
MOONWOLF と学ぶ PASCAL 文法

超入門編

LAZARUS を使った

直観的なプログラム開発とは

MOONWOLF

はじめに

MoonWolf（むーんうるふ）です。

本書のテーマはプログラム言語 PASCAL（パスカル）です。
その PASCAL のグラフィカルな開発言語として

Delphi（デルファイ）と
Lazarus（ラザラス）などがあります。

本書では Lazarus を使いながら PASCAL の文法の要点を説明していきます。

ボーランド Delphi の経営判断の結果とは：

なぜ今、Lazarus なのか？
この疑問に正面から答えたいと思います。

私はかつて、ボーランド Delphi のユーザーだったのです。
それは Windows95、Windows98、Windows2000 の頃でした。

ボーランド Delphi / C++ は業界で不動の地位を獲得していたものの、
その会社は 2000 年代に個人向け、中小企業向け案件の支援を縮小し、
大企業向け案件を経営のターゲットにあてる経営にシフトしました。

その経営方針は、Delphi の個人ユーザー、
中小企業案件のユーザーにとって、非常に残念なものでした。
そのような顧客を軽視する風潮は企業を一気に傾け、
最終的には、2008 年、Embarcadero（エンバカデロ）という
企業に買収されることになりました。

Lazarus が多くの Delphi ユーザーの受け皿となる：

その時、個人、中小企業案件の Delphi ユーザーは大量に
Lazarus に流れていきました。
Lazarus とは Delphi のコピー & ペーストの真似事の開発環境ではなく
立派に独立した正当性のあるプログラム開発環境なのです。

Lazarus の正当性 :

Lazarus の GUI は、かつての Delphi のインターフェースに似ていますが、独自に開発されたものです。

Lazarus には Delphi の著作権の侵害にならない法的な根拠があります。

[1] Lazarus は Free Pascal という言語で独自に開発されています。

[2] 開発コードはすべて公開されています。

この 2 点が、Lazarus が独立したものとしての正当性を証明しています。

これに対して Delphi は Object Pascal にて開発され、
コードは企業秘密として非公開となっています。

つまり、Delphi と Lazarus はそれぞれが独立した存在と言えます。

そのため、無料で個人、中小企業案件をサポートする Lazarus と
有料で、大企業案件を支援する Embarcadero Delphi は
それぞれ、お互いを認識し合い、お互いを磨き合う関係となっています。

Delphi と Lazarus を使ってみた感想 :

私は、2020 年に何十年かぶりにプログラミングを再開したくなりました。
特に Pascal の開発環境を使ったプログラミングが手ごろかと思いました。
コロナ禍による自宅待機のため、少し時間ができたためです。

ボーランド Delphi がなくなっていたことはショックでした。
私は Embarcadero Delphi と、Lazarus の両方を Windows10 に
インストールして比較してみました。

Embarcadero Delphi は基本的に大企業案件をサポートするもの。
年間 5000 ドルを超える商用アプリには課金が必要なこと。
また、課金をしないと個人のサポートが得られないこと。
そして動作がすこし遅く感じられたこと。
このようなことを知りました。

一方、Lazarus を体験してみると、動作が軽い事。
完全無料で使えること。フォーラムで個人でもサポートが得られること。
さらに、作成したソフトを商用利用できること。
これらの点が分かり、私は Lazarus を選択しました。
昔使っていた、Delphi と感覚が似ていて私にはなじみやすかったです。

結論 :

大企業案件なら Embarcadero Delphi という選択肢もあるでしょう。

個人や中小規模の案件なら Lazarus を使う手もあるでしょう。
その中間の方はご予算に合わせて、どちらかを選択したら良いでしょう。

私は個人なので、Lazarus を選択することになったのです。
そして、Lazarus は完全無料の開発環境でも
大量に元 Delphi ユーザーが流れて来ていますので、
決して質の低い IDE ではない点、非常にありがとうございます。
※IDE(Integrated developing Environment)とは、プログラム統合開発環境のこと。

さらに、日本では Lazarus の認知度は低いものの、
プログラム言語 Pascal は、もともと教育用として開発されました。
そのため海外では依然教育目的のプログラム言語として広く使われています。

事例の紹介：(2024 年 9 月現在)

アメリカ - テキサス工科大学

Lazarus プロジェクトは、高校生徒および大学生を対象にプログラミングや STEM 教育を推進する取り組みです。STEM 教育は、Science (科学)、Technology (技術)、Engineering (工学)、Mathematics (数学) を統合して学ぶ教育です。

インドネシア - バンドン工科大学

コンピュータサイエンスコースでは、Lazarus を使って Pascal プログラミングの基礎を学び、特に GUI アプリケーションの開発に焦点を当てた授業が行われています。

フィリピン - フィリピン大学ディリマン校

クロスプラットフォームプログラム開発の一環として、Lazarus がコンピュータサイエンスプログラムで採用されています。

ドイツ - ケルン大学

2024 年 10 月に開催される予定の Free Pascal と Lazarus の国際会議では、教育機関や開発者が Lazarus の活用方法や新技術について議論されます。

