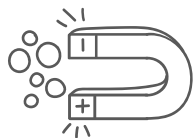


# Uma breve história

Pascal – Object Pascal – Delphi





“Não existe a MELHOR  
linguagem de  
desenvolvimento e sim  
a que melhor se  
encaixa no seu  
projeto/estratégia”



# Pascal – Object Pascal - Delphi

Você já ouviu falar?

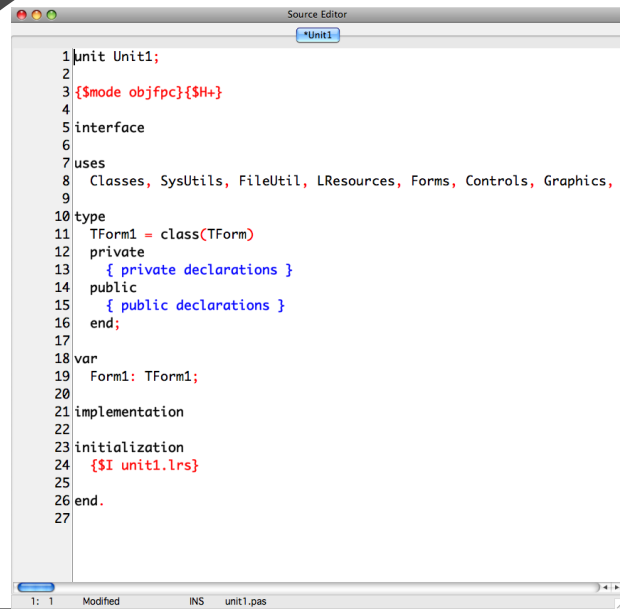
Bem, mal, mais ou menos?  
Conte-me mais...

```
procedure TForm1.Button1Click(Sender: TObject);
type
  TDate = record
    Day: Integer;
    Month: Integer;
    Year: Integer;
  end;

var OrderDate:TDate;
begin
  with OrderDate do
    if Month = 12 then
      begin
        Month := 1;
        Year := Year + 1;
      end
    else
      Month := Month + 1;
    end;
end.
```



# 01. História



```
1 unit Unit1;
2
3 {$mode objfpc}{$H+}
4
5 interface
6
7 uses
8   Classes, SysUtils, FileUtil, LResources, Forms, Controls, Graphics,
9
10 type
11   TForm1 = class(TForm)
12   private
13     { private declarations }
14   public
15     { public declarations }
16   end;
17
18 var
19   Form1: TForm1;
20
21 implementation
22
23 initialization
24   {$I unit1.lrs}
25
26 end.
27
```

# História



## Pascal

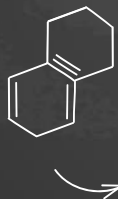
Surgimento nos anos 70  
Programação Estruturada  
Programação Procedural



## Object Pascal

Ramificação do Pascal  
Suporte a Orientação a Objetos

IDE: RAD Studio Delphi ou Lazarus



## Delphi

A partir da versão 2009 não é mais  
considerado Object Pascal



## Linguagem Compilada

Prós: Não precisa ter uma máquina  
virtual por trás

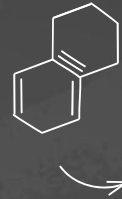
Contras: A maioria do  
gerenciamento fica na mão do Dev

# Delphi



## Borland

Empresa fundada em 1983  
em Austin, Texas



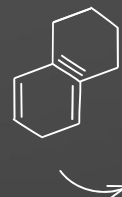
## CodeGear

Fundada em 2006 é uma  
subdivisão da Borland para ficar  
exclusivamente com a produz de  
ferramentas de desenvolvimento



## Embarcadero

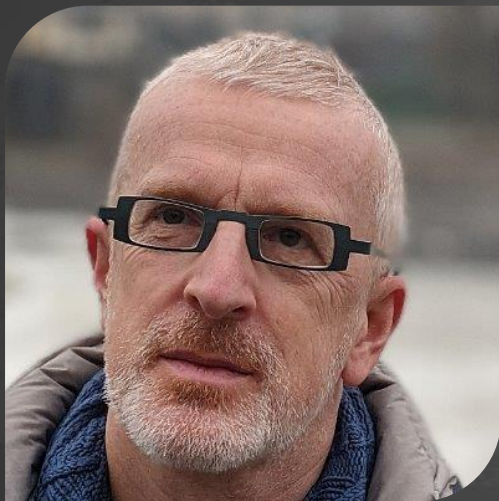
Fundada em 1993.  
Meados de 2007/2010 adquire a  
CodeGear e lança várias versões do  
Delphi.



## Idera

Empresa de softwares B2B,  
ferramentas de banco de dados e  
ferramentas de desenvolvimento  
de aplicativos.

# Referência



Marco Cantù

Um dos grandes embaixadores do  
Delphi e pascal

Site: <https://www.marcocantu.com/>

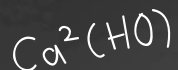


# 02.

## Evoluções da Linguagem







# Algumas versões significativas



## Delphi 5

Meados de 98  
Sistemas Legados ainda  
são mantidos nessa  
versão

## Delphi 7

Meados de 2002  
Sistemas Legados ainda  
são mantidos nessa  
versão

## Delphi 2009/2010

Evolução na Linguagem  
com algumas features  
(Generics)

