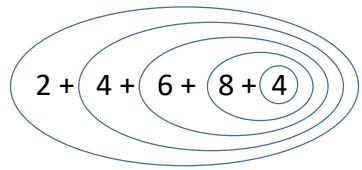
62

Pensamento recursivo



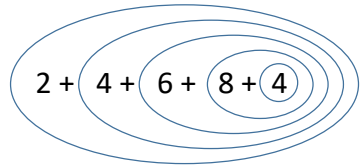
24

Pensamento recursivo



Qual foi o caso base? Só um número para somar
Qual o passo recursivo?
O que fazer com a solução retornada da chamada para o caso mais simples em cada instância?

Pensamento recursivo

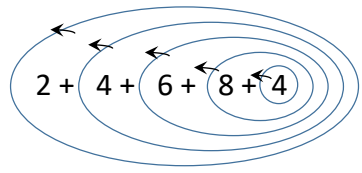


Qual foi o caso base? Só um número para somar
Qual o passo recursivo? Somar nºs restantes (2º ao último)
O que fazer com a solução retornada da chamada para o caso mais simples em cada instância?

65

Pensamento recursivo

24



Qual foi o caso base? Só um número para somar
Qual o passo recursivo? Somar nºs restantes (2º ao último)
O que fazer com a solução retornada da chamada para o caso mais simples em cada instância? Adicionar o 1º número ao total computado pela chamada recursiva

66

Processo recursivo para a solução de um problema

Divide os problemas em problemas menores de mesma natureza.

Natureza do problema	=	Natureza do subproblema
⇑		⇑
Método para solução do problema	↔	Método para solução do subproblema

Composto de duas partes:

- O caso trivial, cuja solução é conhecida e interrompe a recorrência.
- Um método geral, em que há chamada do método para um ou mais problemas menores (subproblemas) de mesma natureza.

67

Funções recursivas

Resolve o problema usando como método de solução a recursão.

É definida em termos de si mesma, a partir das relações de recorrência e do(s) caso(s) base(s).

Regras básicas - No corpo da função há:

- ✓ pelo menos uma condição de parada (testes dos casos bases)
- ✓ chamada de uma cópia de si mesma entregando como argumento um problema mais simples do que o recebido, convergindo para o caso base (passo recursivo)

problema

```
def  função ( [■■■] ) :
    if  caso trivial:
        return resposta conhecida
    solprobmaissimples  = função( [■] )
    retorna a combinação da solução p/o [■] com  solprobmaissimples
```

Argumento mais simples

68

Funções recursivas: como funciona?

A chamada recursiva é uma nova instância da função (uma "cópia") → parâmetros e variáveis locais são distintos.

Só os parâmetros/variáveis criados pela função (ou instância) são visíveis.

Exemplo: Função SubirEscada

Definição Recursiva:

$$subirEscada\ \text{com}\ n\ \text{degraus} = \begin{cases} parar, & n = 0 \\ \text{subir1degrau} + subirEscada\ \text{com}\ n\text{-}1\ \text{degraus}, & n > 0 \end{cases}$$

Do que a função precisa para realizar sua tarefa?

Funções recursivas: como funciona?

A chamada recursiva é uma nova instância da função (uma "cópia") → parâmetros e variáveis locais são distintos.

Só os parâmetros/variáveis criados pela função (ou instância) são visíveis.

Exemplo: Função SubirEscada

Definição Recursiva:

$$subirEscada\ \text{com}\ n\ \text{degraus} = \begin{cases} parar, & n = 0 \\ \text{subir1degrau} + subirEscada\ \text{com}\ n\text{-}1\ \text{degraus}, & n > 0 \end{cases}$$

Do que a função precisa para realizar sua tarefa? Do nº de degraus da escada.

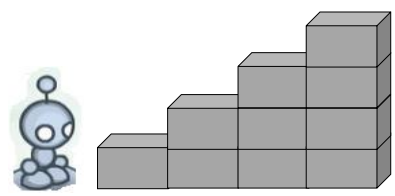
Desenvolvendo a solução

```
def  SubirEscada (n)
        Se   n == 0 :
                chegou no topo da escada/não há escada
        senão:
                Subir 1 degrau
                SubirEscada ( n-1)
```


Simulando a execução

```
def  SubirEscada (n):
    ➡ Se   n == 0 :
                chegou no topo da escada/não há escada
        senão:
                Subir 1 degrau
                SubirEscada ( n-1)
```

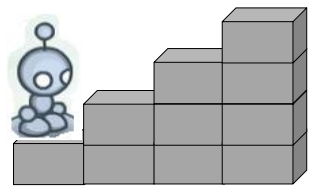
n
4



Simulando a execução

```
def SubirEscada (n):
        Se  n == 0 :
                chegou no topo da escada/não há escada
        senão:
                Subir 1 degrau
                SubirEscada ( n-1)
```

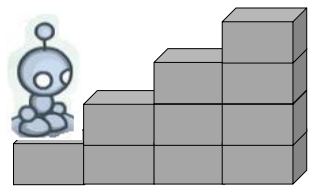
n
4



Simulando a execução

```
def SubirEscada (n):
        Se  n == 0 :
                chegou no topo da escada/não há escada
        senão:
                Subir 1 degrau
                SubirEscada ( n-1)
```

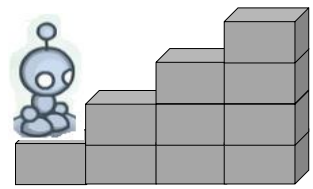
n
4



Simulando a execução

```
def SubirEscada (n):
    Se  n == 0 :
            chegou no topo da escada/não há escada
        senão:
            Subir 1 degrau
            SubirEscada ( n-1)
```

n
3

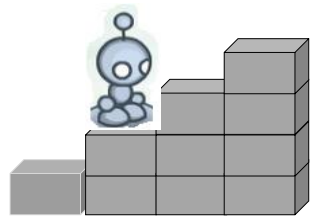


75

Simulando a execução

```
def SubirEscada (n):
        Se  n == 0 :
            chegou no topo da escada/não há escada
        senão:
            Subir 1 degrau
            SubirEscada ( n-1)
```

n
3

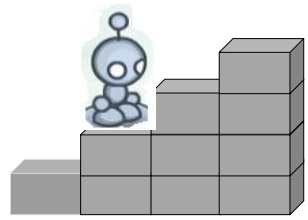


76

Simulando a execução

```
def SubirEscada (n):
        Se  n == 0 :
                chegou no topo da escada/não há escada
        senão:
                Subir 1 degrau
        ➡       SubirEscada ( n-1)
```

n
3

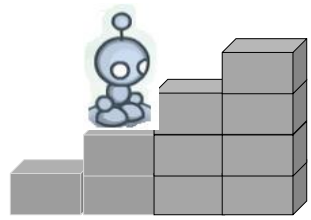


77

Simulando a execução

```
def SubirEscada (n):
    ➡   Se  n == 0 :
                chegou no topo da escada/não há escada
        senão:
                Subir 1 degrau
                SubirEscada ( n-1)
```

n
2

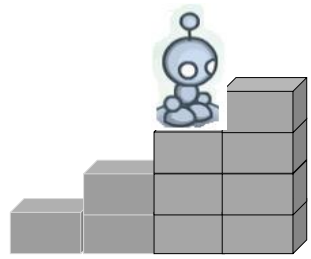


78

Simulando a execução

```
def SubirEscada (n):
        Se  n == 0 :
                chegou no topo da escada/não há escada
        senão:
                Subir 1 degrau
                SubirEscada ( n-1)
```

n
2

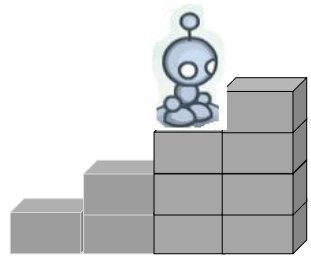


79

Simulando a execução

```
def SubirEscada (n):
        Se  n == 0 :
                chegou no topo da escada/não há escada
        senão:
                Subir 1 degrau
                SubirEscada ( n-1)
```

n
2

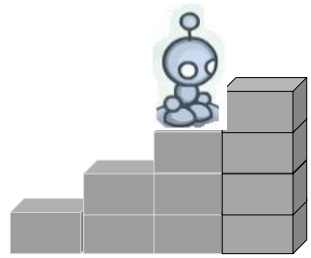


80

Simulando a execução

```
def SubirEscada (n):
    Se  n == 0 :
            chegou no topo da escada/não há escada
        senão:
            Subir 1 degrau
            SubirEscada ( n-1)
```

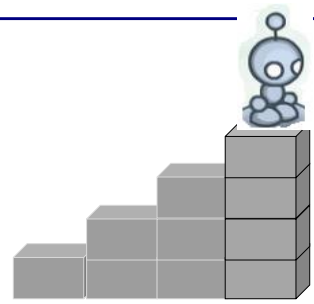
n
1



Simulando a execução

```
def SubirEscada (n):
        Se  n == 0 :
            chegou no topo da escada/não há escada
        senão:
            Subir 1 degrau
            SubirEscada ( n-1)
```

