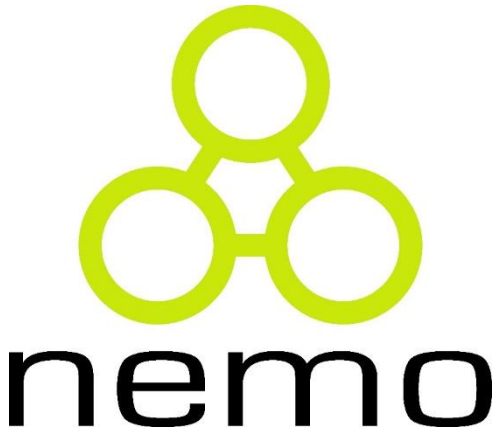


Programação I

Jordana S. Salamon

jssalamon@inf.ufes.br

jordanasalamon@gmail.com

DEPARTAMENTO DE INFORMÁTICA

CENTRO TECNOLÓGICO

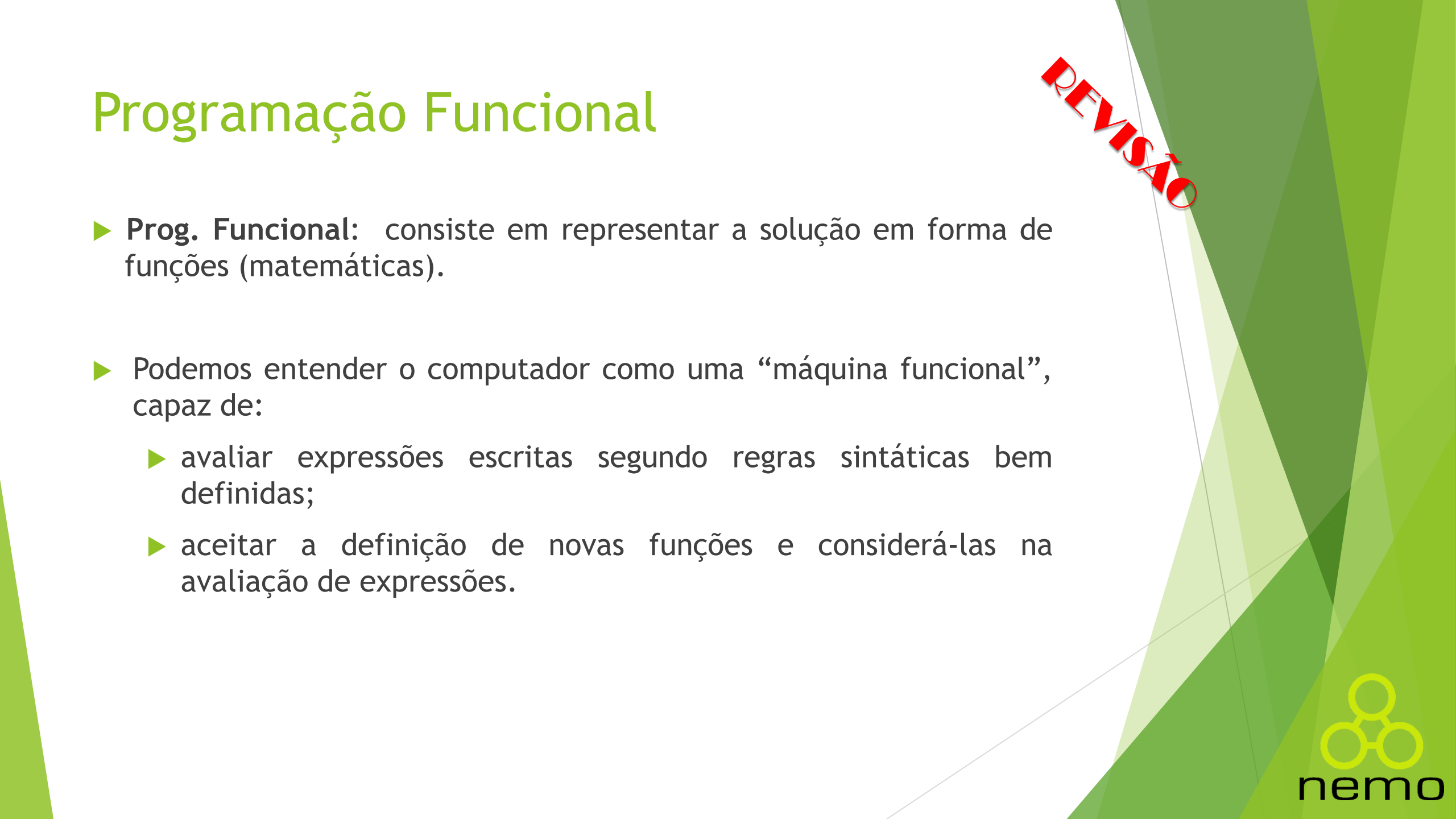
UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Programação Funcional