## Delphi XE 2

2011  
Suporte a  
desenvolvimento para  
MacOSx e IOS

## Delphi XE 5

2013  
Suporte a  
desenvolvimento para o  
Android

## Delphi 11.1 - Alexandria

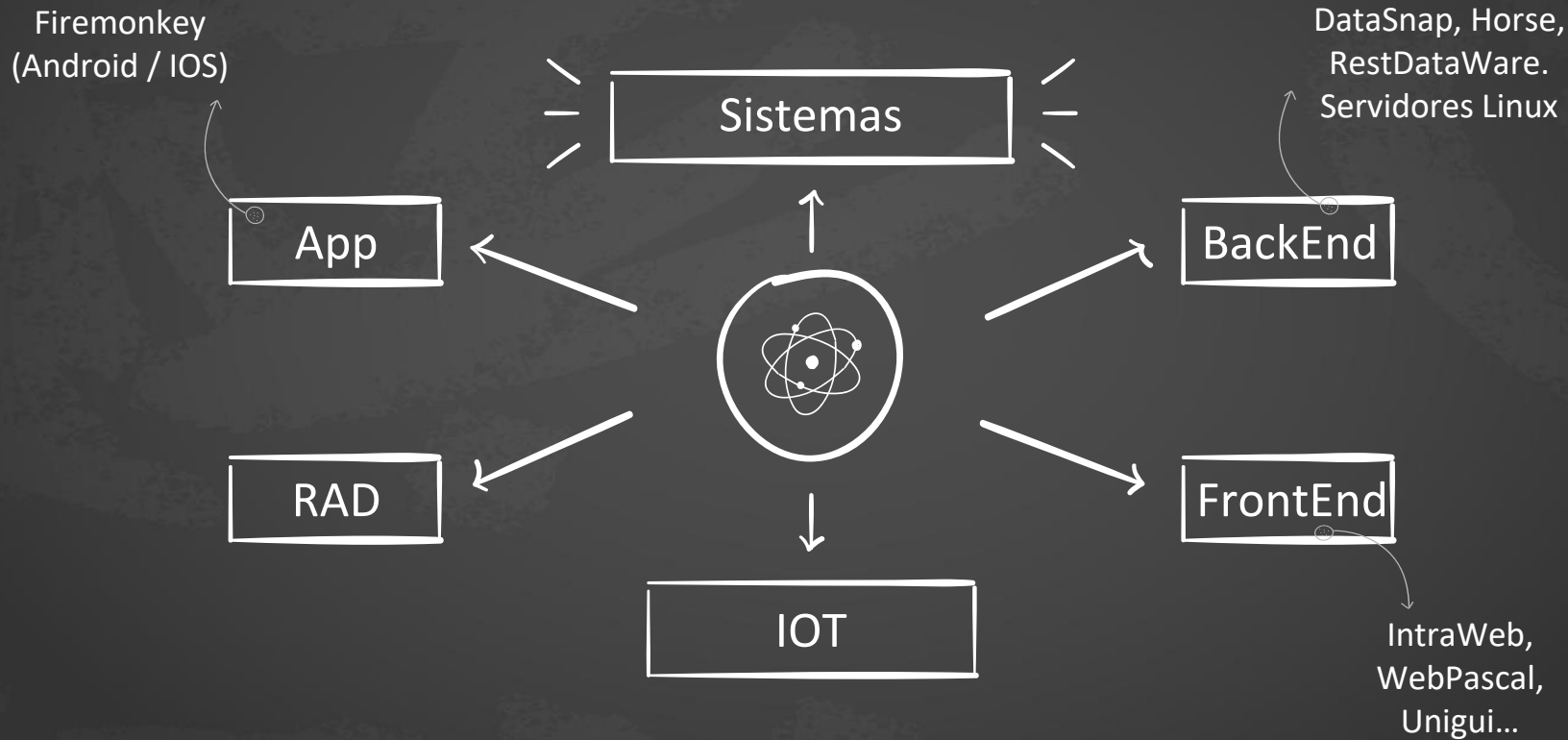
2021  
Melhorias significativas  
na IDE, compilação e etc...



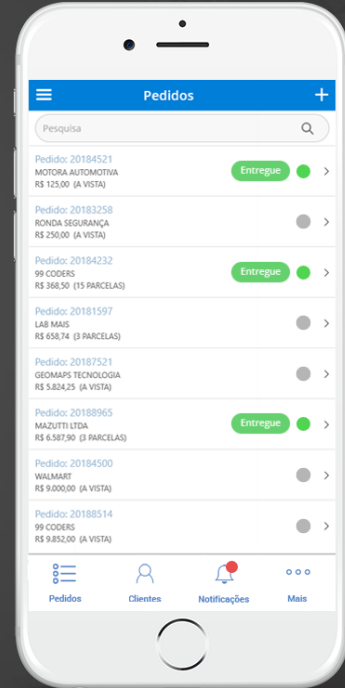
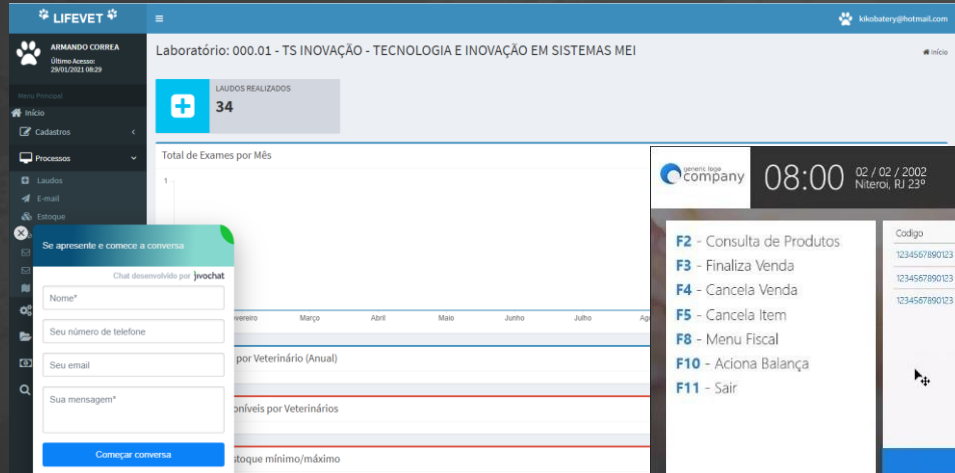
# E no Mundo?