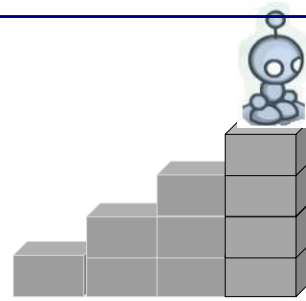
n
1



Simulando a execução

```
def SubirEscada (n):
        Se  n == 0 :
                chegou no topo da escada/não há escada
        senão:
                Subir 1 degrau
                SubirEscada ( n-1)
```

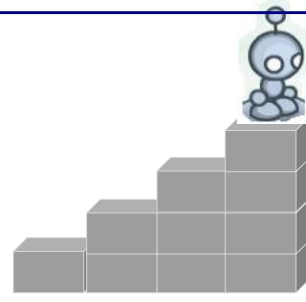
n
1



Simulando a execução

```
def SubirEscada (n):
        Se  n == 0 :
                chegou no topo da escada/não há escada
        senão:
                Subir 1 degrau
                SubirEscada ( n-1)
```

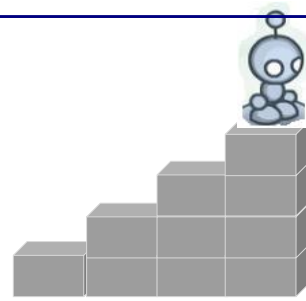
n
0



Simulando a execução

```
def SubirEscada (n):
        Se  n == 0 :
            ➡ chegou no topo da escada/não há escada
        senão:
                Subir 1 degrau
                SubirEscada ( n-1)
```

n
0



85

Roteiro recursivo

- Para se aplicar a **recursividade** deve-se pensar na definição recursiva do problema.

a) Caso base (regra de parada):
Qual é o caso base?
Como reconhecer o caso base?
O que deve ser feito no caso base?

b) Passo Recursivo:
"Como o problema pode ser dividido em problemas menores de mesma natureza? ".
O que deve mudar na chamada recursiva para se aproximar do caso base?
Como deve mudar na chamada recursiva para convergir para o caso base?

c) O que deve ser feito após a chamada recursiva da função, usando o (valor de) retorno da função?

86

Resumo: recursão – forma geral

Esquematicamente, as funções recursivas têm a seguinte forma:

se "condição para o caso de base$_1$" então

resolução direta para o caso de base$_1$

<senão se "condição para o caso de base$_n$" então>

<resolução direta para o caso de base$_n$>

senão

divide a instância do problema em (pelo menos) uma mais simples

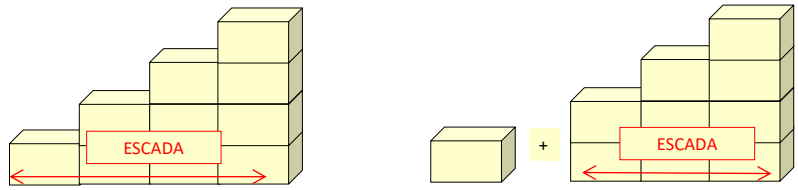
(pelo menos) uma chamada recursiva, com instância menor do que a instância atual do problema, convergindo ao caso base

retorna combinação de resultado das chamadas recursivas

fimse

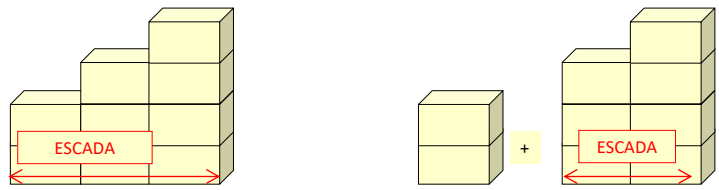
Desenhando a Escada com o Módulo turtle

Desenhando uma escada recursivamente



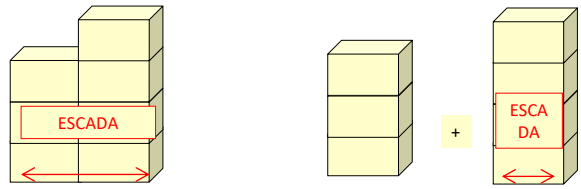
89

Desenhando uma escada recursivamente

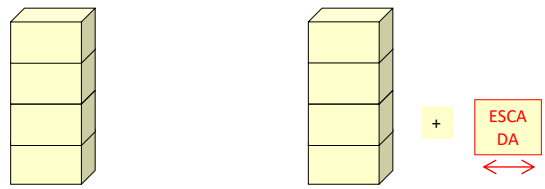


90

Desenhando uma escada recursivamente



Desenhando uma escada recursivamente



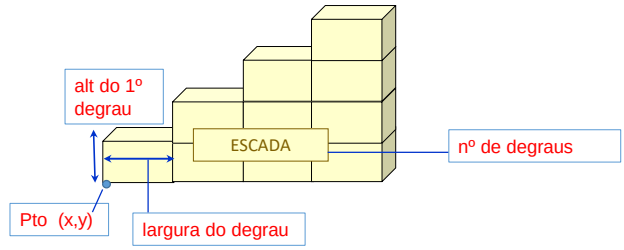
Desenvolvendo a solução

Exemplo: Função DesenharEscada

Definição Recursiva:

$$desenharEscada \text{ com } n \text{ degraus} = \begin{cases} parar, & n = 0 \\ \text{desenhar 1 degrau} + desenharEscada \text{ com } n\text{-}1 \text{ degraus}, & n > 0 \end{cases}$$

Do que a função precisa para realizar sua tarefa?



Desenhando uma escada: solução recursiva

```
import turtle
def Degrau(t,l,a,x,y):
        t.goto(x,y)
        t.begin_fill()
        t.color('LightSteelBlue')
        t.fd(l)
        t.left(90)
        t.fd(a)
        t.left(90)
        t.fd(l)
        t.left(90)
        t.fd(a)
        t.left(90)
        t.end_fill()
        return
```

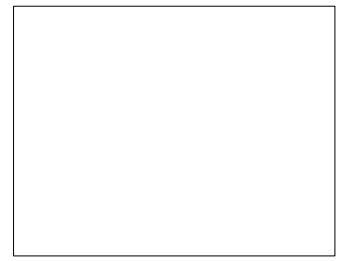
```
def Escada(t,n,l,a,x,y):
    if n == 0:
        return
    Degrau(t,l,a,x,y)
    Escada(t,n-1,l,a+20,x+40,y)
    return
```

Simulando a execução

```
Escada(t,3,40,10,-100,0)
```

```
if n == 0:
    return
Degrau(t,l,a,x,y)
Escada(t,l,a+20,n-1,x+40,y)
return
```

n	l	a	x	y
3	40	10	-100	0

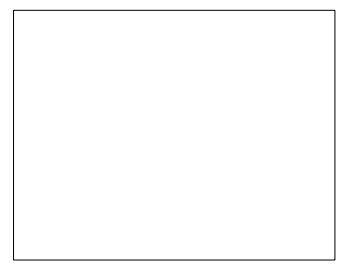


Simulando a execução

```
Escada(t,3,40,10,-100,0)
```

```
if n == 0:
    return
Degrau(t,l,a,x,y)
Escada(t,l,a+20,n-1,x+40,y)
return
```

n	l	a	x	y
3	40	10	-100	0

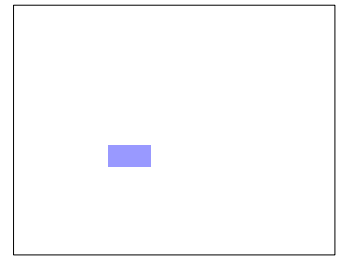


Simulando a execução

```
Escada(t,3,40,10,-100,0)
```

n	l	a	x	y
3	40	10	-100	0

```
if n == 0:
    return
Degrau(t,l,a,x,y)
Escada(t,l,a+20,n-1,x+40,y)
return
```

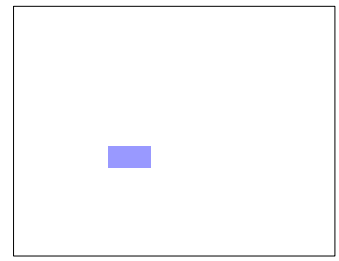


Simulando a execução

```
Escada(t,3,40,10,-100,0)
```

n	l	a	x	y
3	40	10	-100	0

```
if n == 0:
    return
Degrau(t,l,a,x,y)
Escada(t,l,a+20,n-1,x+40,y)
return
```



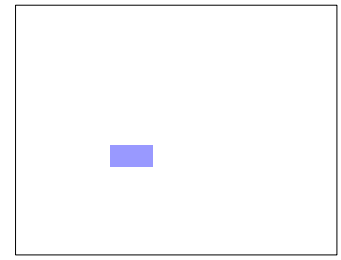
Simulando a execução

```
Escada(t,3,40,10,-100,0)
```

```
Escada(t,2,40,30,-60,0)
```

n	l	a	x	y
2	40	30	-60	0

```
if n == 0:
    return
Degrau(t,l,a,x,y)
Escada(t,l,a+20,n-1,x+40,y)
return
```