REVISÃO

- **Prog. Funcional**: consiste em representar a solução em forma de funções (matemáticas).

- Podemos entender o computador como uma “máquina funcional”, capaz de:
  - avaliar expressões escritas segundo regras sintáticas bem definidas;
  - aceitar a definição de novas funções e considerá-las na avaliação de expressões.



Descrição de Funções em Python

REVISÃO

- ***Notação Matemática*** ***Notação de Pyhton***

  espaco(v, t) = v x t **def** espaco(v, t)**: return** v * t

	INTERFACE DA FUNÇÃO			CORPO DA DEFINIÇÃO
	nome da função	parâmetros		expressão aritmética que define a relação que há entre os parâmetros
def	espaco	(v, t)	**: return**	v * t

Descrição de Funções em Python

REVISÃO

- Funções paramétricas
  - **def** espaco(v, t)**: return** v * t
- Funções não-paramétricas (constantes)
  - **def** pi()**: return** 3.1416

nemo

Descrição de Funções em Python

REVISÃO

- Funções globais
  - **def** quad(x): **return** x * x
  - **def** pi(): **return** 3.1416
  - **def** areaCirc(r): **return** pi() * quad(r)

- Funções locais
  - **def** areaCirc2(r):

    def pi2(): **return** 3.1614

    def quad2(): **return** r * r

    return pi2() * quad2()