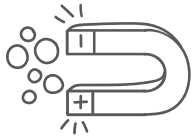


# O que posso Desenvolver em Delphi



# Alguns exemplos reais...





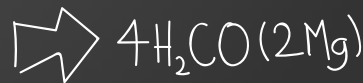
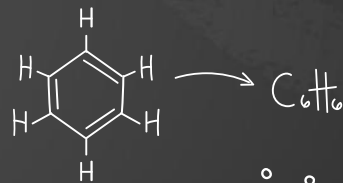
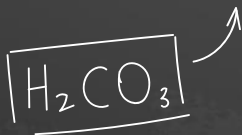
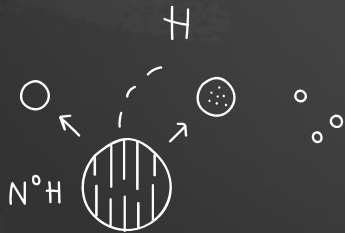
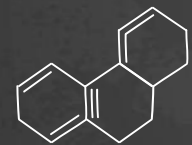
# Dúvidas ?

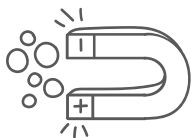




# Sintaxe da Linguagem

Delphi





“A vida não se acaba  
quando deixamos de  
viver e sim quando  
deixamos de buscar  
algo nela!”



# Arquivos de um projeto no Delphi

Abaixo segue alguns arquivos presente em um projeto:

- \*.pas – Código fonte
- \*.dfm – Formulário
- \*.res – Arquivo de recursos para guardar ícones, arquivos e outros
- \*.dpr – Arquivo do programa principal
- \*.bpl – Pacote de unidades e formulários – Delphi
- \*.dll – Biblioteca com funções e procedimentos
- Todo .dfm – obrigatoriamente segue um .pas com o mesmo nome.
- \*.pas podemos ter vários sem ter a necessidade de ter um \*.dfm (formulário)

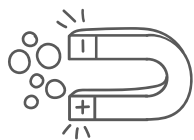
Unit – Unidade de codificação

Form – Formulário / Tela de exibição

```
procedure TForm1.Button1Click(Sender: TObject);
type
  TDate = record
    Day: Integer;
    Month: Integer;
    Year: Integer;
  end;

var OrderDate: TDate;
begin
  with OrderDate do
    if Month = 12 then
      begin
        Month := 1;
        Year := Year + 1;
      end
    else
      Month := Month + 1;
    end;
end;
end.
```





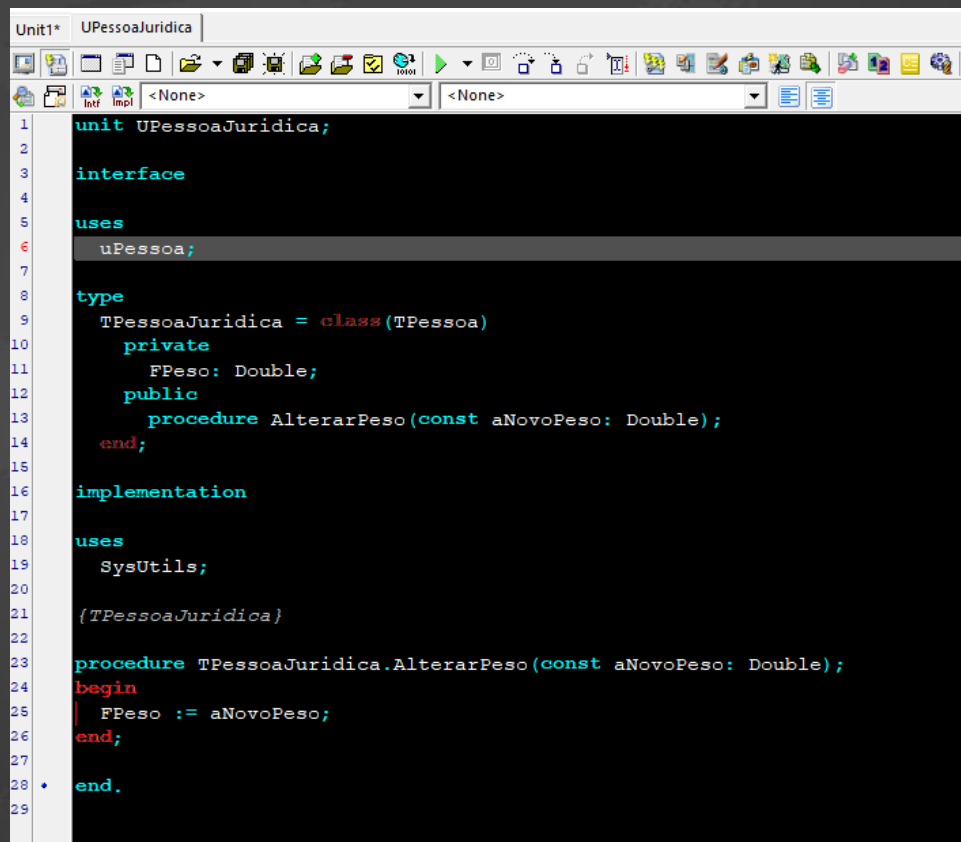
Vamos carregar  
o Delphi!!!



# Bloco de Código em Delphi - Seções

- **Unit** – Nome da Unidade / Arquivo.
- **Interface** – Seção onde declaramos os tipos de dados e unidades “importadas” para a Unit
- **Uses** – Unidades “importadas” para serem usadas na seção interface e implementation
- **Type** – Bloco reservado para a definição dos tipos de dados da Unit (Classes, Objetos, Record e etc...)
- **Implementation** – Corpo das Rotinas e dos Métodos dos tipos definidos da seção Type
- **Uses** – Unidades “importadas” para serem usadas somente na seção implementation – Corpo dos Métodos
- **Var** – Declaração de variáveis global para a Unit – “Sempre escapar”

# Bloco de Código em Delphi - Seções



```
Unit1*  UPessoaJuridica
1      unit UPessoaJuridica;
2
3      interface
4
5      uses
6          uPessoa;
7
8      type
9          TPessoaJuridica = class(TPessoa)
10             private
11                 FPeso: Double;
12             public
13                 procedure AlterarPeso(const aNovoPeso: Double);
14             end;
15
16      implementation
17
18      uses
19          SysUtils;
20
21      {TPessoaJuridica}
22
23      procedure TPessoaJuridica.AlterarPeso(const aNovoPeso: Double);
24      begin
25          FPeso := aNovoPeso;
26      end;
27
28      end.
```

# Bloco de Código em Delphi – Rotinas e Métodos

O Delphi segue um padrão de codificação:

Rotinas e/ou métodos

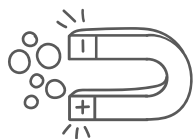
```
procedure <nome>(<parametro1: Tipo>; <parametroN: Tipo>;  
var/const  
    <variavel1: Tipo>;  
begin  
end;
```

```
function <nome>(<parametro1: Tipo>; <parametroN: Tipo>):  
    <TipoDeRetorno>;  
var/const  
    <variavel1: Tipo>;  
begin  
    Result := <valor>;  
end;
```

# Bloco de Código em Delphi – Rotinas e Métodos

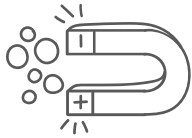
Qual a diferença entre Procedure e Function?