Lembre-se:
Cada chamada é uma nova instância da função

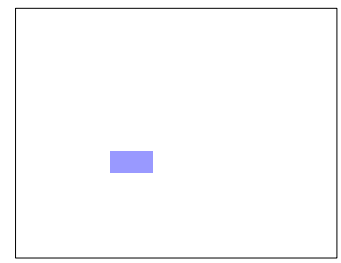
Simulando a execução

```
Escada(t,3,40,10,-100,0)
```

```
Escada(t,2,40,30,-60,0)
```

n	l	a	x	y
2	40	30	-60	0

```
if n == 0:
    return
Degrau(t,l,a,x,y)
Escada(t,l,a+20,n-1,x+40,y)
return
```



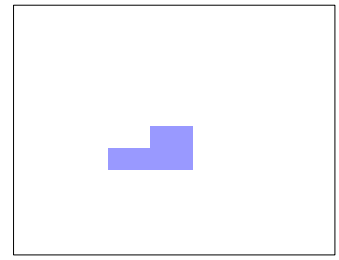
Simulando a execução

```
Escada(t,3,40,10,-100,0)
```

```
Escada(t,2,40,30,-60,0)
```

n	l	a	x	y
2	40	30	-60	0

```
if n == 0:
    return
Degrau(t,l,a,x,y)
Escada(t,l,a+20,n-1,x+40,y)
return
```



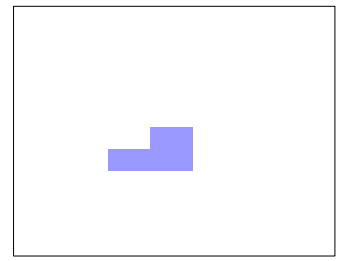
Simulando a execução

```
Escada(t,3,40,10,-100,0)
```

```
Escada(t,2,40,30,-60,0)
```

n	l	a	x	y
2	40	30	-60	0

```
if n == 0:
    return
Degrau(t,l,a,x,y)
Escada(t,l,a+20,n-1,x+40,y)
return
```



Simulando a execução

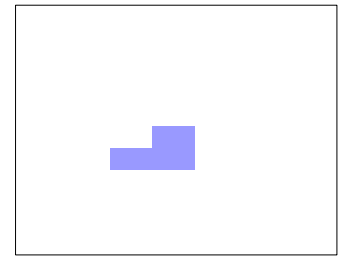
```
Escada(t,3,40,10,-100,0)
```

```
Escada(t,2,40,30,-60,0)
```

```
Escada(t,1,40,50,-20,0)
```

n	l	a	x	y
1	40	50	-20	0

```
if n == 0:
    return
Degrau(t,l,a,x,y)
Escada(t,l,a+20,n-1,x+40,y)
return
```



Lembre-se:
Cada chamada é uma nova instância da função

Simulando a execução

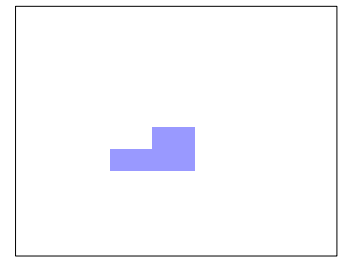
```
Escada(t,3,40,10,-100,0)
```

```
Escada(t,2,40,30,-60,0)
```

```
Escada(t,1,40,50,-20,0)
```

n	l	a	x	y
1	40	50	-20	0

```
if n == 0:
    return
Degrau(t,l,a,x,y)
Escada(t,l,a+20,n-1,x+40,y)
return
```

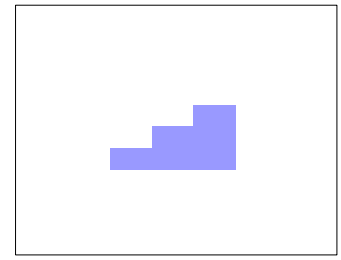


Simulando a execução

```
Escada(t,3,40,10,-100,0)
Escada(t,2,40,30,-60,0)
Escada(t,1,40,50,-20,0)
```

n	l	a	x	y
1	40	50	-20	0

```
if n == 0:
    return
Degrau(t,l,a,x,y)
Escada(t,l,a+20,n-1,x+40,y)
return
```

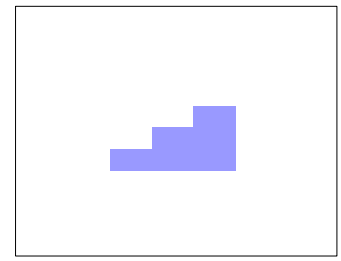


Simulando a execução

```
Escada(t,3,40,10,-100,0)
Escada(t,2,40,30,-60,0)
Escada(t,1,40,50,-20,0)
```

n	l	a	x	y
1	40	50	-20	0

```
if n == 0:
    return
Degrau(t,l,a,x,y)
Escada(t,l,a+20,n-1,x+40,y)
return
```

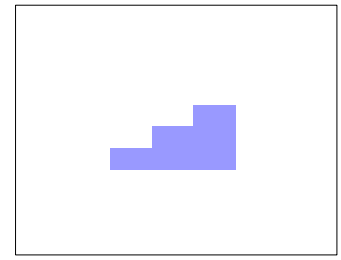


Simulando a execução

```
Escada(t,3,40,10,-100,0)
Escada(t,2,40,30,-60,0)
Escada(t,1,40,50,-20,0)
Escada(t,0,40,70,20,0)
```

n	l	a	x	y
0	40	70	20	0

```
if n == 0:
    return
Degrau(t,l,a,x,y)
Escada(t,l,a+20,n-1,x+40,y)
return
```



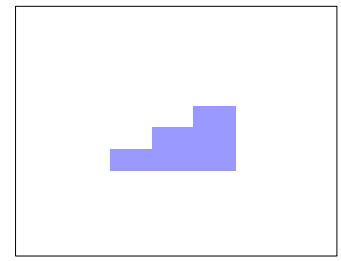
Lembre-se:
Cada chamada é uma nova instância da função

Simulando a execução

```
Escada(t,3,40,10,-100,0)
Escada(t,2,40,30,-60,0)
Escada(t,1,40,50,-20,0)
Escada(t,0,40,70,20,0)
```

n	l	a	x	y
0	40	70	20	0

```
if n == 0:
    return
Degrau(t,l,a,x,y)
Escada(t,l,a+20,n-1,x+40,y)
return
```

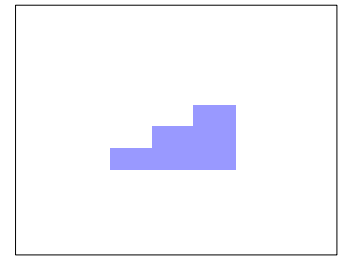


Simulando a execução

```
Escada(t,3,40,10,-100,0)
Escada(t,2,40,30,-60,0)
Escada(t,1,40,50,-20,0)
Escada(t,0,40,70,20,0)
```

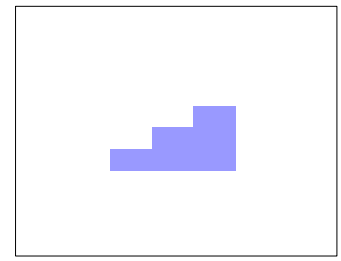
n	l	a	x	y
0	40	70	20	0

```
if n == 0:
    return
Degrau(t,l,a,x,y)
Escada(t,l,a+20,n-1,x+40,y)
return
```



Simulando a execução

```
Escada(t,3,40,10,-100,0)
Escada(t,2,40,30,-60,0)
Escada(t,1,40,50,-20,0)
Escada(t,0,40,70,20,0)
```

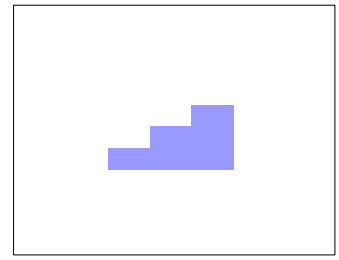


Simulando a execução

```
Escada(t,3,40,10,-100,0)
Escada(t,2,40,30,-60,0)
Escada(t,1,40,50,-20,0)
```

n	l	a	X	y
1	40	50	-20	0

```
if n == 0:
    return
Degrau(t,l,a,x,y)
Escada(t,l,a+20,n-1,x+40,y)
return
```

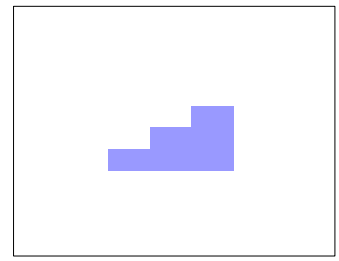


Simulando a execução

```
Escada(t,3,40,10,-100,0)
Escada(t,2,40,30,-60,0)
Escada(t,1,40,50,-20,0)
```

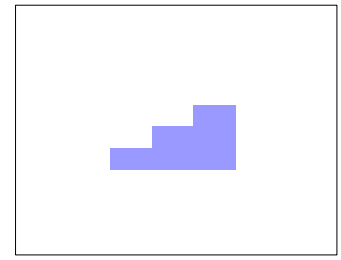
n	l	a	x	y
1	40	50	-20	0

```
if n == 0:
    return
Degrau(t,l,a,x,y)
Escada(t,l,a+20,n-1,x+40,y)
return
```



Simulando a execução

```
Escada(t,3,40,10,-100,0)
Escada(t,2,40,30,-60,0)
Escada(t,1,40,50,-20,0)
```

