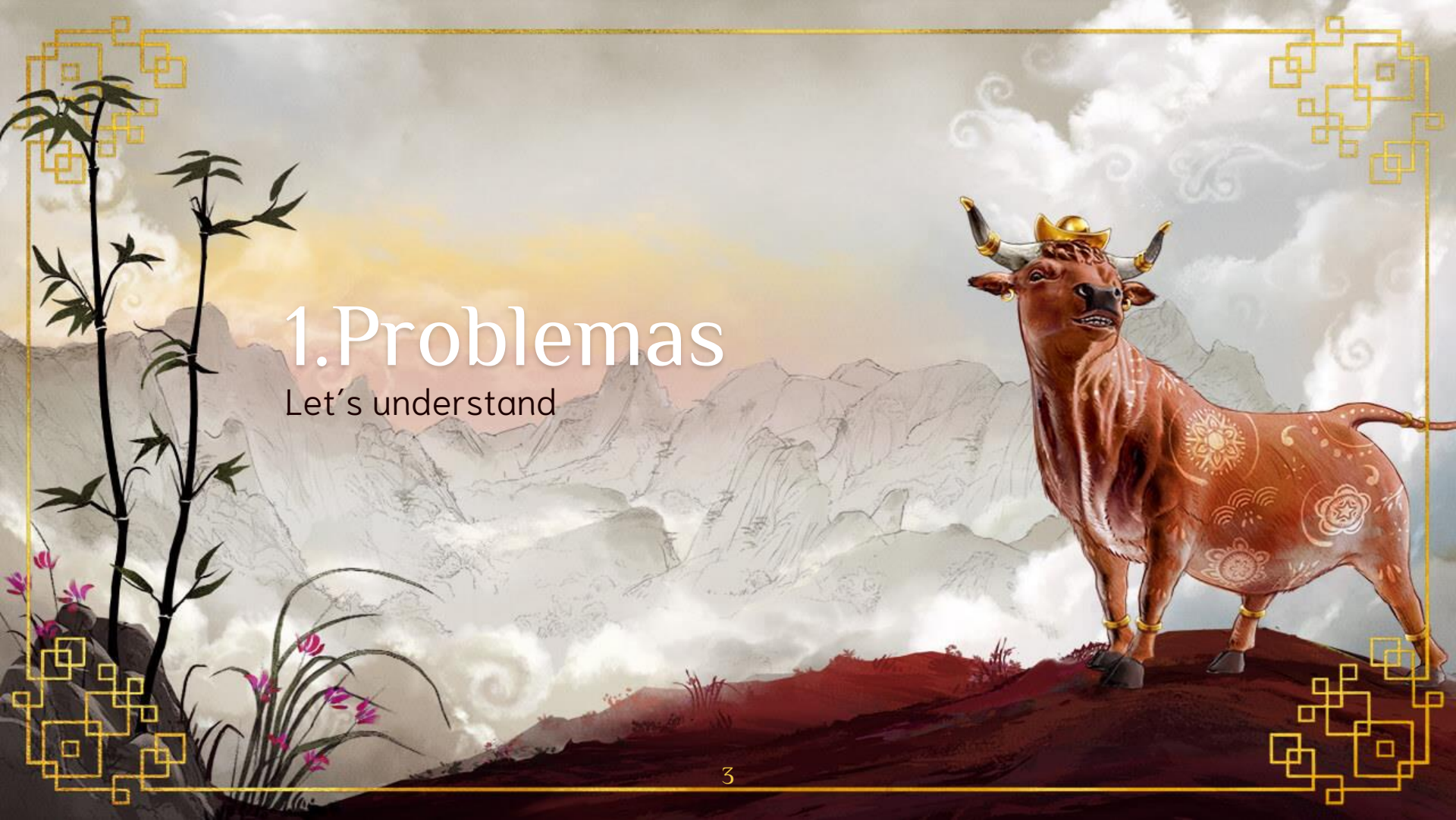
Como Resolver Problemas Computacionais

“Ensinar é indicar o caminho,
mas na viagem cada um vai ver
o que quiser ver.”

