**Dicas para a prova**

* Estruturas de repetição;
* Estruturas de decisão;
* Estrutura sequencial (o que está na sequência do código, qual a lógica);
* Memória, processamento e entrada/saída;
* Dominar isso para a prova e para as próximas etapas
* Um problema que vai ser resolvido e deve estar estruturado com aproximação da realidade;
* Pode trazer anotação de casa, com conceitos e exemplos;
* Sinalizar no código o que é de cada letra (a, b, c, d, e, etc);
* Habilidade de estruturar dentro daquilo que se conhece em cada área;
* Não deixem a prova em branco, ele considera tudo que é colocado;
* Mostrar as variáveis que são caracteres (stings, inputs, etc) e números (int, float, etc);

**Resumo do conteúdo abordado**

**Tipos compostos:**

* For
* For/else
* While
* While/else
* Break
* Continue

**Aula 7**

**Contextualização:**

* A aula de hoje é como pensar na consciência: o que é abstrato e eu posso pensar em calcular conclusões?
* Fazer inferências lógicas que tem uma saída.
* Todo conhecimento que temos e pensamos é uma abstração que resulta em descrições, que caracterizam os objetos e geram classes diferentes.
* Tudo abstrato pode se tornar um programa (tão abstrato como nosso pensamento), em algo concreto, com ações e descrições.

**Abstração:**

* Operação intelectual em que um objeto de reflexão é isolado de fatores que comumente lhe estão relacionados na realidade.
* O resultado dessa operação (termo, ideia, concepção etc.); abstrato.
* Quanto melhor elaborado um programa é, maior o nível de abstração utilizado.
* O que acontece entre uma entrada e uma saída.

**\*Vantagens da abstração:**

* Programa menos acoplado (serve para qualquer contexto, devido a forma genérica, generalizada);
* Mais fácil realizar manutenção;
* Operações mais genéricas (Faz mais com menos código);
* Menor retrabalho;
* Necessita de menos codificação;
* Melhor estruturação de programas maiores.