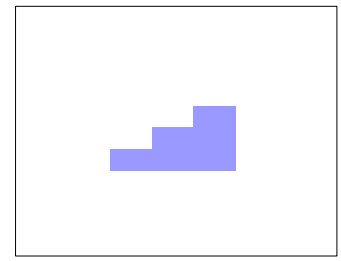


Simulando a execução

```
Escada(t,3,40,10,-100,0)
Escada(t,2,40,30,-60,0)
```

n	l	a	x	y
2	40	30	-60	0

```
if n == 0:
    return
Degrau(t,l,a,x,y)
Escada(t,l,a+20,n-1,x+40,y)
return
```



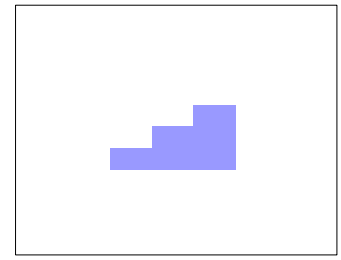
Simulando a execução

```
Escada(t,3,40,10,-100,0)
```

```
Escada(t,2,40,30,-60,0)
```

n	l	a	x	y
2	40	30	-60	0

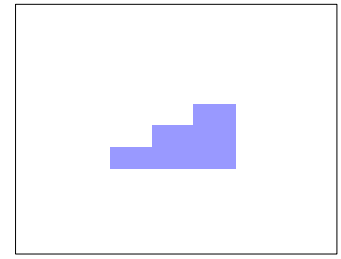
```
if n == 0:
    return
Degrau(t,l,a,x,y)
Escada(t,l,a+20,n-1,x+40,y)
return
```



Simulando a execução

```
Escada(t,3,40,10,-100,0)
```

```
Escada(t,2,40,30,-60,0)
```

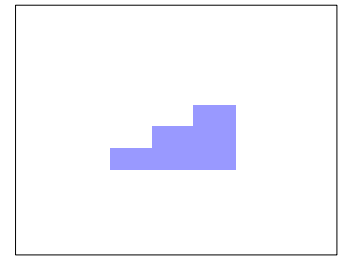


Simulando a execução

```
Escada(t,3,40,10,-100,0)
```

n	l	a	x	y
3	40	10	-100	0

```
if n == 0:
    return
Degrau(t,l,a,x,y)
Escada(t,l,a+20,n-1,x+40,y)
return
```

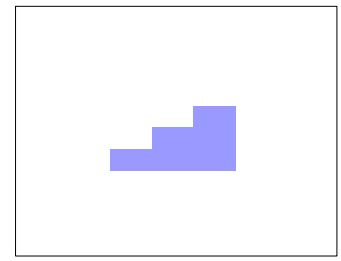


Simulando a execução

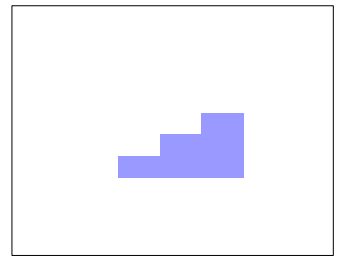
```
Escada(t,3,40,10,-100,0)
```

n	l	a	x	y
3	40	10	-100	0

```
if n == 0:
    return
Degrau(t,l,a,x,y)
Escada(t,l,a+20,n-1,x+40,y)
return
```



Simulando a execução



Complete o exercício

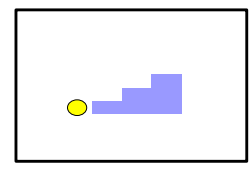
1. Crie uma função recursiva (**subirEscada**) que recebe a tartaruga, o número de degraus e as coordenadas do ponto inicial da escada.
2. Utilize as funções:

```
def sobeDegrau(t,x,y):
    bola(t,'white')
    t.up()
    t.goto(x+25,y)
    bola(t,'yellow')
    return
```

```
def bola(t,cor):
    t.color(cor)
    t.begin_fill()
    t.down()
    t.circle(10)
    t.end_fill()
    return
```

3. Para testar sua função utilize o script:

```
t=turtle.Turtle()
t.hideturtle()
Escada(t,10,40,10,-100,0)
t.goto(-112,0)
bola(t,'yellow')
subirEscada(t,10,-110,12)
```



Função subirEscada recursiva

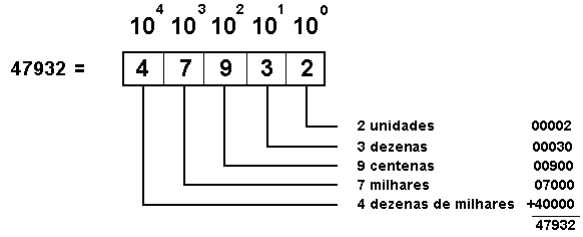
```
import turtle

def sobeDegrau(t,x,y):
    bola(t,'white')
    t.up()
    t.goto(x+25,y)
    bola(t,'yellow')
    return

def subirEscada(t,n,x,y):
    if n == 0:
        return
    sobeDegrau(t,x,y)
    subirEscada(t,n-1,x+40,y+20)
    return
```

121

Interpretação Recursiva de um Número no Sistema Decimal

Sistema decimal

No sistema decimal, como em qualquer sistema posicional, o valor de cada algarismo depende do lugar que ele ocupa na escrita do número. Partindo da primeira casa, da direita para a esquerda, cada posição determina a multiplicação do algarismo por uma potência de 10 (1, 10, 100, 1000...).

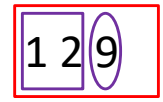


Número ⟺ sequência de algarismos

Interpretação recursiva de um número no sistema decimal

Um número no sistema decimal é:

- ✓ *um algarismo ou*
- ✓ *um algarismo precedido por um nº*



Mãos na massa!!

Crie uma função recursiva para exibir os algarismos de um número positivo verticalmente

Exemplo: 326 → 3
2
6

Qual é o caso base?

Qual o passo recursivo

O que fazer com a solução retornada da chamada para o caso mais simples em cada instância?

Desenvolvendo a solução

Qual é o caso base? Nº <10 – apenas um algarismo

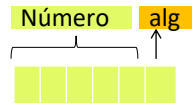
Qual o passo recursivo? Separar a unidade, chamar para o número formado pelos algarismo sem a unidade

O que fazer com a solução retornada da chamada para o caso mais simples em cada instância? Exibir a unidade do Nº

$$\text{impVertical}(\textit{número}) = \begin{cases} \text{exibir}, \textit{número} < 10 \\ \text{impVertical}(\textit{número}/10)\ \text{e exibir a unidade do } \textit{número}, \textit{número} \geq 10 \end{cases}$$

Escrever um número na vertical

```
def ImpVertical(num):
        if num < 10:
            print(num)
            return
        ImpVertical(num//10)  # separa o número que  precede a unidade
        print(num%10)         # separa a unidade do número
        return
```



Simulando a execução

```
impVertical(326)
```

```
if num < 10:
    print(num)
    return
impVertical(num//10)
print(num%10)
return
```

num
326

–

Simulando a execução

```
impVertical(326)
```

```
if num < 10:
    print(num)
    return
impVertical(num//10)
print(num%10)
return
```

num
326

–

Simulando a execução

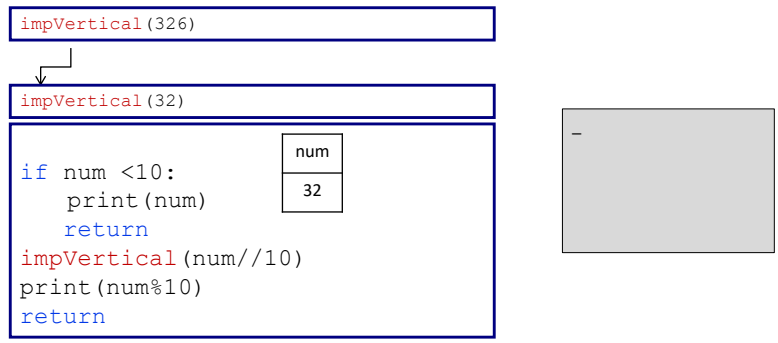
```
impVertical(326)
```

```
if num < 10:
    print(num)
    return
impVertical(num//10)
print(num%10)
return
```