nemo

Princípios para resolução de problemas

- Abstração
- Generalização
- Instanciação
- Modularização

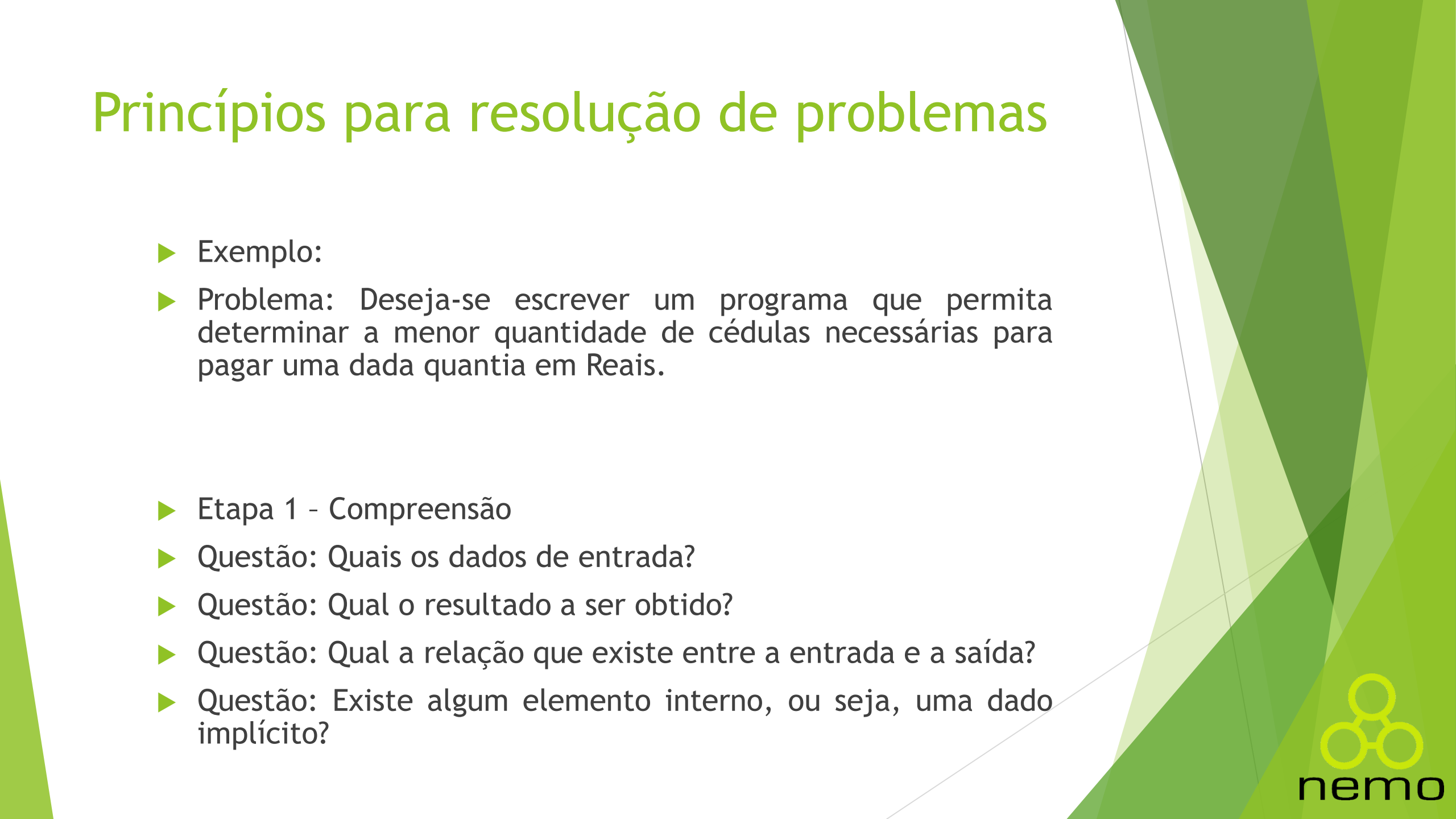
Princípios para resolução de problemas

- Em síntese, a fase de desenvolvimento compreende as seguintes subfases:
  1) construção da solução;
  2) planejamento do teste;
  3) execução manual do teste;
  4) codificação da solução;
  5) teste com o uso do computador.

Princípios para resolução de problemas

- Exemplo:
- Problema: Deseja-se escrever um programa que permita determinar a menor quantidade de cédulas necessárias para pagar uma dada quantia em Reais.

- Etapa 1 – Compreensão
- Questão: Quais os dados de entrada?
- Questão: Qual o resultado a ser obtido?
- Questão: Qual a relação que existe entre a entrada e a saída?
- Questão: Existe algum elemento interno, ou seja, uma dado implícito?



Princípios para resolução de problemas

- Etapa 2 – Planejamento
- Conheço algum problema parecido? Existe um problema mais simples?
- Podemos entender como um problema mais simples um que busque determinar quantas cédulas de um dado valor são necessárias para pagar a quantia desejada.
- Exemplo:

Quantia de dinheiro	Valor da Cédula	Quantidade de cédulas	Resto
289,00	100	289 / 100 = 2	89,00
89,00	50	89 / 50 = 1	39,00
39,00	10	39 / 10 = 3	9,00
9,00	5	9 / 5 = 1	4,00
4,00	1	4 / 1 = 4	0,00
	TOTAL	**11**	

nemo

Princípios para resolução de problemas

- Etapa 3 - Desenvolvimento
- Solução 1 - Definimos a seguir a função nCedulasA em que cada linha da coluna “Quantidade de cédulas” corresponde a uma sub-expressão da definição que, somadas, retornam o resultado esperado.

```
>>> def nCedulasA(q):
return q//100 + (q % 100)//50 + ((q % 100) % 50)//10 +
(((q % 100) % 50) % 10)//5 + ((((q % 100) % 50) % 10) % 5)//1
>>> nCedulasA(289)
11
```

nemo

Princípios para resolução de problemas

- Etapa 3 – Desenvolvimento

```
>>> nCedulasA(289)
➔ 289//100 + (289 % 100)//50 + ((289 % 100) % 50)//10 + (((289 % 100) %
  50) % 10)//5 + ((((289 % 100) % 50) % 10) % 5)//1
➔ 2 + 89//50 + (89%50)//10 + ((89%50)%10)//5 + (((89%50)%10)%5)//1
➔ 2 + 1 + 39//10 + (39 % 10)//5 + ((39 % 10) % 5)//1
➔ 2 + 1 + 3 + 9//5 + (9 % 5)//1
➔ 2 + 1 + 3 + 1 + 4//1
➔ 2 + 1 + 3 + 1 + 4
➔ 11
```