- \*Function sempre retorna algum valor.
- \*Procedure não retorna valor.
- \*Var poderá sofrer alteração de valor.
- \*Const nunca sofre alterações.



# Nosso primeiro programa





# Dúvidas?

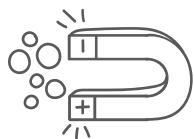




# Tipos de Dados

Delphi





“Tudo o que um sonho  
precisa para ser  
realizado é alguém que  
acredite que ele possa  
ser realizado.”



# Tipos de Dados

Alguns tipos de dados principais suportado pelo Delphi:

## Tipos Ordinais

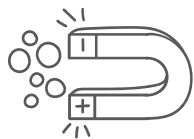
- Char (um caractere)
- Byte (0..255)
- Word (0..65.535)
- Integer (-32768..32767)
- LongInt (-2.147.843.648..2.147.483.647)
- Boolean (True/False)

## Tipos Reais

- Single
- Double
- Extended
- Currency

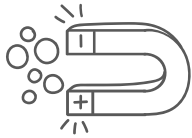
## Outros Tipos

- String
- Array
- Record
- Enumerados



Vamos carregar  
o Delphi!!!





Dúvidas?



# Operadores

Delphi



# Operadores

## Operadores Lógicos

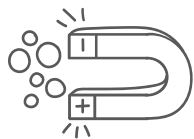
- And
- Or
- Not

## Operadores Relacionais

- = (igual)
- <> (diferente)
- < (menor)
- > (maior)
- <= (menor igual)
- >= (maior igual)
- In (entre)

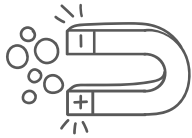
## Operadores Matemáticos

- \* (multiplicador)
- + (soma ou concatenação de String)
- - (subtração)
- / (divisão)
- Div (divisão inteira)
- Mod (resto da divisão)



Vamos carregar  
o Delphi!!!





# Dúvidas?

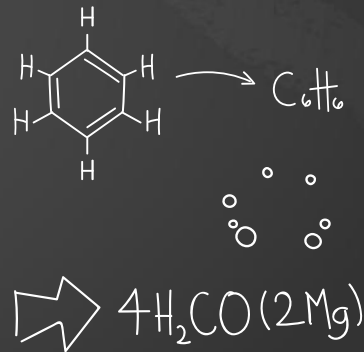




# Comandos Condicionais

Delphi

# Delphi



# Comando “If”

➤ If <condição> then  
begin  
    bloco verdadeiro;  
end  
else  
begin  
    bloco falso;  
end;

➤ If <condição> then  
begin  
    bloco verdadeiro;  
end  
else  
    bloco falso;

➤ if <condição> then  
    bloco verdadeiro; //<- apenas uma linha

## Comando "If"

```
1 procedure TForm6.Button1Click(Sender: TObject);  
2   var  
3     xMaioridade: Boolean;  
4     xIdade: Byte;  
5   begin  
6     xIdade      := 18;  
7     xMaioridade := False;  
8  
9     if xIdade >= 18 then  
10      xMaioridade := True;  
11   end;
```

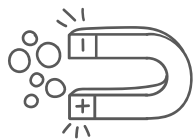
# Comando “Case”

- case <valorOrinal> of
  - <valor1>: bloco para o tipo <valor1>;
  - <valor2>: bloco para o tipo <valor2>;
  - <valor3>: bloco para o tipo <valor3>;
  - else bloco para outro tipo não mapeado;end;
- \*Utilizamos o case para condições extensas e por ficar uma leitura e entendimento mais agradável – boas práticas e clean code.

# Comando “Case”

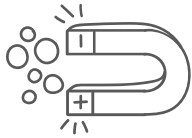
```
type  
TMeusEstados = (tpSaoPaulo, tpRioDeJaneiro, tpSantaCatarina);
```

```
1 function TForm6.RetornarNomeEstado(aMeuEstado: TMeusEstados): String;  
begin  
  case aMeuEstado of  
    tpSaoPaulo:  
      Result := 'São Paulo';  
    tpRioDeJaneiro:  
      Result := 'Rio de Janeiro';  
    tpSantaCatarina:  
      Result := 'Santa Catarina';  
  end;  
end;
```



Vamos carregar  
o Delphi!!!





Dúvidas?



# Comandos de Repetição

Delphi





# Comando “For”

➤ for **variavel** := <inicio> **to** <fim> do  
begin  
...  
  <break, continue>  
end;

\*Muito utilizado para fazer algum tipo de cálculo.  
\*Percorrer listas.

➤ for **variavel** := <inicio> **downto** <fim> do  
begin  
...  
  <break, continue>  
end;

# Comando “For”

```
for I := 0 to (FListaNotas.Count - 1) do
begin
    if (I = 0) then
        xMensagem := xMensagem + Format('Notas Fiscais processadas: %d', [FListaNotas[i].NumeroNota])
    else
        xMensagem := xMensagem + Format(', %d', [FListaNotas[i].NumeroNota]);
    end;
end;

for I := Length(aTexto) to downto 1 do
    if (not (aTexto[I]) in aConsidera) then
        raise Exception.Create(Format('Caracter "%s" não é permitido.', [aTexto[I]]));
```

# Comando “While”

➤ while <condição> do  
begin  
...  
<break, continue>  
end;

\*Muito utilizado para “varrer” uma tabela.  
\*Percorrer listas.