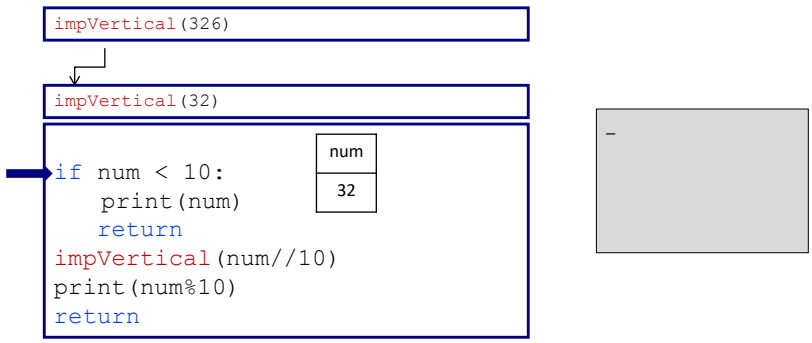
num
326

–

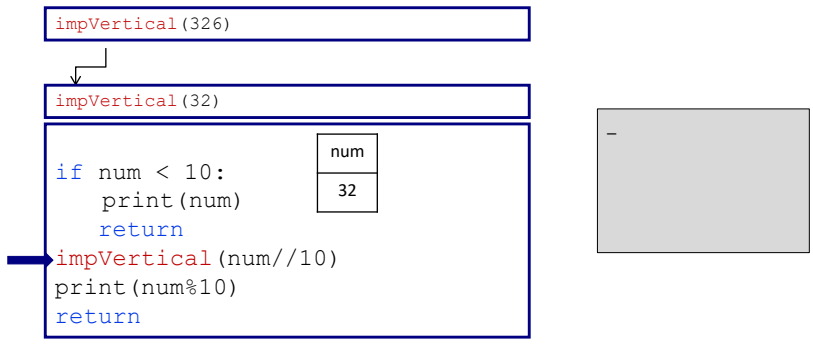
Simulando a execução



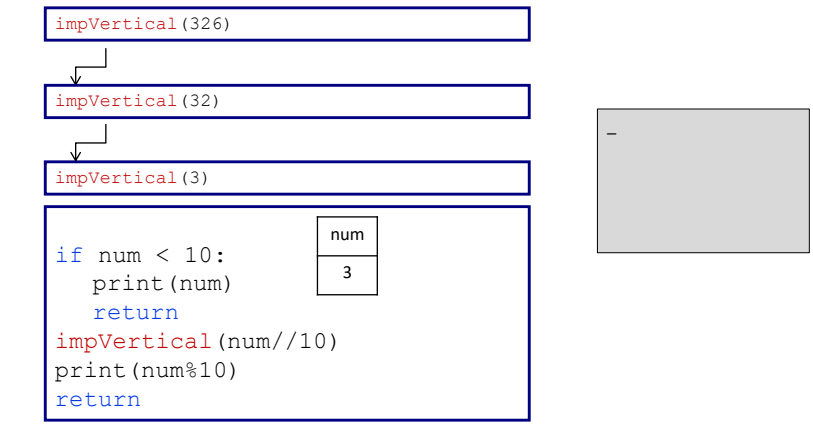
Simulando a execução



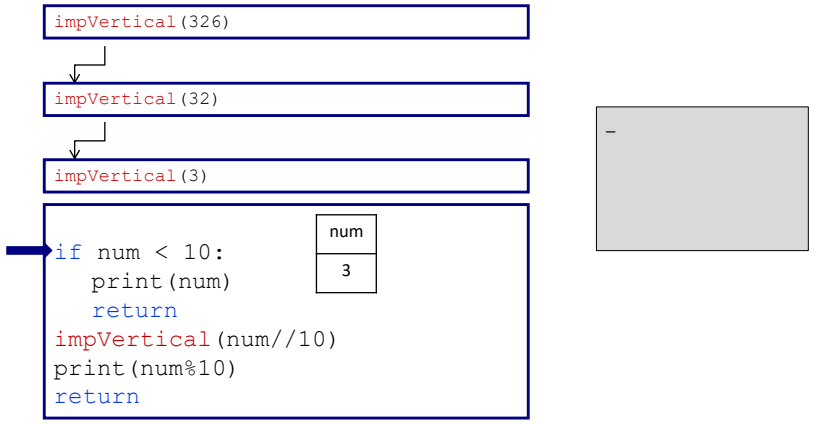
Simulando a execução



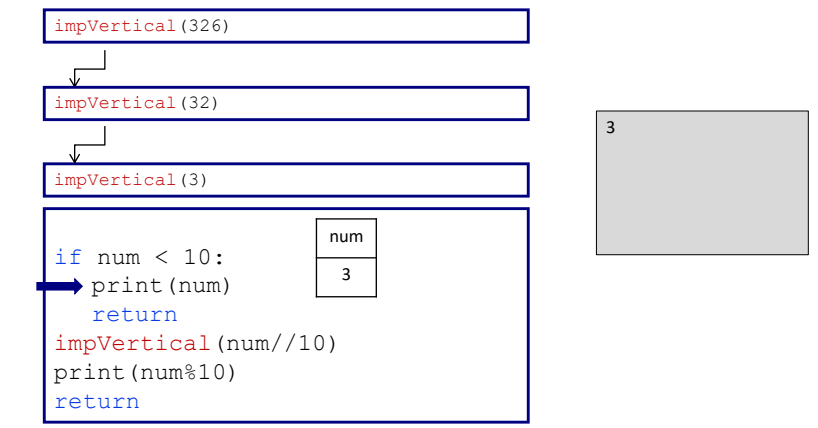
Simulando a execução



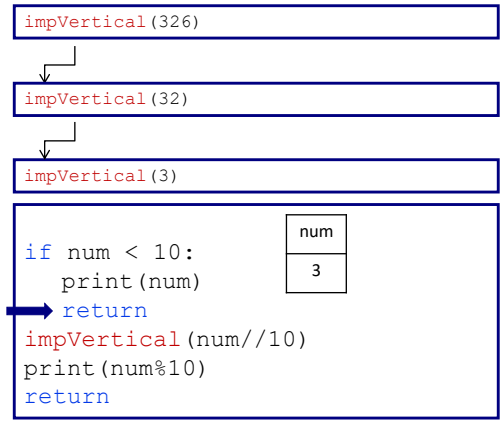
Simulando a execução



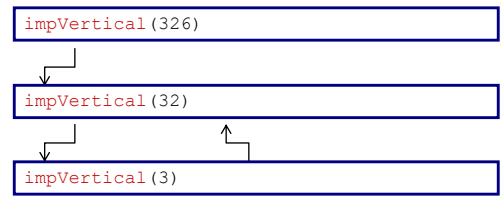
Simulando a execução



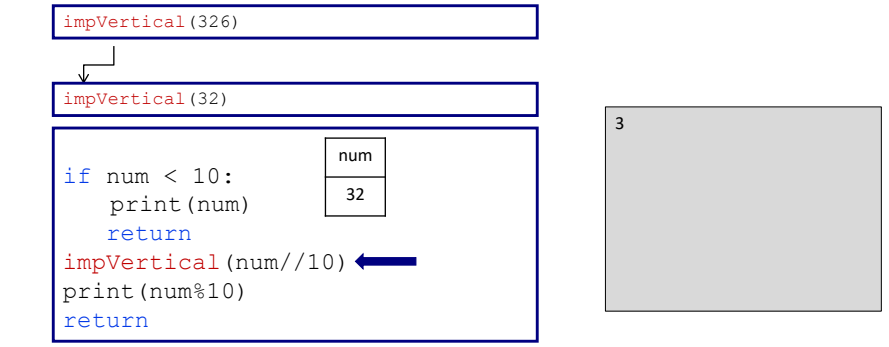
Simulando a execução



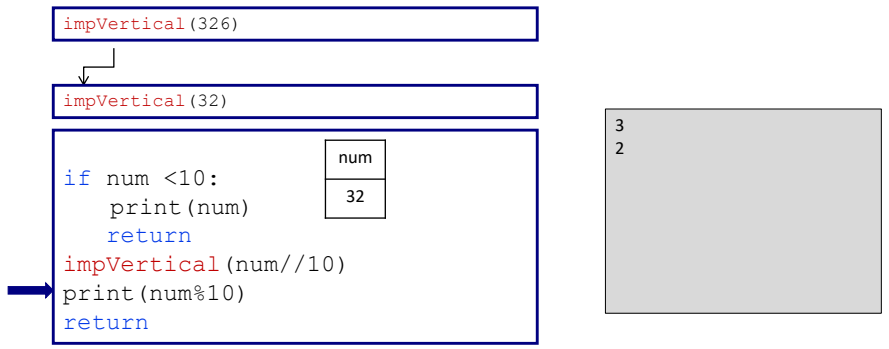
Simulando a execução



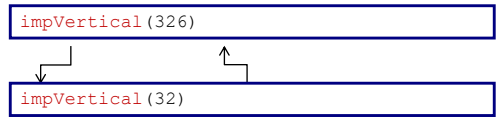
Simulando a execução



Simulando a execução

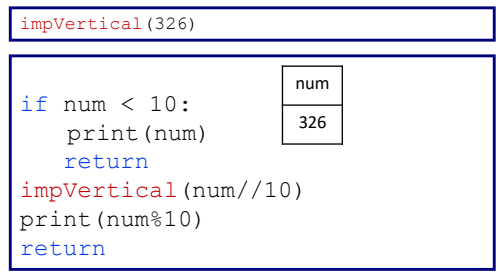


Simulando a execução




```
3
2
```


Simulando a execução




```
3
2
```

Simulando a execução

```
impVertical(326)
```

```
if num < 10:
    print(num)
    return
impVertical(num//10)
print(num%10)
return
```

num
326

```
3
2
6
```


Simulando a execução

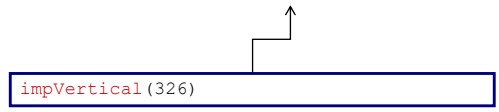
```
impVertical(326)
```

```
if num < 10:
    print(num)
    return
impVertical(num//10)
print(num%10)
return
```

num
326

```
3
2
6
```

Simulando a execução

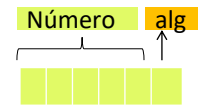



```
3
2
6
```


O que muda para escrever um número de forma invertida, na vertical?

```
def ImpVertical(num):
    if num < 10:
       print(num)
       return
    ImpVertical(num//10)  # separa o número que  precede a unidade
    print(num%10)         # separa a unidade do número
    return
```

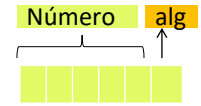
O que muda para exibir o número invertido, verticalmente?



$$impVertical(número) = \begin{cases} \text{exibir, } número < 10 \\ impVertical(número//10) \text{ e exibir a unidade do } número, \; número \geq 10 \end{cases}$$

Uma solução

```
def ImpInvVertical(num):
        if num < 10:
            print(num)
            return
        print(num%10)                  # exibe a unidade do número
        ImpInvVertical(num//10)        # separa o número que  precede a unidade
        return
```



$$\text{impInvVertical(número)} = \begin{cases} \text{exibir, } número < 10 \\ \text{exibir a unidade do } número \text{ e impInvVertical}(número/10),\ número \geq 10 \end{cases}$$

Contando algarismos

Crie uma função recursiva para contar algarismos de um número inteiro

Qual é o caso base?