(Plotino)



1.Problemas

Let´s understand

O difícil caminho fácil

- Problema é algo que nos incomoda
- Atacar o efeito sem descobrir a causa.
- Encontrar soluções antes de conhecer o problema.
- Não existem problemas sem solução, existem problemas mal formulados.
- Todo o problema trás em si sua solução.
- Os efeitos colaterais deixam sequelas e cicatrizes maiores e mais prejudiciais que os sintomas originais.

Passos para Resolver um Problema

1. Compreender o problema
2. Construir um plano de ação
3. Executar o plano
4. Rever a solução

(Polya)

1. Compreensão do Problema

- Quais são os dados do problema?
- Quais são as incógnitas? (o que eu não sei?)
- Quais as condições ou restrições?
- É possível satisfazer as condições?
- Fazer um esquema separando os dados em partes

2. Construir um Plano de Ação

- Encontrar uma conexão entre o que sei com o que quero saber
- Já resolvi problema semelhante? Consigo adaptar métodos usados em outros problemas para este?
- Conheço resultados ou fórmulas que possam a judar?
- Posso enunciar o problema de forma diferente?
- Consigo resolver parte do problema?

2. Construir um Plano de Ação

- Estratégias:
  - Transformar o problema em um caso particular que seja mais acessível ou...
  - Transformar o problema em outro equivalente
  - Esquecer o problema por algum tempo
- Necessário engenho e arte

3. Executar o Plano

- Necessário paciência e cuidado
- Percebe claramente que cada passo está correto?
- Pode provar que cada passo está correto?
- Se a dificuldade for técnica, persista
- Se o plano não puder ser executado, revise-o

4. Rever a Solução

- Frequentemente deixado de lado, mas muito importante
- Forma de consolidar o conhecimento e desenvolver habilidade de resolução de problemas
- Posso checar o resultado? Parece razoável?
- Posso encontrar uma maneira alternativa de resolver o problema?
- Posso usar o mesmo método em outro problema?

Perguntas:

- Existe alguma palavra, frase ou parte da proposição do problema que não entendo?
- Qual é a dificuldade do problema?
- Qual é a meta?
- Quais são os dados que estou usando como ponto de partida?
- Conheço algum problema similar?

Dicas:

- Tornar a propor o problema usando seus próprios termos.
- Explicar aos colegas em que consiste o problema.
- Modificar o formato da proposição do problema (usar gráficos, desenhos etc.)
- Quando é muito geral, concretizar o problema usando exemplos.
- Quando é muito específico, tentar generalizar o problema.

2.Algoritmos

Let´s prepare

Entendendo os Termos

- Os ingredientes são as **entradas de dados**
- O bolo é a **saída de dados**
- A receita do bolo é o **algoritmo**
- A receita, ou algoritmo é o **software**
- Os utensílios (forno, panela) são o **hardware**
- O cozinheiro é a **CPU**

Mousse de Chocolate

Ingredientes: 200g de chocolate meio amargo, ½ caixa de leite condensado, 1 caixa de creme de leite

1. Derreta o chocolate em **banho maria**
2. Depois de derretido, adicione o leite condensado e o creme de leite e mexa tudo até virar uma mistura **homogênea**
3. Leve a geladeira e deixe esfriar e atingir a **consistência** de mousse.

Nível de Detalhes / Abstração

- De acordo com o cozinheiro
- Banho Maria ?????
- Consistência de mousse ?????
- Sempre devemos escrever uma instrução bem definida para o hardware que executará
- Abstrair significa esconder os detalhes de que não precisamos

O Problema Algoritmico

- As entradas e saídas do problema devem ser especificadas.
- Para cada entrada deve ter uma saída correspondente.
- O problema computacional é uma relação formada por:
  1. uma caracterização de uma coleção válida, possivelmente infinita, das possíveis entradas do problema,
  2. uma especificação das saídas desejadas como função das entradas.

Algoritmos - características

- É uma sequência de instruções que resolve um determinado problema.
- Sequência de instruções finita
- Deve conter apenas instruções elementares: claras e não ambíguas
- Deve ser uma sequência sistemática de passos
- Tanto o tempo gasto pelo algoritmo quanto os recursos utilizados são limitados

3.Exemplos

Let´s code

Exemplo 1

Um caminhão de um frigorífico tem a capacidade de transportar 200 peças de queijo ou 500 potes de manteiga. Você acabou de ser contratada pela dona do frigorífico para desenvolver um algoritmo que possibilite o planejamento adequado do transporte dos produtos. Geralmente, se tem uma quantidade de peças e potes. O algoritmo deve calcular e fornecer a quantidade do segundo produto para que o caminhão sempre transporte sua capacidade máxima.