# Comando “While”

```
while (not xQry.EOF) do
begin
    xNome := xQry.FieldName('Nome').AsString;
    xIdade := xQry.FieldName('Idade').AsInteger;

    Self.ValidarAlgumaCoisa(xNome, xIdade);

    xQry.Next;
end;
```

# Comando “Repeat”

➤ repeat

...

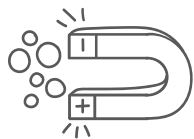
<break, continue>

until (<condição>);

\*Sempre vai passar pelo menos uma única vez.

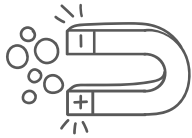
# Comando “Repeat”

```
procedure TForm6.Button2Click(Sender: TObject);  
var  
    xSoma: Integer;  
begin  
    xSoma := 0;  
  
    repeat  
        xSoma := xSoma + 1;  
    until (xSoma = 10);  
  
    ShowMessage (xSoma.ToString);  
end;
```



Vamos carregar  
o Delphi!!!





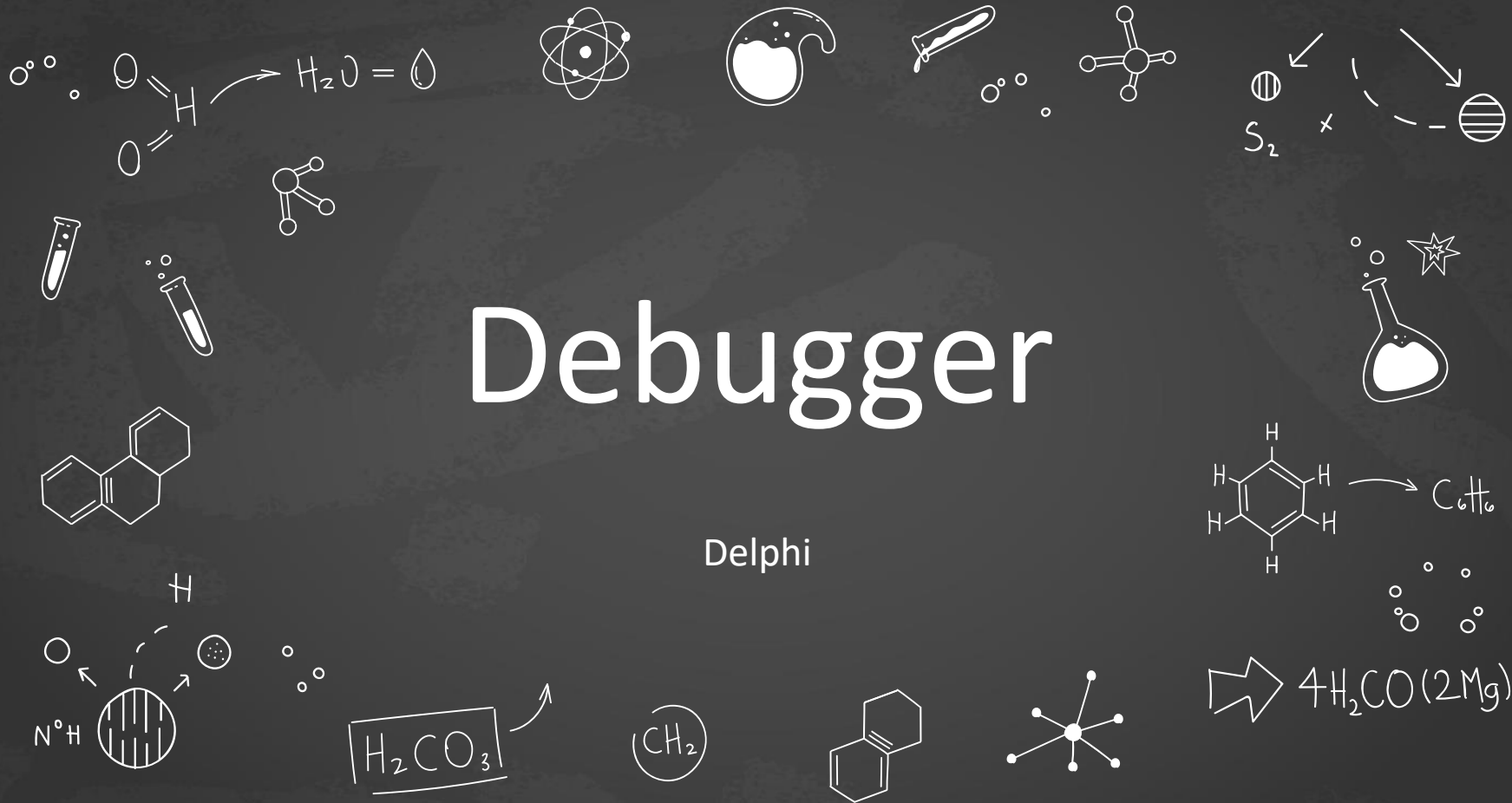
Dúvidas?





# Debugger

Delphi



# Alguns atalhos

- F7 – Debug – Trace Into
- F8 – Debug – Step Over
- F9 – Run
- Ctrl + F9 – Compile Project
- F11 – Object Inspector
- F12 – Toggle Form/Unit
- Ctrl+F12 – Search Units
- Shift+F12 – Search Forms

## Debug Windows

- BreakPoints – Ctrl + Alt + B
- Call Stack – Ctrl + Alt + S
- Watches – Ctrl + Alt + W
- Threads – Ctrl + Alt + T

# Debugger

The screenshot displays the Delphi IDE's debugger interface. The main window shows the source code of `Unit6.pas`, specifically the `TForm6.Button2Click` procedure. The code is as follows:

```
if xIdade >= 18 then  
    xMaioridade := True;  
end;  
  
procedure TForm6.Button2Click(Sender: TObject);  
var  
    xSoma: Integer;  
begin  
    xSoma := 0;  
    repeat  
        xSoma := xSoma + 1;  
    until (xSoma = 10);  
    ShowMessage(xSoma.ToString);  
end;  
  
function TForm6.RetornarNomeEstado(aMeuEstado: TMeusEstados): String;  
begin  
    case aMeuEstado of  
        tpSaoPaulo: tpSaoPaulo;
```

The debugger is currently paused at line 47, which is the `tpSaoPaulo` case in the `RetornarNomeEstado` function. The `Call Stack` on the left shows the sequence of calls leading to the current state. The `Watch List` shows the variable `xSoma` with a value of 2. The `Local Variables` pane shows the `tpSaoPaulo` variable. The `Breakpoints` pane at the bottom shows a breakpoint set at line 47 of `Unit6.pas`.