Nº com 1 algarismo (Nº <10), retornar 1

Qual o passo recursivo?

Separar a unidade, chamar para Nº//10

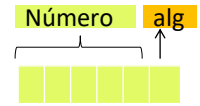
O que fazer com a solução retornada da chamada para o caso mais simples em cada instância?

Somar 1 à quantidade de algarismos contada pela chamada recursiva

Contar algarismos de um número

- Definição recursiva para contar algarismos de um número:

$$\text{qtAlg}(n) = \begin{cases} 1, & n < 10 \\ 1 + \text{qtAlg}(n//10), & n \geq 10 \end{cases}$$



Solução recursiva

```
def  contaAlg (num):
    if num <10:
       return 1
    qt=contaAlg(num//10)
    return 1 + qt
```

OU

```
def  contaAlg (num):
    if num <10:
       return 1
    return 1 + contaAlg(num//10)
```

Simulando a execução

```
contaAlg(326)
```

```
if num <10:
    return 1
qt = contaAlg(num//10)
return 1 + qt
```

num	
326	

Simulando a execução

```
contaAlg(326)
```

```
if num <10:
    return 1
qt = contaAlg(num//10)
return 1 + qt
```

num	
326	

Simulando a execução

```
contaAlg(326)
```

```
if num <10:
    return 1
qt = contaAlg(num//10)
return 1 + qt
```

num	
326	

Simulando a execução

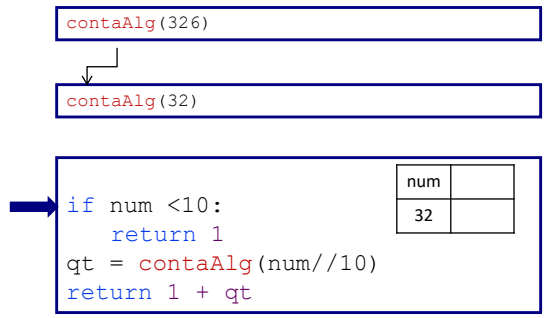
```
contaAlg(326)
```

```
contaAlg(32)
```

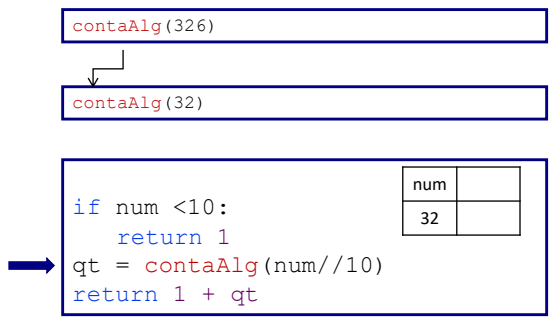
```
if num <10:
    return 1
qt = contaAlg(num//10)
return 1 + qt
```

num	
32	

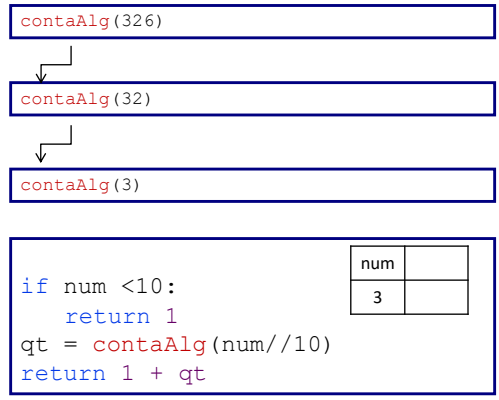
Simulando a execução



Simulando a execução

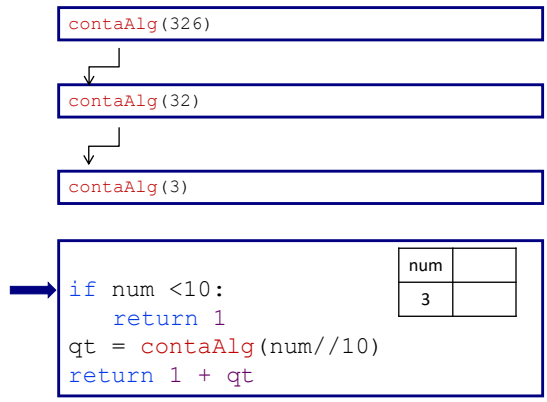


Simulando a execução



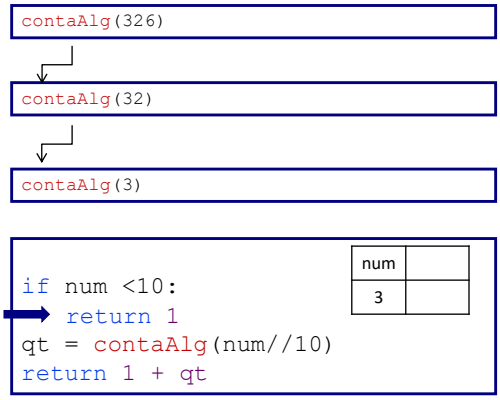
157

Simulando a execução

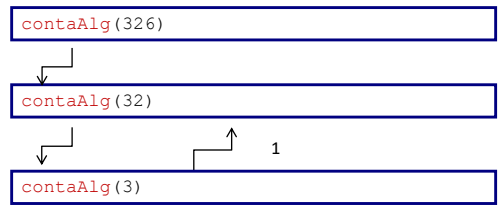


158

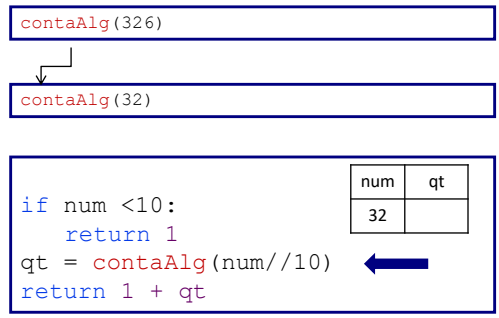
Simulando a execução



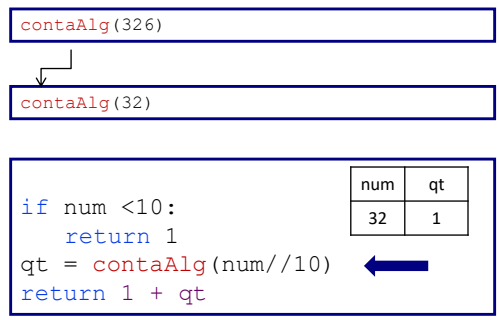
Simulando a execução



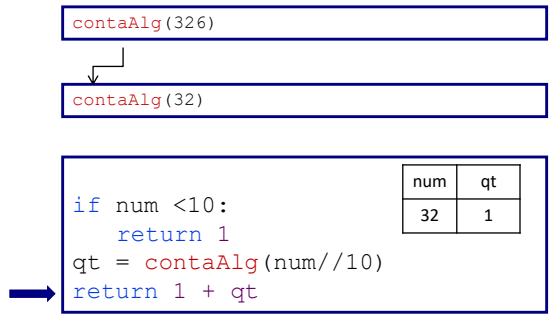
Simulando a execução



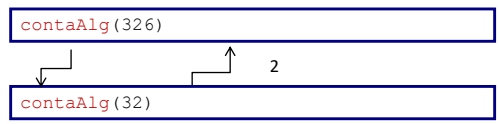
Simulando a execução



Simulando a execução



Simulando a execução



Simulando a execução

```
contaAlg(326)
```

```
if num <10:
    return 1
qt = contaAlg(num//10)
return 1 + qt
```

num	qt
326	

Simulando a execução

```
contaAlg(326)
```

```
if num <10:
    return 1
qt = contaAlg(num//10)
return 1 + qt
```

num	qt
326	2

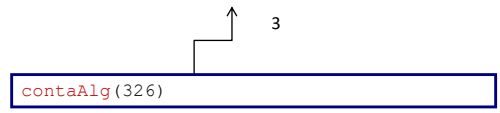
Simulando a execução

```
contaAlg(326)
```

num	qt
326	2

```
if num <10:
    return 1
qt = contaAlg(num//10)
return 1 + qt
```


Simulando a execução

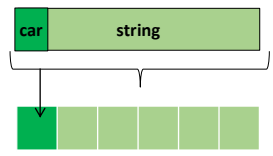


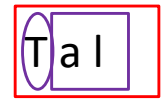
Interpretação Recursiva de uma Sequência de Caracteres

Interpretação recursiva de uma string

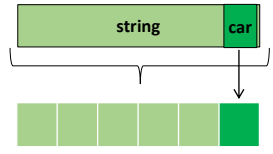
Uma cadeia de caracteres é:

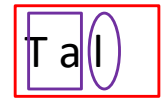
- ✓ *uma sequência de caracteres vazia ou*
- ✓ *um caractere seguido de uma cadeia de caracteres ou*





- ✓ *um caractere precedido por uma sequência de caracteres*





170

Mãos na massa!!