1. **Ler o enunciado do problema:** leia com atenção, destacando as variáveis existentes: quantidade de manteiga e de queijo
2. **Interpretar o texto e as imagens:** temos que imaginar a situação, formando as imagens mentais: o caminhão, as peças de queijo, os potes de manteiga, e a relação entre as peças e os potes

Exemplo 1 (cont.)

3. **Identificar os dados e as relações:** número de peças de queijo (até 200) e número de potes de manteiga (até 500)
4. **Definir as incógnitas**: peças de queijo e potes de manteiga
5. **Escrever as equações:** podemos utilizar a regra de 3 simples: 200 peças está para 500 potes, assim como Q peças está para M potes.
6. **Resolver o sistema de equações**: 500Q = 200M .: Q = 0,4M e M = 2,5Q
7. **Responder o problema**: No lugar de cada queijo, consigo colocar 2,5 manteigas, e cada manteiga ocupa 0,4 pedaços de queijo. Para calcular a quantidade de cada um que posso levar, devo verificar a quantidade de espaço livre de cada um e multiplicar pelo seu equivalente:

totalQueijo = (500 – numeroManteigas) * 0,4

totalManteiga = (200 – numeroQueijo) * 2,5

```
Algoritmo Frigorifico {
    //DECLARAÇÃO DE VARIÁVEIS
    int queijo, manteiga;
    float totalQueijo, totalManteiga;
    //ENTRADA DE DADOS
    escreva("Quantas peças de queijo? ");
    leia (queijo);
    escreva("Quantos potes de manteiga? ");
    leia(manteiga);
    //PROCESSAMENTO
    totalQueijo = (500 - manteiga) * 0.4;
    totalManteiga = (200 - queijo) * 2.5;
    //SAÍDA DE DADOS
    escreva("\nPara " + manteiga + " de manteiga, " + totalQueijo + " de queijo");
    escreva("\nPara " + queijo + " de queijo, " + totalManteiga + " de manteiga");
}
```

Exemplo 2

Uma fazendeira acabou de montar um cercado. Nele, a fazendeira colocou galinhas e coelhos compartilhando o mesmo espaço. A fazendeira gostaria que você fizesse um algoritmo que determinaria a quantidade de coelhos e a quantidade de galinhas, apenas informando os números totais de cabeças e de pés existentes no cercado.

1. **Ler o enunciado do problema:** leia com atenção, destacando as variáveis existentes: quantidade de cabeças, quantidade de pés
2. **Interpretar o texto e as imagens**: temos que imaginar aqui a situação, formando as imagens mentais: as galinhas e os coelhos. Pensar que as galinhas possuem dois pés e os coelhos 4.

Exemplo 2 (cont.)

3. **Identificar os dados e as relações:** como já foi dito, as galinhas possuem dois pés e os coelhos 4 pés. A quantidade de cabeças é a quantidade total de animas: galinhas mais coelhos.
4. **Definir as incógnitas**: número de cabeças de galinhas (G), número de cabeças de coelhos (C), número de pés de galinhas (PG), número de pés de coelhos (PC), número total de cabeças (TC), número total de pés (TP)
5. **Escrever as equações**: TC = G + C; TP = PG + PC; TP = (2 * G) + (4 * C)
6. **Resolver as equações**: G = TC – C → TP = 2 * (TC – C) + 4 * C → TP = 2TC–2C+4C → TP = 2TC + 2C → **C = (TP – 2TC) / 2** ou C = TC – G → TP = 2 * G + 4 * (TC – G) → TP = 2G+4TC–4G → TP = 4TC – 2G → **G = (TP – 4TC) / 2**
7. **Solucionar o problema**: Dado a quantidade de cabeças e a quantidade de pés, substitui em uma das equações finais acima. Encontrando a quantidade de galinhas, ou a quantidade de coelhos, diminui-se da quantidade de cabeças, encontrando o outro valor: **G = TC – C** ou **C = TC – G**

```
Algoritmo Fazenda {
    //DECLARAÇÃO DE VARIÁVEIS
    inteiro numCabecas, numPes, qtdCoelhos, qtdGalinhas;
    //ENTRADA DE DADOS
    escreva ("Quantas cabeças? ");
    leia (numCabecas);
    escreva ("Quantos pés? ");
    leia (numPes);
    //PROCESSAMENTO
    qtdCoelhos <- (numPes - 2 * numCabecas) / 2;
    qtdGalinhas <- numCabecas - qtdCoelhos;
    //SAÍDA DE DADOS
    escreva("\nQuantidade de coelhos: " + qtdCoelhos);
    escreva("Quantidade de galinhas: " + qtdGalinhas);
}
```

