**Revisão prova bimestral**

**҉Paradigmas** (é classificado de acordo com as suas funcionalidades) - Aula 1

-> *Imperativo* - Orientada a ações – manipula os valores de uma variável (leitura e atribuição) – Ex: Cobol – V: Eficiente; D: Difícil de ler

-> *Orientada a Objeto* - Problema é analisado quanto aos objetos que o compõem – composto por conjuntos de objetos que juntos “conversam” e juntos constroem um comportamento global – Ex: Java – V: Organização e reutilização de código; D: Menos eficientes

-> *Funcional* - Computação vista como avaliação de funções matemáticas (resolução de problemas a partir de modelos de funções matemáticas) – foco na aplicação de funções para resolver um problema – Ex: Haskell – V: Otimização dos programas; D: Implementações podem ser ineficientes

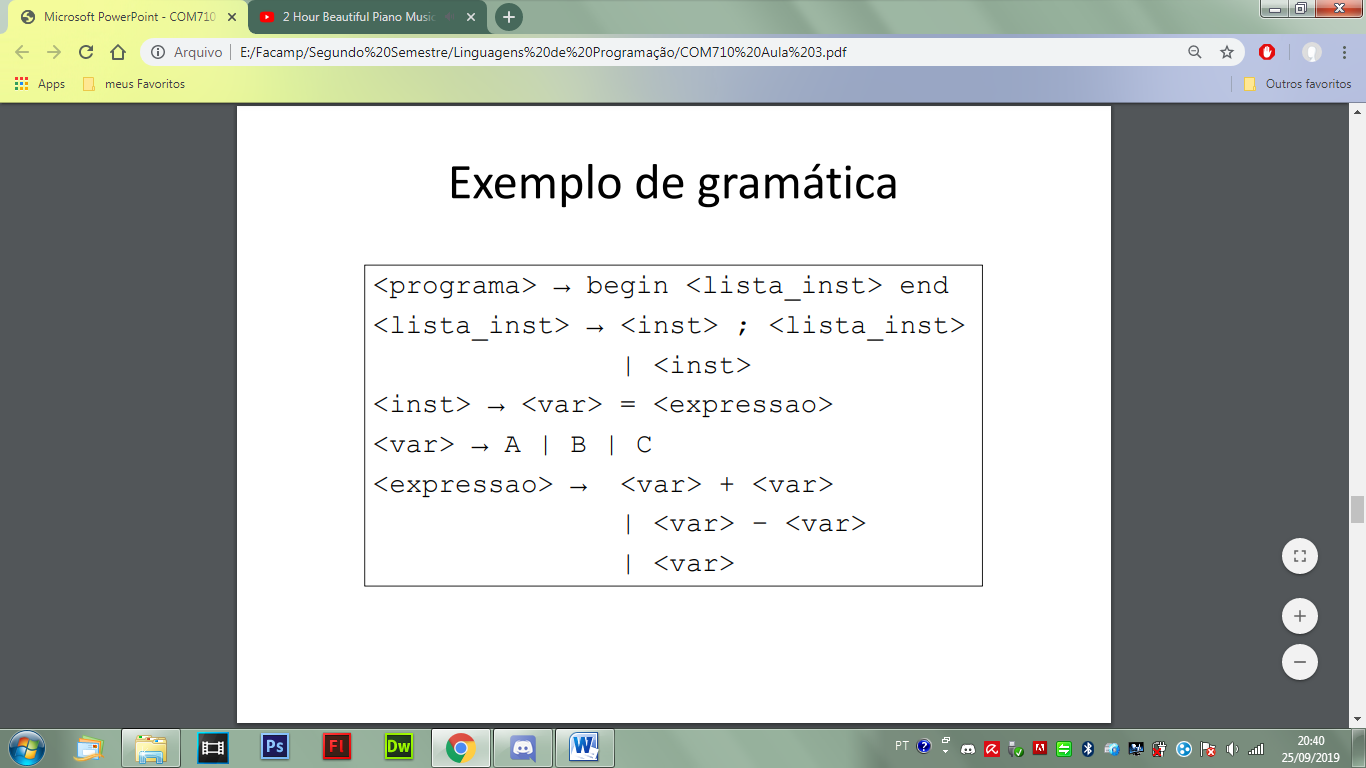
->*Lógica* - É a linguagem baseada na lógica – os programas consistem em declarar fatos que podem ser associações ou regras que produzem fatos deduzidos de outros – Ex: Prolog – V: LP mais próxima do raciocínio lógico; D: Dificuldade em expressar algoritmos complexos