Crie uma função que conte as letras maiúsculas de uma frase

a) Caso base (regra de parada) :
Qual é o caso base?
Como reconhecer o caso base?
O que deve ser feito no caso base?

b) Passo Recursivo:
"Como o problema pode ser dividido em problemas menores de mesma natureza? ".
O que deve mudar na chamada recursiva para se aproximar do caso base?
Como deve mudar na chamada recursiva para convergir para o caso base?

c) O que deve ser feito após a chamada recursiva da função, usando o (valor de) retorno da função?

Desenvolvendo a solução

Qual é o caso base? string vazia
Qual o passo recursivo? string[1:]
O que fazer com a solução retornada da chamada para o caso mais simples em cada instância? Testar se o 1º caractere é letra maiúscula e somar 1 se for

Uma solução

```
def contaMai(palavra):
    if not palavra:
        return 0
    qt=contaMai(palavra[1:])
    if palavra[0]>='A' and palavra[0]<='Z':
        return 1+qt
    else:
        return qt


print(contaMai('F1M!'))
```


Uma solução

```
def contaMai(palavra):
    if not palavra:
        return 0
    qt=contaMai(palavra[1:])
    if palavra[0]>='A' and palavra[0]<='Z':
        return 1+qt
    else:
        return qt


print(contaMai('F1M!'))
```

Em Python, a string vazia tem valor lógico False!!!

Simulando a execução

```
contaMai('F1M!')
```

```
def contaMai(pal):
  if not palavra:
     return 0
  qt=contaMai(palavra[1:])
  if palavra[0]>='A' and palavra[0]<='Z':
       return 1+qt
  else:
       return qt
```

pal	
F1M!	

Simulando a execução

```
contaMai('F1M!')
```

```
def contaMai(pal):
  if not palavra:
     return 0
  qt=contaMai(palavra[1:])
  if palavra[0]>='A' and palavra[0]<='Z':
       return 1+qt
  else:
       return qt
```

pal	
F1M!	

Simulando a execução

```
contaMai('F1M!')
```

```
def contaMai(pal):
  if not palavra:
     return 0
  qt=contaMai(palavra[1:])
  if palavra[0]>='A' and palavra[0]<='Z':
       return 1+qt
  else:
       return qt
```

pal	
F1M!	

Simulando a execução

```
contaMai('F1M!')
```

```
contaMai('1M!')
```

```
def contaMai(pal):
  if not palavra:
     return 0
  qt=contaMai(palavra[1:])
  if palavra[0]>='A' and palavra[0]<='Z':
     return 1+qt
  else:
     return qt
```

pal	
1M!	

Simulando a execução

```
contaMai('F1M!')
```

```
contaMai('1M!')
```

```
def contaMai(pal):
    if not palavra:
        return 0
    qt=contaMai(palavra[1:])
    if palavra[0]>='A' and palavra[0]<='Z':
        return 1+qt
    else:
        return qt
```

pal	
1M!	

Simulando a execução

```
contaMai('F1M!')
```

```
contaMai('1M!')
```

```
def contaMai(pal):
    if not palavra:
        return 0
    qt=contaMai(palavra[1:])
    if palavra[0]>='A' and palavra[0]<='Z':
        return 1+qt
    else:
        return qt
```

pal	
1M!	

Simulando a execução

```
contaMai('F1M!')
```

```
contaMai('1M!')
```

```
contaMai('M!')
```

```
def contaMai(pal):
    if not palavra:
        return 0
    qt=contaMai(palavra[1:])
    if palavra[0]>='A' and palavra[0]<='Z':
        return 1+qt
    else:
        return qt
```

pal	
M!	

Simulando a execução

```
contaMai('F1M!')
```

```
contaMai('1M!')
```

```
contaMai('M!')
```

```
def contaMai(pal):
    if not palavra:
        return 0
    qt=contaMai(palavra[1:])
    if palavra[0]>='A' and palavra[0]<='Z':
        return 1+qt
    else:
        return qt
```

pal	
M!	

Simulando a execução

```
contaMai('F1M!')
```

```
contaMai('1M!')
```

```
contaMai('M!')
```

```
def contaMai(pal):
    if not palavra:
      return 0
    qt=contaMai(palavra[1:])
    if palavra[0]>='A' and palavra[0]<='Z':
        return 1+qt
    else:
        return qt
```

pal	
M!	

Simulando a execução

```
contaMai('F1M!')
```

```
contaMai('1M!')
```

```
contaMai('M!')
```

```
contaMai('!')
```

```
def contaMai(pal):
    if not palavra:
      return 0
    qt=contaMai(palavra[1:])
    if palavra[0]>='A' and palavra[0]<='Z':
        return 1+qt
    else:
        return qt
```

pal	
!	

Simulando a execução

```
contaMai('F1M!')
contaMai('1M!')
contaMai('M!')
contaMai('!')
```

```
def contaMai(pal):
    if not palavra:
        return 0
    qt=contaMai(palavra[1:])
    if palavra[0]>='A' and palavra[0]<='Z':
        return 1+qt
    else:
        return qt
```

pal	
!	

Simulando a execução

```
contaMai('F1M!')
contaMai('1M!')
contaMai('M!')
contaMai('!')
```

```
def contaMai(pal):
    if not palavra:
        return 0
    qt=contaMai(palavra[1:])
    if palavra[0]>='A' and palavra[0]<='Z':
        return 1+qt
    else:
        return qt
```

pal	
!	

Simulando a execução

```
contaMai('F1M!')
contaMai('1M!')
contaMai('M!')
contaMai('!')
contaMai('')
```

```
def contaMai(pal):
    if not palavra:
      return 0
    qt=contaMai(palavra[1:])
    if palavra[0]>='A' and palavra[0]<='Z':
        return 1+qt
    else:
        return qt
```

pal	
''	

187

Simulando a execução

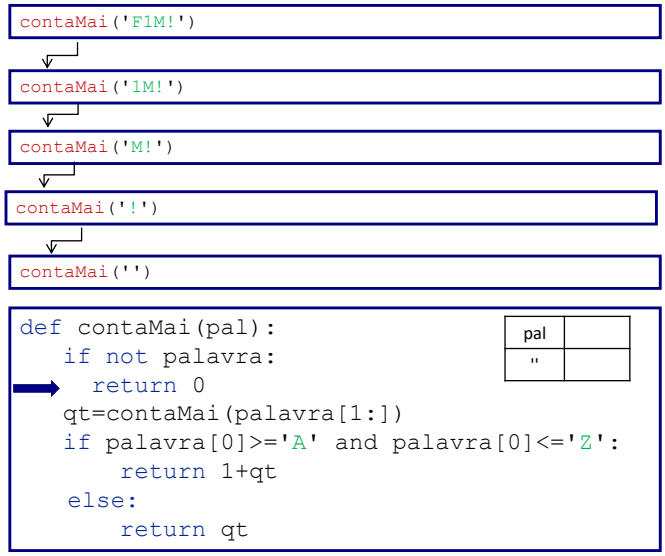
```
contaMai('F1M!')
contaMai('1M!')
contaMai('M!')
contaMai('!')
contaMai('')
```

```
def contaMai(pal):
➡   if not palavra:
      return 0
    qt=contaMai(palavra[1:])
    if palavra[0]>='A' and palavra[0]<='Z':
        return 1+qt
    else:
        return qt
```

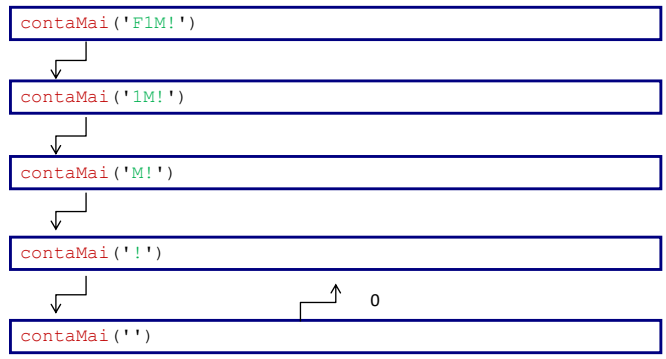
pal	
''	

188

Simulando a execução



Simulando a execução



Simulando a execução

```
contaMai('F1M!')
contaMai('1M!')
contaMai('M!')
contaMai('!')
```

```
def contaMai(pal):
    if not palavra:
      return 0
    qt=contaMai(palavra[1:])
    if palavra[0]>='A' and palavra[0]<='Z':
        return 1+qt
    else:
        return qt
```

pal	
!	