**Call Stack - Thread 11208**

- Unit6.TForm6.Button2Click(\$2F2EEA0)
- Vcl.Controls.TControl.Click
- Vcl.StdCtrls.TCustomButton.Click
- Vcl.StdCtrls.TCustomButton.CNCommand(???)
- Vcl.Controls.TControl.WndProc(48401, 2158, 985)
- Vcl.Controls.TWinControl.WndProc(48401, 2158)
- Vcl.StdCtrls.TButtonControl.WndProc(48401, 215)
- Vcl.Controls.TControl.Perform(???; ???; 985198)

**Watch List - Thread 11208**

Watch Name	Value
xSoma	2

**Local Variables - Thread 11208**

Variable Name	Value
tpSaoPaulo	tpSaoPaulo

**Breakpoints**

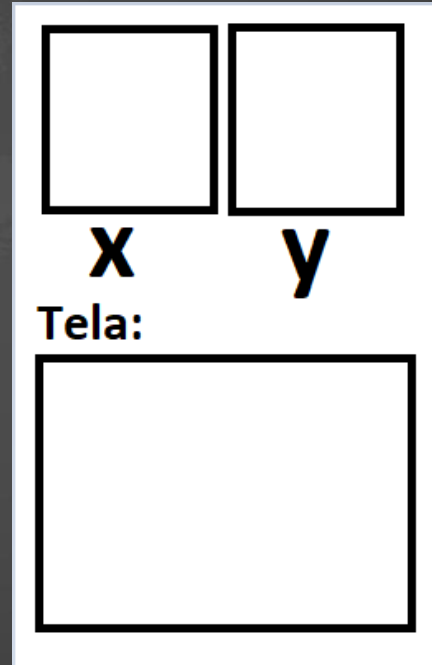
Filename/Address	Line/Leng...	Condition	Thread	Action	Pass Count	Group
Unit6.pas	47			Break	0	

# Teste de Mesa - While

```
var
  x,y: Integer;

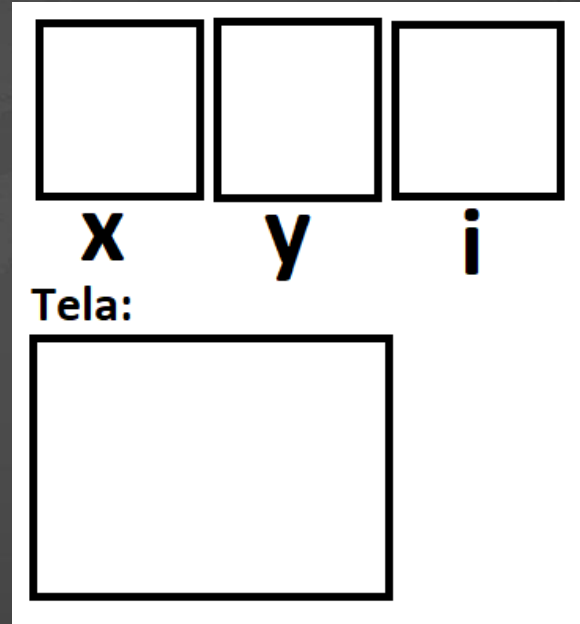
x := 5;
y := 0;

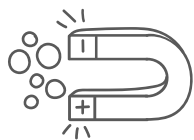
while (x > 2) do
begin
  writeln(x);
  y := y + x;
  x := x - 1;
end;
```



# Teste de Mesa - For

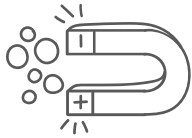
```
var  
  x,y,i: Integer;  
  
x := 4;  
y := x + 2;  
  
for i := 0 to i < x do  
begin  
  writeln (x + ' ' + y);  
  y := y + i;  
end;
```





Vamos carregar  
o Delphi!!!





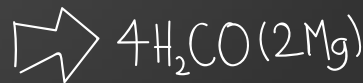
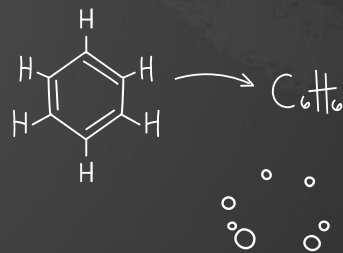
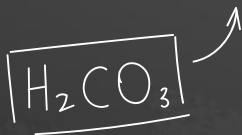
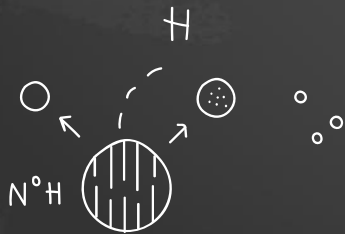
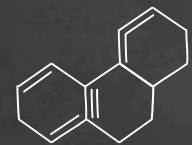
Dúvidas?



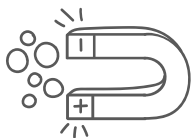


# Procedures e Functions

Delphi







“Se você não sabe para  
onde ir, qualquer  
caminho serve”



# Procedures e Functions

- Procedures
- Functions
- Parâmetros de Referência
- Parâmetros Constantes
- Parâmetros de Array Aberto
- Parâmetros de Array Aberto de Tipo Variant
- Sobrecarga de Funções (Function Overloading)
- Parâmetros Default
- Métodos / Rotinas
- Declaração Forward

# Procedures e Functions

- Em Pascal, uma rotina pode assumir duas formas: um procedimento ou uma função. Em teoria, um procedimento é uma operação que você pede ao computador para executar, uma função é uma computação que retorna um valor. Essa diferença é enfatizada pelo fato de uma função ter um resultado, um valor de retorno, enquanto um procedimento não. Ambos os tipos de rotinas podem ter vários parâmetros de tipos de dados especificados.