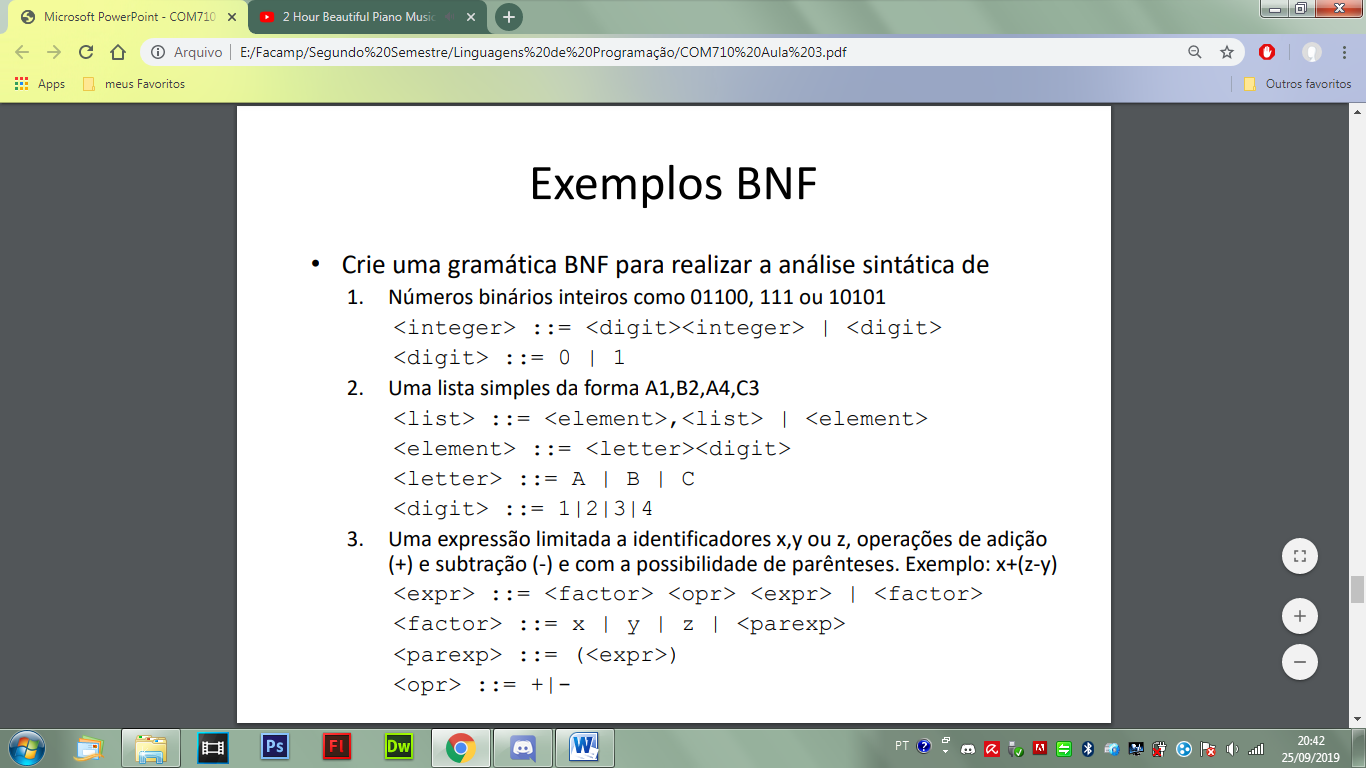
**҉BNF** - Aula 3/Lista 3

-> É a forma da linguagem saber o que é cada coisa

->O que estiver entre <> são não terminais(termo que precisa ser expandido) e os símbolos que não estão entre <> são terminais

->É utilizado para definir linguagens

 ->Ex:

->Ex2:

**҉Conversão de tipo** – Aula 6

-> Mudança forçada de um tipo de variável

Ex: int i = 10;

double d = i;

sout+tab(d);

100000000000000000000000

Ex2: double i = 10.0;

Int d = i;

Sout+tab(d);