- É a melhor solução?

nemo

Princípios para resolução de problemas

- Etapa 3 – Desenvolvimento

- Solução 2 - Considerando uma propriedade das cédulas, ou seja, já que uma cédula qualquer é múltiplo das menores, a determinação do resto não precisa considerar as cédulas maiores do que a cédula que estamos considerando em um dado ponto.

```
>>> def nCedulasB(q): return q // 100 + (q % 100)//50 + (q %
50)//10 + (q % 10)//5 + (q % 5)//1
>>> nCedulasB(289)
11
```

nemo

Princípios para resolução de problemas

- Etapa 4 – Avaliação do processo

```
>>> def nCedulasC(q): return n100(q) + n50(q) + n10(q) + n5(q) + n1(q)
>>> def n100(q): return q // 100
>>> def n50(q): return resto100(q) // 50
>>> def n10(q): return resto50(q) // 10
>>> def n5(q): return resto10(q) // 5
>>> def n1(q): return resto5(q) // 1
>>> def resto100(q): return q % 100
>>> def resto50(q): return q % 50
>>> def resto10(q): return q % 10
>>> def resto5(q): return q % 5
>>> nCedulasC(289)
11
```

```
>>> def nCedulasD(q):
        def n100(): return q // 100
        def n50(): return resto100() // 50
        def n10(): return resto50() // 10
        def n5(): return resto10() // 5
        def n1(): return resto5() // 1
        def resto100(): return q % 100
        def resto50(): return resto100() % 50
        def resto10(): return resto50() % 10
        def resto5(): return resto10() % 5
        return n100() + n50() + n10() + n5() + n1()
>>> nCedulasD(289)
11
```

nemo

Abstração

- Escreva a expressão que determina a hipotenusa de um triângulo de lados 4 e 10.

```
>>> from math import sqrt
>>> sqrt((10 * 10) + (4 * 4))
10.770329614269007
```

- E se quisermos chamar várias vezes essa mesma expressão?

```
>>> def hipotenusa(): return sqrt((10 * 10) + (4 * 4))
>>> hipotenusa()
10.770329614269007
```

nemo

Generalização

- Porque escrever uma definição de hipotenusa que retorna sempre o mesmo valor, ao invés de generalizá-la?

```
>>> def hipotenusa(x, y): return sqrt((x * x) + (y * y))
```

Instanciação

- Uma vez definida a função genérica para cálculo da hipotenusa, podemos instanciá-la para diferente valores de catetos:

```
>>> hipotenusa(10, 4)
10.770329614269007
>>> hipotenusa(35, 18)
39.357337308308857
>>> hipotenusa(9, 12)
15.0
```

Modularização

- Quando nos deparamos com problemas maiores, um bom princípio é:
  - Dividir para facilitar a conquista.