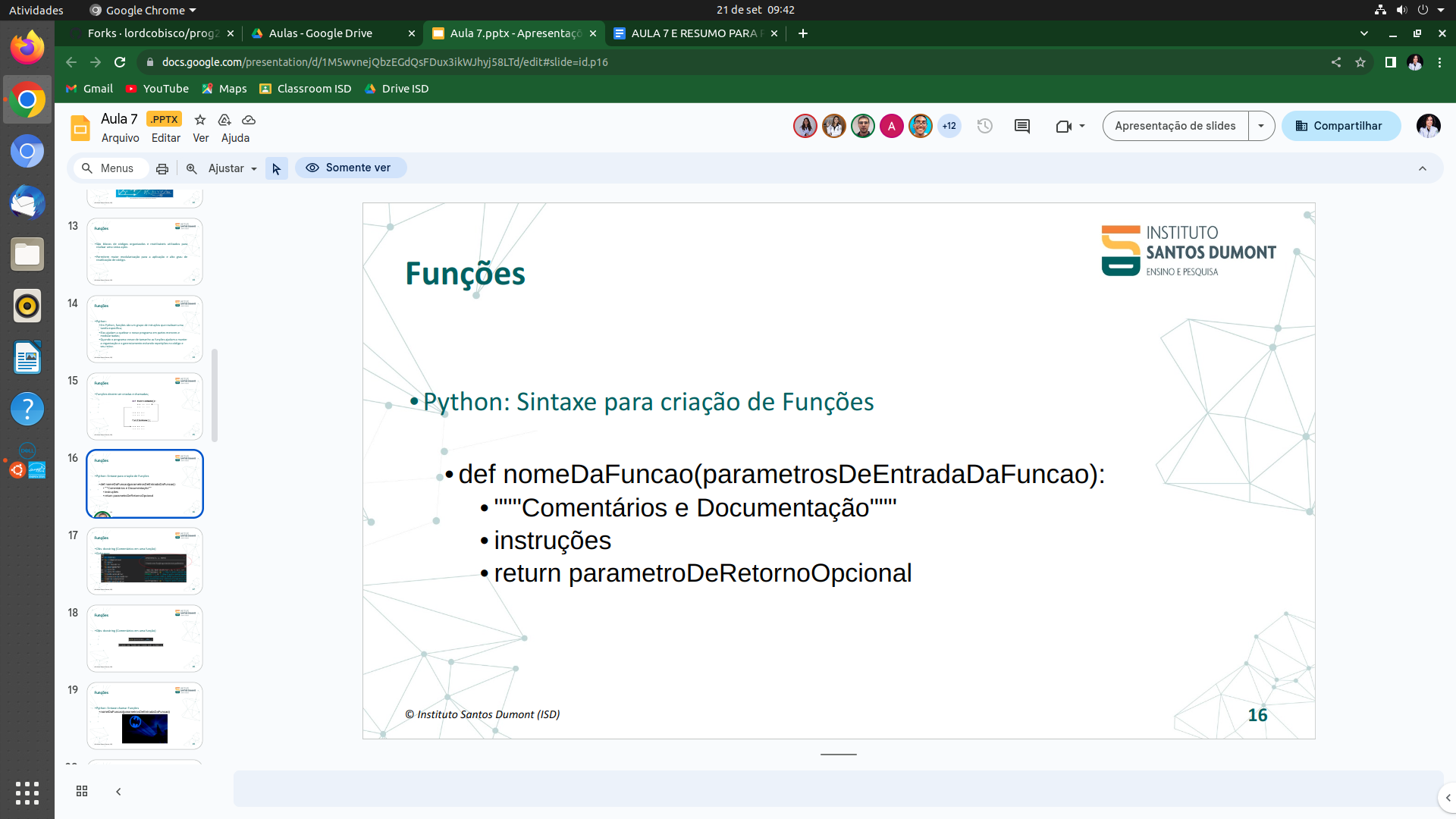
**\* Desvantagens da abstração:**

* O programa tende a ficar mais lento
* Maior nível de raciocínio exigido
* Pode não ser interessantes em aplicações pequenas
* Aumenta o tempo de codificação inicial

**Objetivos de aprendizagem:**

* Definir as estruturas necessárias para criar funções
* Reescrever programas na forma de funções
* Definir classe e sua aplicação
* Reconhecer estruturas da linguagem que utilizam classe

**Funções:**

* São blocos de códigos organizados e reutilizáveis utilizados para realizar uma única ação.
* Formada por entrada, saída e responsabilidade;
* Permitem maior modularização para a aplicação e alto grau de reutilização de código.
* Uma caixinha em que cada um tenha uma responsabilidade.
* Em Python, funções são um grupo de intruções que realizam uma tarefa específica;
* Elas ajudam a quebrar o nosso programa em partes menores e modularizadas;
* Quando o programa cresce de tamanho as funções ajudam a manter a organização e o gerenciamento evitando repetições no código e seu reúso.
* Essa responsabilidade pode estar em diferentes cenários e em várias entradas e saídas.
* Toda função é formada por dois elementos;
* Funções devem ser criadas e chamadas (def functionName(): / functionName();)
* 

**Figura 1:** sintaxe da função em Python

* A função é uma memória e o uso da função é processamento;
* Se for colocar entradas, devem ser colocadas por vírgula. Se for colocar saída, deve ser só uma.
* No condicionamento, a caixa seria a entrada e os dados seriam a saída.

**Classe:**

* A classe utiliza as funções, de forma mais complexa;
* Praticamente tudo que se faz em Python é uma classe (ex: int, float, etc);
* Todo valor em Python é na verdade um objeto;
* Toda classe pode ter funções e características;
* Um objeto possui um estado e uma coleção de métodos que ele pode executar;