Não roda

**҉Conversão implícita(direta)**

->Se um tipo de menor tamanho for atribuído a um parâmetro de função ou variável onde se pede um maior

->Ex: em Java: int n = 10;

double d = n; //d receberá 10.000000

**҉Conversão explicita**

-> Conversão forçada, com perda de precisão

->Quando se deseja converter um tipo maior para um menor (atribuição, parâmetro, etc) deve-se explicitar (casting) a conversão

->Ex: em Java: double d = 10.5;

int n = (int) d; //n receberá 10

**҉Coerção**

->Forçar uma conversão

**҉Escopo** (região onde a variável atua) – Aula 5

->Região do código onde a declaração é visível

Ex: int b = 1;

-

| <- Escopo de vida de c

|

-

If(b<10){

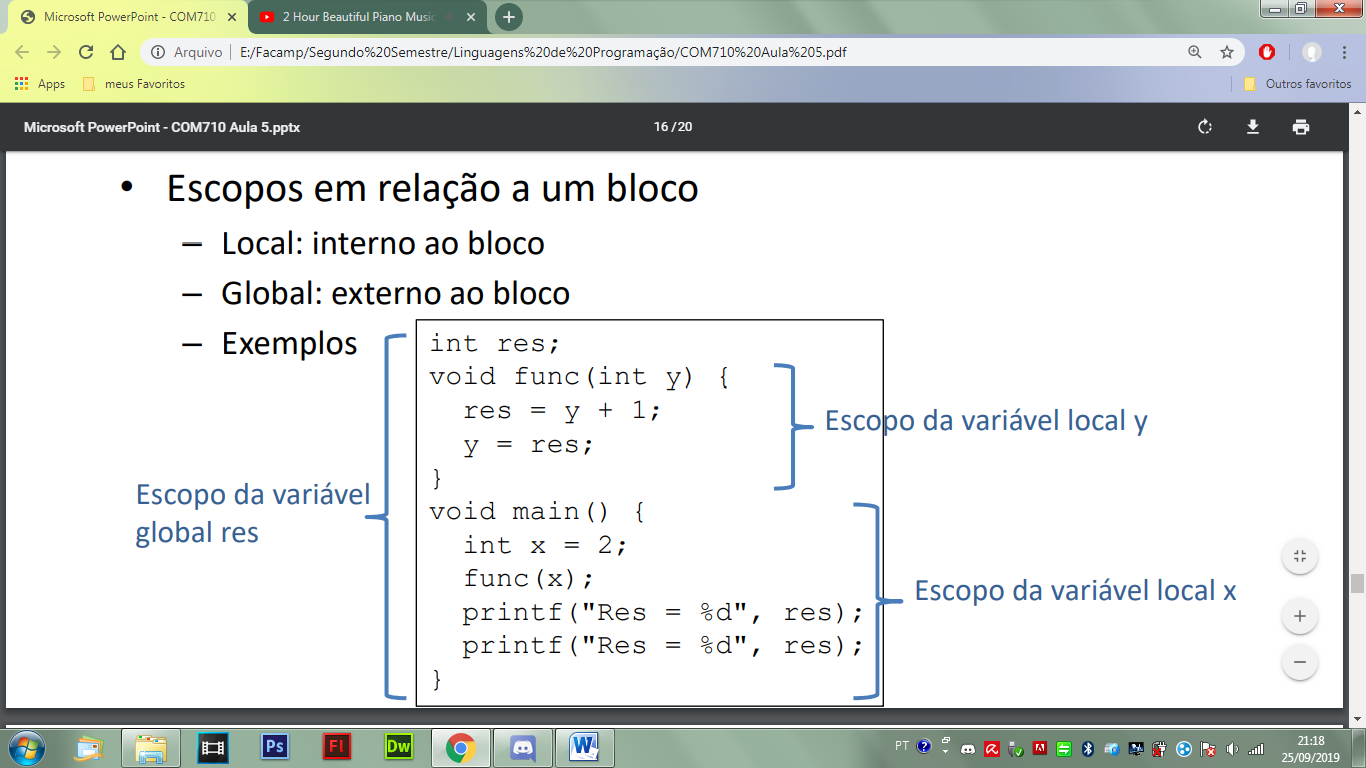
Int c = 20;

}

Sout+tab(c);