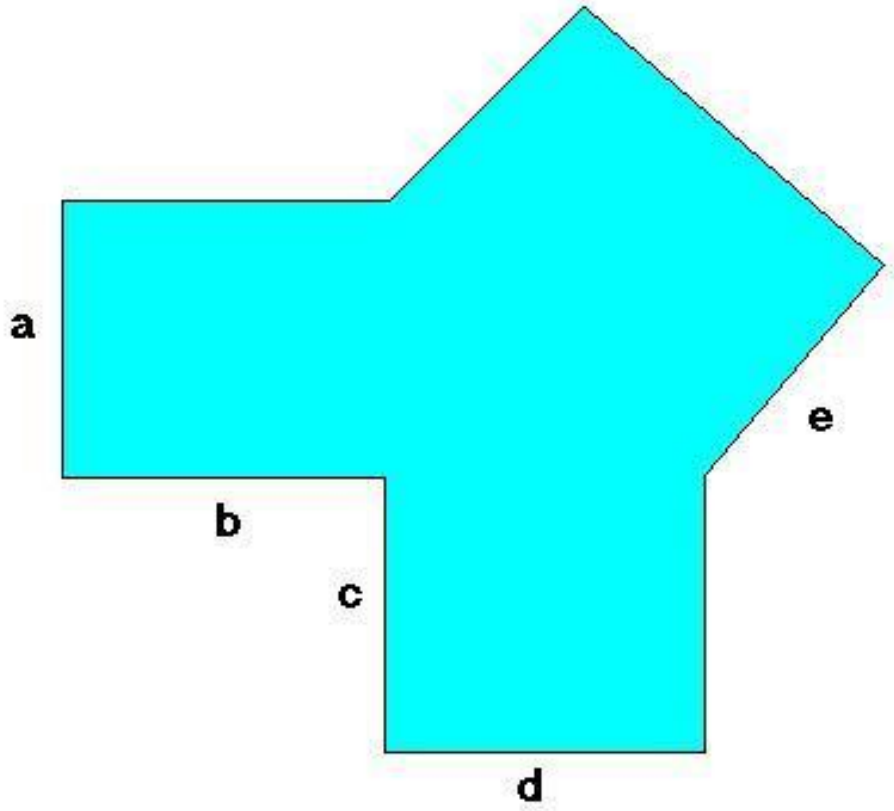
- Basicamente este princípio consiste em:

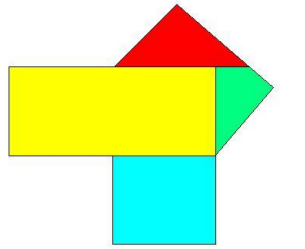
  i) quebrar o problema inicial em problemas menores;

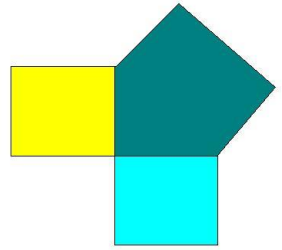
  ii) elaborar a solução para cada um dos problemas menores e;

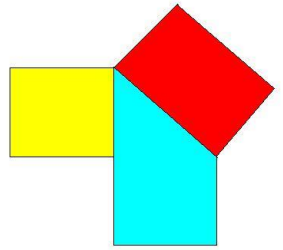
  iii) combinar estes subproblemas para obter a solução do problema inicial