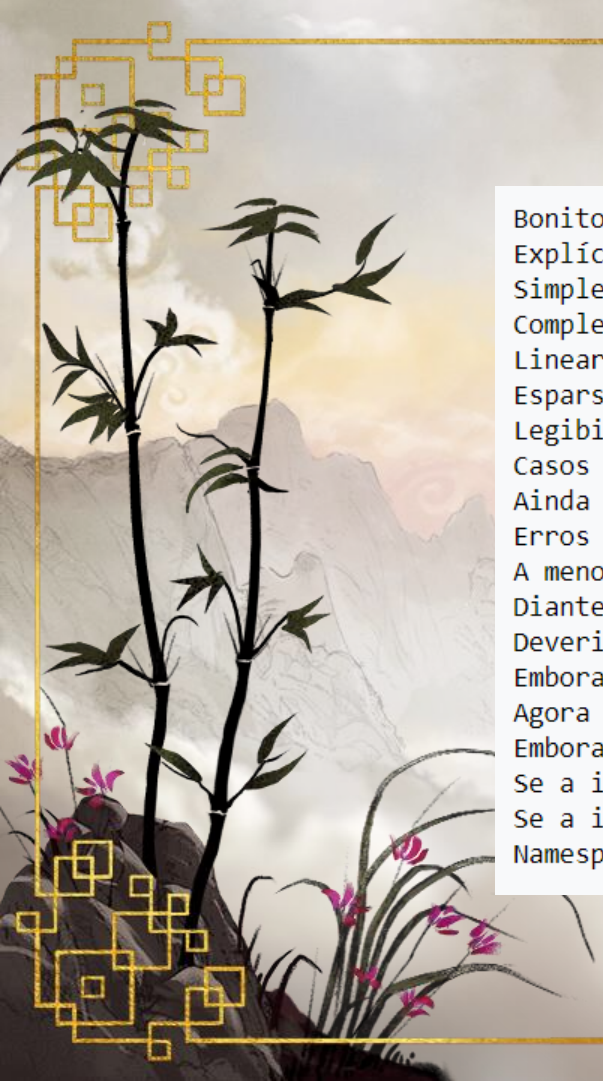
4.Curiosidade

Let´s fun

```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```



```
Bonito é melhor que feio
Explícito é melhor que implícito
Simples é melhor que complexo
Complexo é melhor que complicado
Linear é melhor do que aninhado
Esparso é melhor que denso
Legibilidade conta
Casos especiais não são especiais o bastante para quebrar as regras.
Ainda que praticidade vença a pureza
Erros nunca devem passar silenciosamente.
A menos que sejam explicitamente silenciados
Diante da ambiguidade, recuse a tentação de adivinhar
Deveria haver um — e preferencialmente apenas um — modo óbvio para fazer algo.
Embora esse modo possa não ser óbvio a princípio a menos que você seja holandês
Agora é melhor que nunca
Embora nunca freqüentemente seja melhor que já
Se a implementação é difícil de explicar, é uma má ideia
Se a implementação é fácil de explicar, pode ser uma boa ideia
Namespaces são uma grande ideia — vamos ter mais dessas!
```

谢Perguntas?

professora@lucilia.com.br

Créditos

- Template: SlidesCarnival
- Ilustração de fundo: Alex Monge
- Wikipedia
- Beecrowd