->Escopos em relação a um bloco

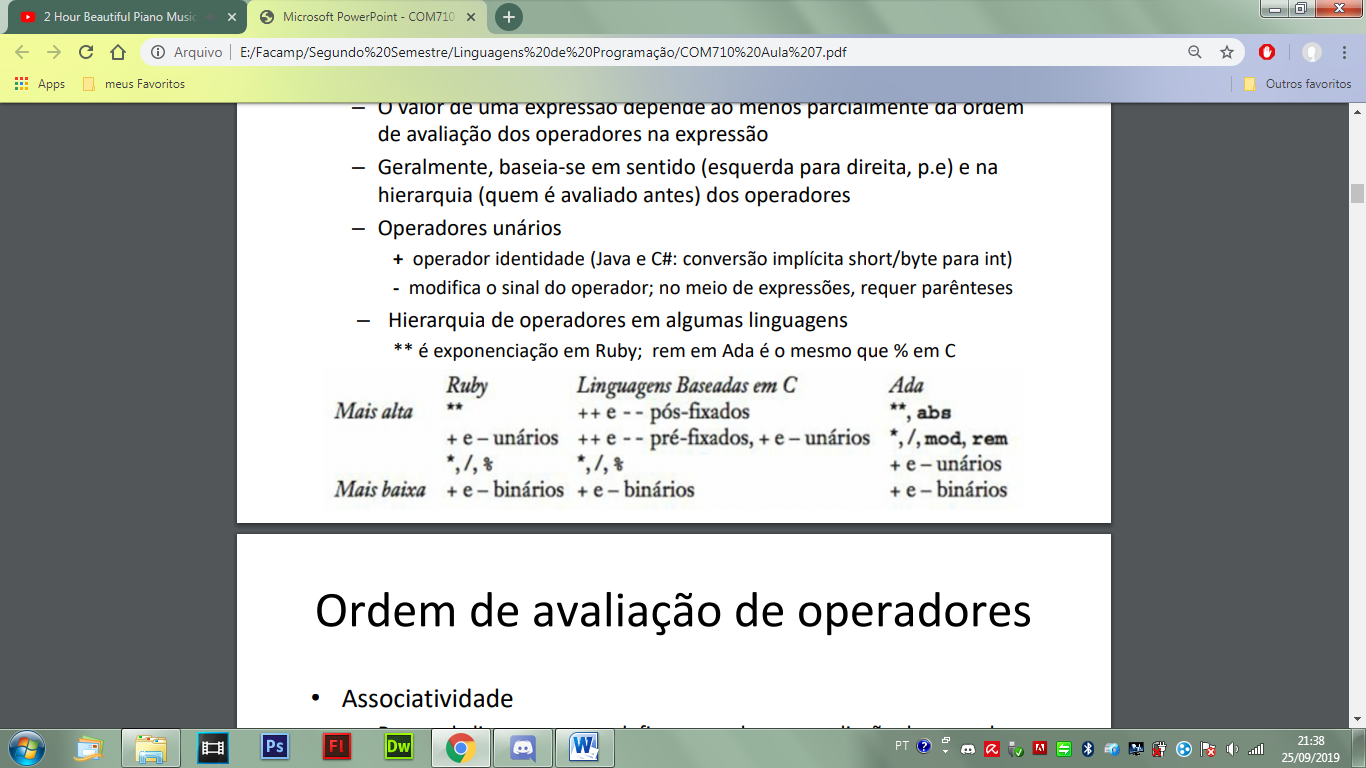
–> Local: interno ao bloco

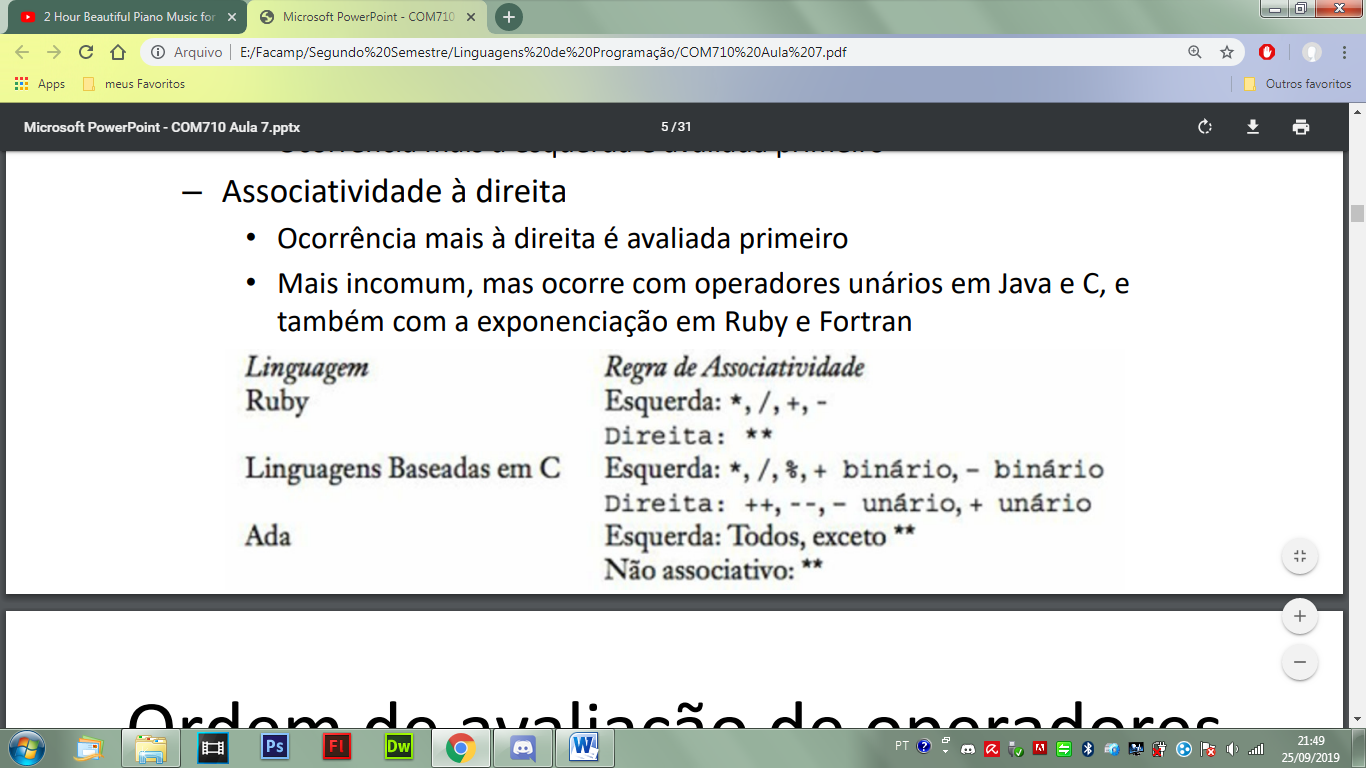
 – >Global: externo ao bloco

**҉Aula 7** **inteira**

->Expressões: São os meios fundamentais de especificar computações em uma linguagem de programação – ordem das avaliações dos operandos e operadores são ditadas pelas regras de associatividade e de precedência da linguagem - diferenças e conversões de tipos e avaliação em curto-circuito

->Expressões aritméticas: Em linguagens de programação consistem em operadores, operandos, parênteses e chamadas a funções – os operadores podem ser: unário[+(operador identidade) e –(modifica o sinal do operador)](que possui um único operando), binário(dois operandos) e ternário(três operandos)

->Ordem de avaliação dos operandos: O valor da expressão depende da ordem de avaliação dos operadores na expressão(geralmente da direita para a esquerda, ou pela hierarquia)

 ->Associatividade: Regras que definem a ordem da avaliação dos operadores de mesmo nível de precedência, a associatividade pode ser tanto pra direita quanto pra esquerda

->Expressões condicionais: Forma comprimida de expressar uma expressão if-then-else – forma: expressão\_1 ? expressão\_2 : expressão\_3 – Ex: average = (count == 0) ? 0 : sum / count;

->Efeitos colaterais funcionais: Efeito colateral de uma função ocorre quando ela modifica um de seus parâmetros ou uma variável global – Exemplo em C: int a = 5;

int fun1() {

a = 17;

return 3; }

void main() {

a = a + fun1(); }

->Transparência referencial: substituições no código que não alteram o resultado final

->Sobrecarga de operadores: Uso de um operador para múltiplos propósitos

->Expressões relacionais: Combinam operandos com operadores relacionais