Modularização



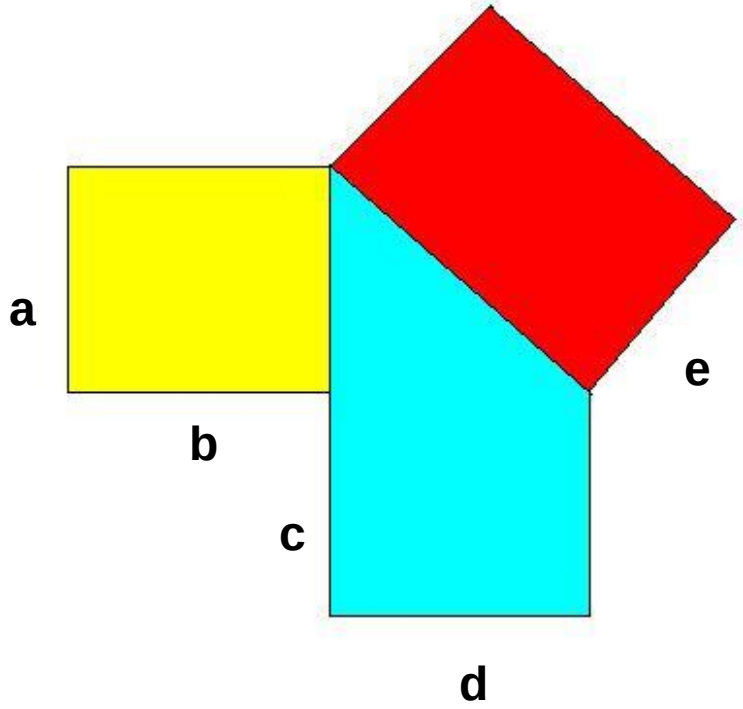


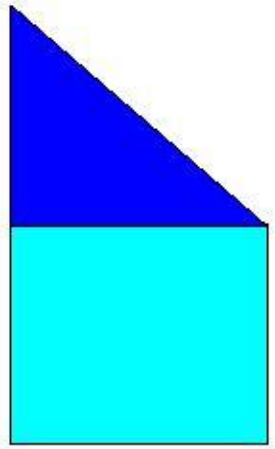




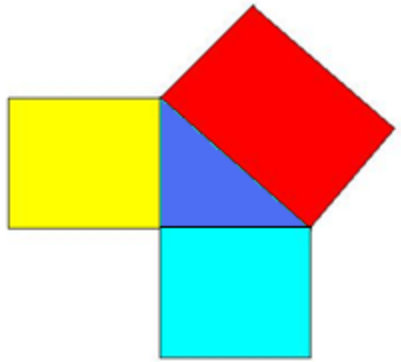
nemo

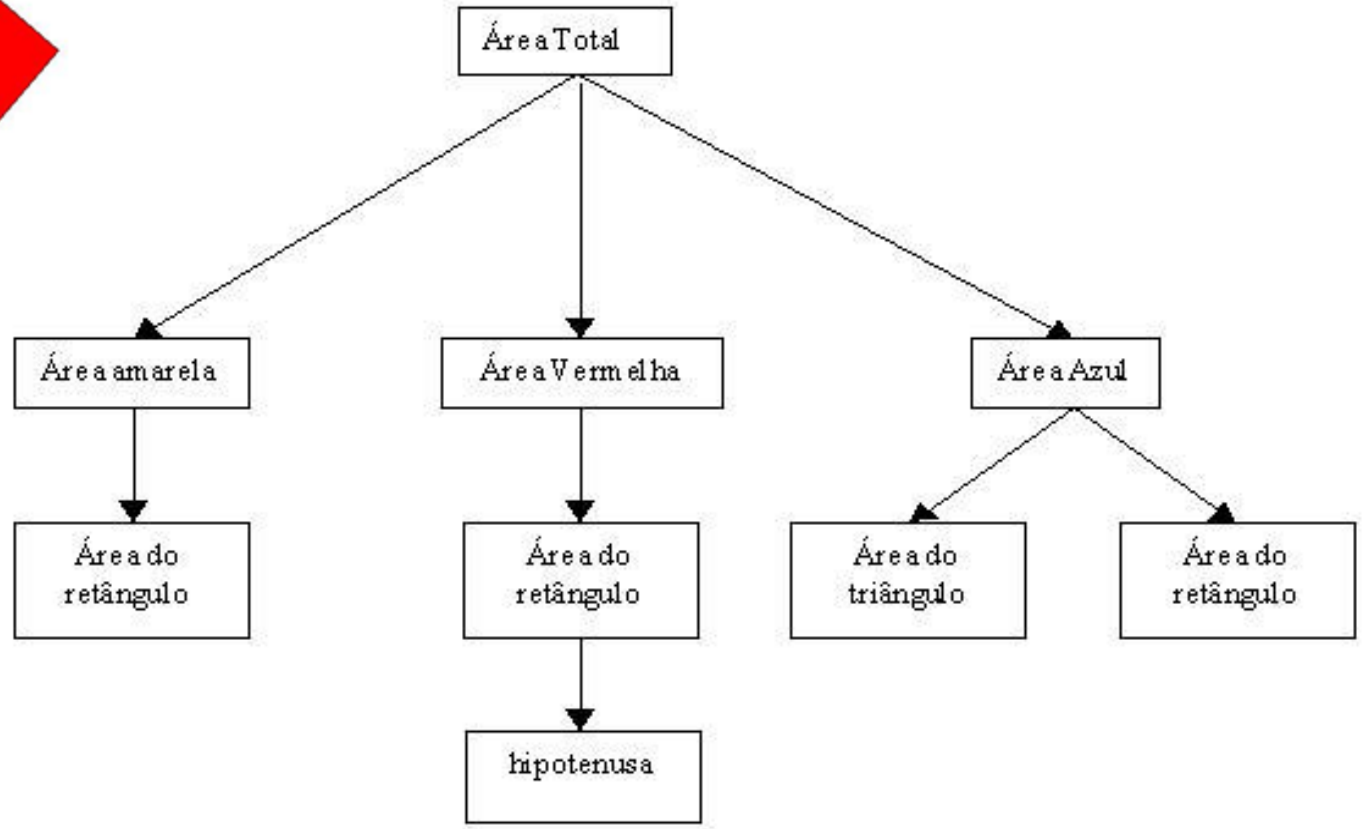
Modularização





Modularização

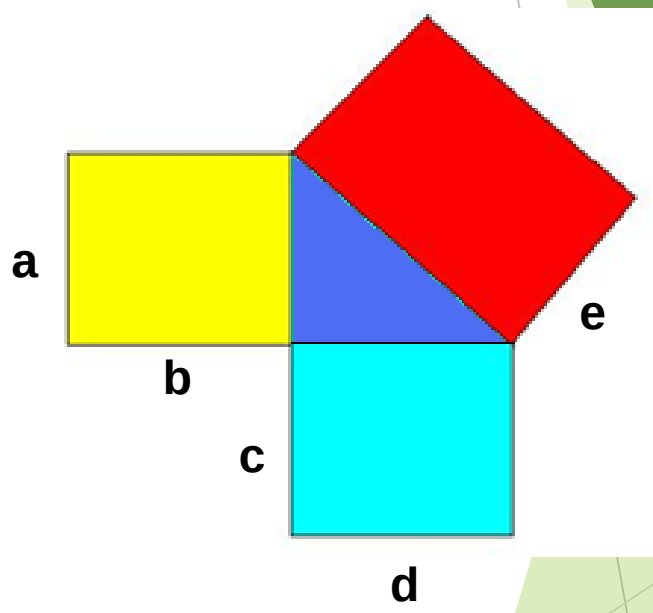




nemo

Modularização

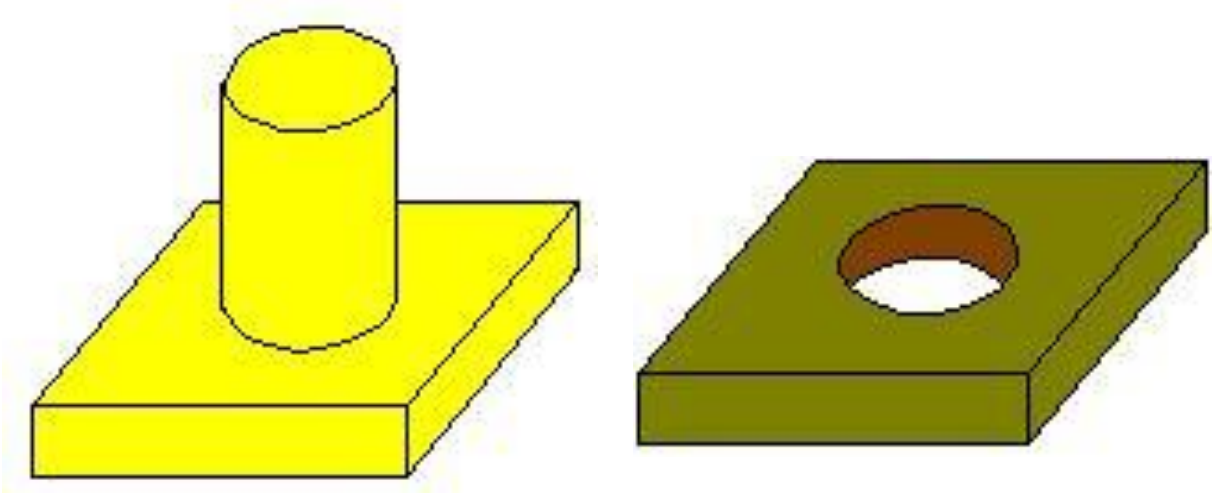
```
>>> def areaRetangulo(x, y): return x * y

>>> def areaTrianguloRetangulo(x, y):
    return (x * y)/2

>>> def areaTotal(a, b, c, d, e):
    return areaRetangulo(a, b)
    + areaRetangulo(hipotenusa(a, d), e)
    + areaAzul(a, c, d)

>>> def areaAzul(a, c, d):
    return areaRetangulo(c, d)
    + areaTrianguloRetangulo(a, d)
```



nemo

Exercício

- Descreva como calcular o volume das peças



That’s all Folks!