# Procedures e Functions

```
procedure <nome>(<parametro1: Tipo>; <parametroN: Tipo>;  
var/const  
    <variavel1: Tipo>;  
begin  
end;
```

```
function <nome>(<parametro1: Tipo>; <parametroN: Tipo>):  
    <TipoDeRetorno>;  
var/const  
    <variavel1: Tipo>;  
begin  
    Result := <valor>;  
end;
```

# Procedures e Functions

```
procedure Hello;  
begin  
  writeln('Hello World!');  
end;
```

Como Usamos:

```
Hello;  
  
x := Double(100);
```

```
function Double (value: Integer): Integer;  
begin  
  Double := Value * 2;  
end;
```

```
function Double (value: Integer): Integer;  
begin  
  Result := Value * 2;  
end;
```

# Parâmetros de Referência

- As rotinas Pascal permitem a passagem de parâmetros por valor e por referência. Passar parâmetros por valor é o padrão: o valor é copiado na pilha e a rotina usa e manipula a cópia, não o valor original.
- Passar um parâmetro por referência significa que seu valor não é copiado na pilha no parâmetro formal da rotina. Em vez disso, o programa se refere ao valor original, também no código da rotina. Isso permite que o procedimento ou função altere o valor do parâmetro. A passagem de parâmetro por referência é expressa pela palavra-chave **var**.

# Parâmetros de Referência

```
procedure DoubleTheValue (var Value: Integer)
```

```
begin
```

```
    Value := Value * 2;
```

```
end;
```

```
procedure DoubleTheValue (Value: Integer)
```

```
begin
```

```
    Value := Value * 2;
```

```
end;
```

## Situação A

```
var
```

```
    x: Integer;
```

```
begin
```

```
    x := 10;
```

```
    DoubleTheValue (x);
```

```
    ShowMessage(IntToStr(x));
```

```
end;
```

# Parâmetros Constantes

- Como alternativa aos parâmetros de referência, você pode usar um parâmetro const. Como você não pode atribuir um novo valor a um parâmetro constante dentro da rotina, o compilador pode otimizar a passagem de parâmetros. O compilador pode escolher uma abordagem semelhante aos parâmetros de referência, mas o comportamento permanecerá semelhante aos parâmetros de valor, pois o valor original não será afetado pela rotina.



# Parâmetros Constantes

```
function DoubleTheValue (const Value: Integer): Integer;  
begin  
    Value := Value * 2; //compiler error  
    Result := Value;  
end;
```

# Parâmetros de Array Aberto

- Uma função ou procedimento Pascal sempre tem um número fixo de parâmetros. No entanto, existe uma maneira de passar um número variável de parâmetros para um runtime usando uma matriz aberta. A definição básica de um parâmetro de matriz aberta é a de uma matriz aberta tipada. Isso significa que você indica o tipo do parâmetro, mas não sabe quantos elementos desse tipo o array terá.

# Parâmetros de Array Aberto

```
function Sum (const A: Array of Integer): Integer;
```

```
var
```

```
  I: Integer;
```

```
begin
```

```
  Result := 0;
```

```
  for I := Low (A) to High(A) do
```

```
    Result := Result + A[I];
```

```
end;
```

# Parâmetros de Array Aberto de Tipo Variant

- Além das matrizes abertas tipadas, o Delphi permite definir matrizes abertas de tipos variantes ou não tipadas. Esse tipo especial de array tem um número indefinido de valores, o que pode ser útil para passar parâmetros.
- Tecnicamente, o array de construção de const permite que você passe um array com um número indefinido de elementos de diferentes tipos para uma rotina de uma só vez.

# Parâmetros de Array Aberto de Tipo Variant

RTL

```
function Format (const Format: String; const Args: array of const) : String;  
begin  
...  
end;
```

Como usamos:

```
N := 20;  
  
writeln (Format('Total: %d', [N]));  
  
writeln (Format('Int: %d, Float: %f', [N, 12.4]));
```

# Sobrecarga de Funções – Function Overloading

A ideia de sobrecarga é simples: o compilador permite definir duas funções ou procedimentos usando o mesmo nome, desde que os parâmetros sejam diferentes. Verificando os parâmetros, de fato, o compilador pode determinar qual das versões da rotina você deseja chamar. Considere esta série de funções extraídas da unidade Math da VCL:

```
function Min (A, B: Integer): Integer; overload;  
function Min (A, B: Int64): Int64; overload;  
function Min (A, B: Single): Single; overload;  
function Min (A, B: Double): Double; overload;  
function Min (A, B: Extended): Extended; overload;
```

Min (10, 20) – O programa executará a primeira função.

# Parâmetros Default

Outro recurso relacionado à sobrecarga, é que você pode atribuir um valor padrão ao parâmetro ou parâmetros de uma função, para poder chamar a função com ou sem o parâmetro. se o parâmetro estiver ausente na chamada, ele assumirá o valor padrão.

```
procedure NewMessage (Msg: string; Caption: String = 'Message'; Separator: String = SLineBreak);  
begin  
  write (caption);  
  write (': ');  
  write (Msg);  
  write (Separator);  
end;
```

```
NewMessage('Something wrong here!');  
NewMessage('Something wrong here!', 'Attention');  
NewMessage('Hello', 'Message', '--');
```

# Métodos e Rotinas

## O que é um método?

Um método é um tipo especial de função ou procedimento relacionado a um tipo de dados, uma classe. Em Delphi, toda vez que tratamos de um evento, precisamos definir um método, geralmente um procedimento. Em geral, no entanto, o termo método é usado para indicar funções e procedimentos relacionados a uma classe.

```
procedure TForm1.Button1Click(Sender: TObject);  
begin  
    //Seu código aqui  
end;
```



# Métodos e Rotinas

## O que é uma rotina?

Ao contrário de um método uma rotina não está associada a um tipo de dados e/ou uma classe.

```
procedure CalcularSaldo;  
begin  
  //Seu código aqui  
end;
```

# Declaração Forward

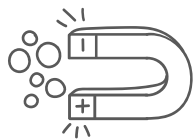
Quando você precisa usar um identificador (de qualquer tipo), o compilador já deve ter visto algum tipo de declaração para saber a que se refere o identificador. Por esse motivo, você geralmente fornece uma declaração completa antes de usar qualquer rotina. No entanto, há casos em que isso não é possível. Se o procedimento A chama o procedimento B, e o procedimento B chama o procedimento A, quando você começar a escrever o código, você precisará chamar uma rotina para a qual o compilador ainda não viu uma declaração.