Simulando a execução

```
contaMai('F1M!')
contaMai('1M!')
contaMai('M!')
contaMai('!')
```

```
def contaMai(pal):
    if not palavra:
      return 0
    qt=contaMai(palavra[1:])
    if palavra[0]>='A' and palavra[0]<='Z':
        return 1+qt
    else:
        return qt
```

pal	qt
!	0

Simulando a execução

```
contaMai('F1M!')
contaMai('1M!')
contaMai('M!')
contaMai('!')
```

```
def contaMai(pal):
    if not palavra:
      return 0
    qt=contaMai(palavra[1:])
    if palavra[0]>='A' and palavra[0]<='Z':
        return 1+qt
    else:
        return qt
```

pal	qt
!	0

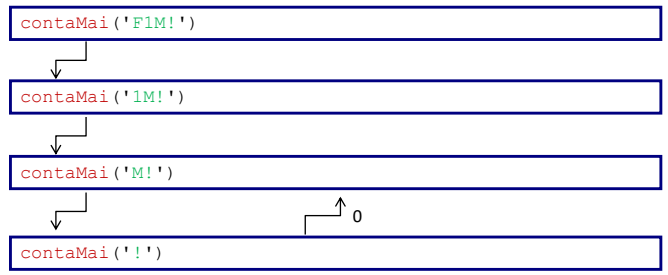
Simulando a execução

```
contaMai('F1M!')
contaMai('1M!')
contaMai('M!')
contaMai('!')
```

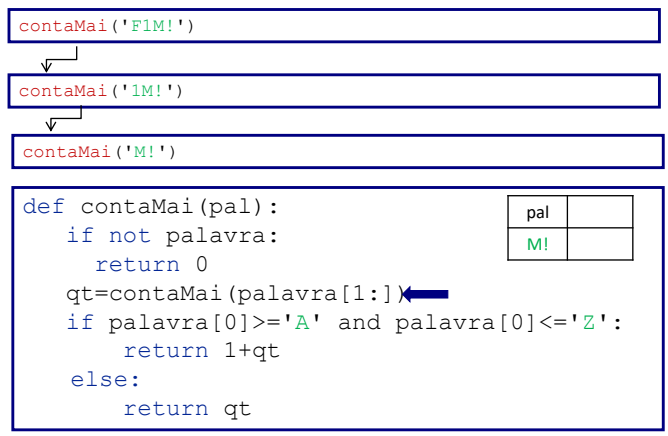
```
def contaMai(pal):
    if not palavra:
      return 0
    qt=contaMai(palavra[1:])
    if palavra[0]>='A' and palavra[0]<='Z':
        return 1+qt
    else:
        return qt
```

pal	qt
!	0

Simulando a execução



Simulando a execução



Simulando a execução

```
contaMai('F1M!')
```

```
contaMai('1M!')
```

```
contaMai('M!')
```

```
def contaMai(pal):
    if not palavra:
      return 0
    qt=contaMai(palavra[1:])
    if palavra[0]>='A' and palavra[0]<='Z':
        return 1+qt
    else:
        return qt
```

pal	qt
M!	0

Simulando a execução

```
contaMai('F1M!')
```

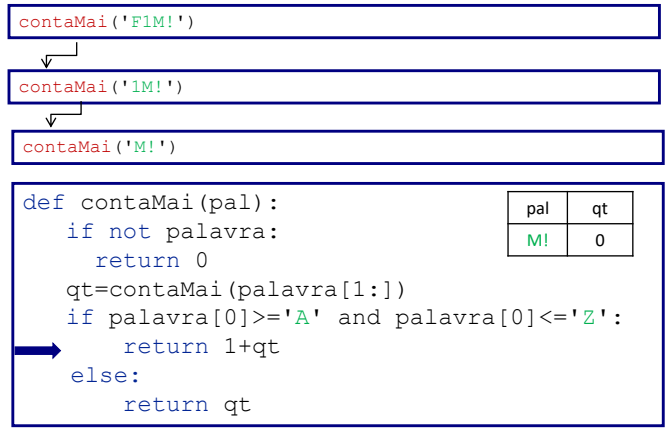
```
contaMai('1M!')
```

```
contaMai('M!')
```

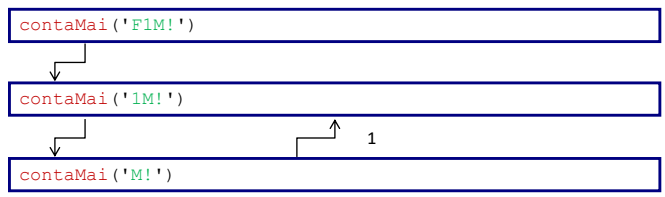
```
def contaMai(pal):
    if not palavra:
      return 0
    qt=contaMai(palavra[1:])
    if palavra[0]>='A' and palavra[0]<='Z':
        return 1+qt
    else:
        return qt
```

pal	qt
M!	0

Simulando a execução



Simulando a execução



Simulando a execução

```
contaMai('F1M!')
```

```
contaMai('1M!')
```

```
def contaMai(pal):
  if not palavra:
     return 0
  qt=contaMai(palavra[1:])
  if palavra[0]>='A' and palavra[0]<='Z':
     return 1+qt
  else:
     return qt
```

pal	
1M!	

Simulando a execução

```
contaMai('F1M!')
```

```
contaMai('1M!')
```

```
def contaMai(pal):
  if not palavra:
     return 0
  qt=contaMai(palavra[1:])
  if palavra[0]>='A' and palavra[0]<='Z':
     return 1+qt
  else:
     return qt
```

pal	qt
1M!	1

Simulando a execução

```
contaMai('F1M!')
```

```
contaMai('1M!')
```

```
def contaMai(pal):
  if not palavra:
     return 0
  qt=contaMai(palavra[1:])
  if palavra[0]>='A' and palavra[0]<='Z':
     return 1+qt
  else:
     return qt
```

pal	qt
1M!	1

Simulando a execução

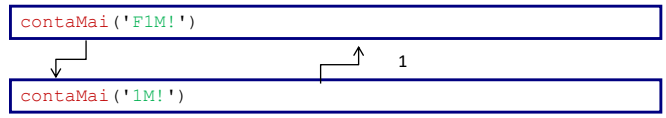
```
contaMai('F1M!')
```

```
contaMai('1M!')
```

```
def contaMai(pal):
  if not palavra:
     return 0
  qt=contaMai(palavra[1:])
  if palavra[0]>='A' and palavra[0]<='Z':
     return 1+qt
  else:
     return qt
```

pal	qt
1M!	1

Simulando a execução



Simulando a execução

```
contaMai('F1M!')
```

```
def contaMai(pal):
  if not palavra:
     return 0
  qt=contaMai(palavra[1:])
  if palavra[0]>='A' and palavra[0]<='Z':
       return 1+qt
  else:
       return qt
```

pal	
F1M!	

Simulando a execução

```
contaMai('F1M!')
```

```
def contaMai(pal):
  if not palavra:
     return 0
  qt=contaMai(palavra[1:])
  if palavra[0]>='A' and palavra[0]<='Z':
       return 1+qt
  else:
       return qt
```

pal	qt
F1M!	1

Simulando a execução

```
contaMai('F1M!')
```

```
def contaMai(pal):
  if not palavra:
     return 0
  qt=contaMai(palavra[1:])
  if palavra[0]>='A' and palavra[0]<='Z':
       return 1+qt
  else:
       return qt
```

pal	qt
F1M!	1

Simulando a execução

```
contaMai('F1M!')
```

```
def contaMai(pal):
  if not palavra:
     return 0
  qt=contaMai(palavra[1:])
  if palavra[0]>='A' and palavra[0]<='Z':
       return 1+qt
  else:
       return qt
```

pal	qt
F1M!	1

Simulando a execução

2

```
contaMai('F1M!')
```

Strings verticais

Crie uma função recursiva que exiba verticalmente uma string

a) Caso base (regra de parada) :
Qual é o caso base?
Como reconhecer o caso base?
O que deve ser feito no caso base?

b) Passo Recursivo:
"Como o problema pode ser dividido em problemas menores de mesma natureza? "
O que deve mudar na chamada recursiva para se aproximar do caso base?
Como deve mudar na chamada recursiva para convergir para o caso base?

c) O que deve ser feito após a chamada recursiva da função, usando o (valor de) retorno da função?

Desenvolvendo a solução

Qual é o caso base?
string vazia
Qual o passo recursivo?
exibir o 1º caractere, chamar para string[1:]
O que fazer com a solução retornada da chamada para o caso mais simples em cada instância?
nada!

DEPARTAMENTO DE INFORMÁTICA PUC-RIO

Uma solução

```
def exibeVertical(palavra):
    if not palavra:
        return
    print(palavra[0])
    exibeVertical(palavra[1:])
    return



exibeVertical('palavra')
```

213