# Declaração Forward

```
procedure Hello; forward;  
procedure DoubleHello; forward;
```

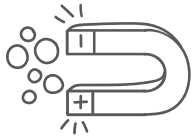
```
procedure Hello;  
begin  
  if MessageDlg('Do you want a double message?', mtConfirmation, [mbYes, mbNo], 0) = mrYes then  
    DoubleHello  
  else  
    ShowMessage ('Hello');  
end;
```

```
procedure DoubleHello;  
begin  
  Hello;  
  Hello;  
end;
```



Vamos carregar  
o Delphi!!!





# Dúvidas?





# Arrays e Records

Delphi

# Arrays Types

Os tipos de matriz definem listas de um número fixo de elementos de um tipo específico. Você geralmente usa um índice entre colchetes para acessar um dos elementos da matriz.

Type

```
TDayTemperatures = array [1..24] of Integer;
```

```
TMonthTemps = array [1..31] of TDayTemperatures;
```

Existe a possibilidade de termos matrizes dinâmicas.

Type

```
TMatriz = array of String;
```

# Record Types

Os tipos de registro definem coleções fixas de itens de diferentes tipos. Cada elemento, ou campo, tem seu próprio tipo. A definição de um tipo de registro lista todos esses campos, dando a cada um, um nome que você usará posteriormente para acessá-lo.

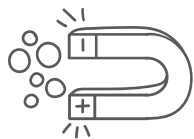
Type

```
TMyDate = record  
  Year: Integer;  
  Month: Byte;  
  Day: Byte;  
end;
```

var

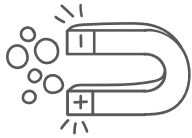
```
  xBirthDay: TMyDate;  
begin  
  xBirthDay.Year := 2022;  
  xBirthDay.Month := 9;  
  xBirthDay.Day := 29;  
end;
```





Vamos carregar  
o Delphi!!!





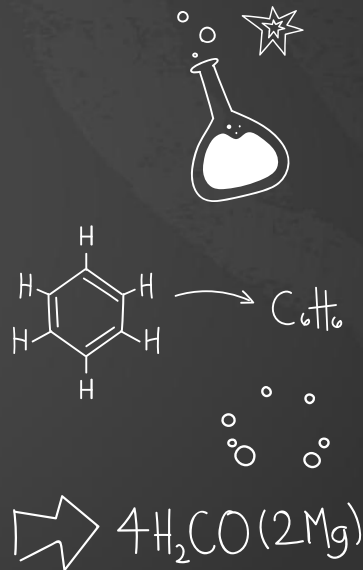
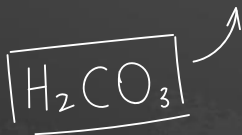
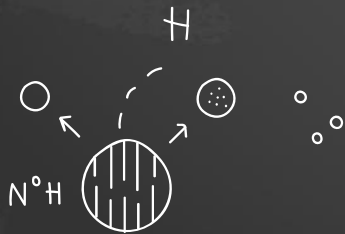
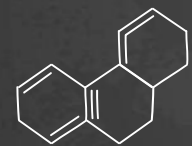
# Dúvidas?





# Gerenciamento de Memória

Delphi



# Gerenciamento de Memória

O Delphi não possui um “Garbage Collection” igual as outras linguagens



“Você é responsável pelo que você cria.”

# Gerenciamento de Memória

4 áreas da memória:



# Gerenciamento de Memória

## Estáticos

- Segmento de Dados: Variáveis Globais – Tamanho de até 2 GB.
- Segmento de Código: Constantes, Inicializações de Variáveis Globais, Rotinas e Métodos no Link Editor.

\*Aqui o gerenciamento fica por conta da IDE, a única coisa que devemos ter cuidado são com variáveis globais.

## Dinâmicos

- Segmento de Pilha (Stack): Chamadas de Procedures e Funções, Parâmetros passados e Variáveis locais – Memory Leak
- Segmento de Monte (Heap): Objetos, Instâncias de Objetos e ponteiros.

\*Aqui devemos tomar muito cuidado, pois uma má programação poderá causar um vazamento de memória ou inchaço e parar a operação do cliente.

# Gerenciamento de Memória

## Exemplo

```
procedure TForm1.Button1Click(Sender: TObject);
var
  xAnimal: TAnimal;
begin
  try
    case TEnumAnimal(cmbAnimal.ItemIndex) of
      stCachorro:
        xAnimal := TCachorro.Create;
      stGato:
        xAnimal := TGato.Create;
      stPassaro:
        xAnimal := TPassaro.Create;
    end;

    ShowMessage(xAnimal.RetornarSom);
    ShowMessage('Tenho ' + IntToStr(xAnimal.Patas) + ' patas.');
```

Criando o objeto

```
  finally
    FreeAndNil(xAnimal);
  end;
end;
```

Destruindo o objeto

# RTL – Run Time Library

Delphi





# RTL – Run Time Library

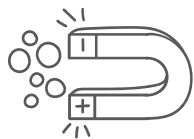
Biblioteca de Rotinas/Métodos nativa do Delphi

- Conversores de Valores
- Formataadores de Valores
- Data e Hora
- Manuseio de Files

\*A maioria presente na Unit SysUtils.pas

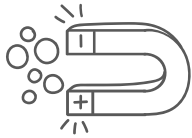
# RTL – Run Time Library

- MessageDlg
- StringReplace
- Trim



Vamos carregar  
o Delphi!!!





# Dúvidas?