加えて、Windows の Lazarus で書かれたプログラムコードは、
MacOS 版 Lazarus、Linux 版 Lazarus でも同じコードでコンパイルできます。
実際に私のゲームは、MacOS でも、Linux でも動くことが確認できました。

Lazarus が個人、中小企業案件、また教育にも向いていることが分かりますね。
それでは、Lazarus を使った Pascal 文法の世界の探求を始めてみましょうか！

MoonWolf

2024 年 10 月 10 日

※誤字・脱字がありましたら申し訳ありません。出版後も常に作品の改善に努めております。



MoonWolf(むーんうるふ)

※この画像は PixAI.Art により描かれています。
(Model : Moonbeam)

著者紹介 :

13歳の頃より BASIC でプログラムを始める。大学にて Turbo Pascal を教わる。Windows95～Windows2000 の頃ボーランド Delphi を使う。2020年より、 Lazarus(Pascal)を始める。2023年夏以降、Lazarus、Python、JavaScript、「なでしこ3」、SQL DBなどの入門書籍を Amazon より出版。Unity C#、Java、Linux にも積極的に取り組む。過去にプロのプログラマーとして、SQL DB に接続した結果を Web で表示するプログラム開発の経験を持つ。また、企業全体を統合管理する会計物流システムである ERP 導入プロジェクト参加の経験を合わせて持つ。小さじ一杯分だけでも日本人のプログラムカアップ支援に貢献したいと願っている。

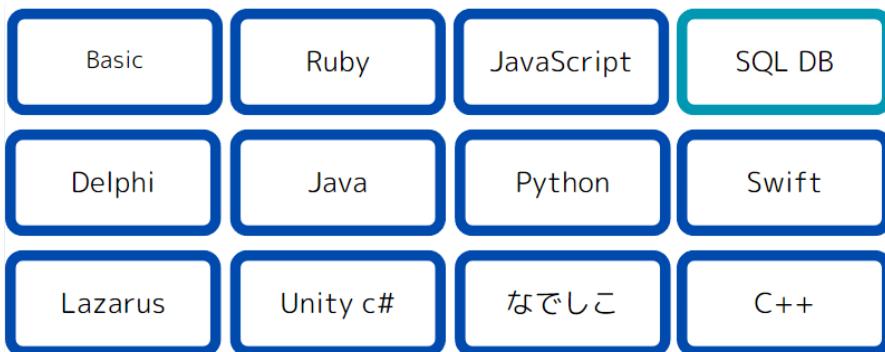
内容

はじめに	1
文章変更歴	7
第1章 なぜ今、Pascal と Lazarus に注目が集まるのか	8
第2章 他の言語と比較した Pascal 文法の特徴	13
第3章 Pascal を学ぶための Lazarus の準備	15
Lazarus で新規プロジェクトの立ち上げ方法	19
第4章 Lazarus を使った Pascal 文法(初級編)	22
Pascal の変数定義	23
Pascal 変数の禁止事項	25
Lazarus での文法学習の環境を整える	27
Lazarus での Hello World!	30
ここまで Lazarus のまとめ 1	32
コラム : Lazarus は OOP のフル知識がなくても使える	33
変数の定義、計算結果などの Memo1 への表示	37
コラム : コードに全角スペースが入った場合のエラー	40
Pascal の算術演算子・比較演算子	41
Pascal の論理演算子	44
コラム : Memo1 の文字サイズを調整する方法	46
ここまで Lazarus のまとめ 2	48
コラム : MoonWolf、Lazarus OOP 概略を語る 1	50
第5章 Lazarus を使った Pascal 文法(中級編)	56
Pascal の配列の特徴	57
Pascal の1次元配列	58
Pascal の2次元配列	59
Lazarus Record 型データの使い方	61
コラム : Lazarus のデータのスコープ	68
Pascal の条件判定文	69
コラム : Pascal の列挙型データ	71
コラム Pascal の定数	72
Pascal の条件判定 Case 文	74
Pascal の繰り返し処理	75
Pascal の For 構文 (Lv.1)	76
Pascal の For 構文 (Lv.2)	77
Pascal の For 構文 (Lv.10)	78
Pascal の While 構文	79
第6章 Lazarus を使った Pascal 文法(上級編)	80
Pascal の ProceduRe と Function	81
Lazarus での手作り Procedure	84
第4章 sqlite3 の入手と準備	90
あとがき (MoonWolf & ChatGPT4)	92
◆◆◆ご協力者の皆様◆◆◆	94
参考文献	95
MoonWolf の作品	98

文章變更歷

DATE	Document Version	Remarks
2024.10.5	Kindle 1.00	Kindle 版 初版

第1章 なぜ今、Pascal と Lazarus に注目が集まるのか



プログラム言語と言ってもここに表示できないくらい、たくさんあります。

この書籍は少し内容が変わっています。2024年10月現在、日本では

ほとんど認知されていない、Lazarusを取り上げて話を進めていくわけですから。

認知がされていないから、その機能が劣るのか？

いいえ、劣らないどころか、むしろ優れている点の方が多いです。

つまり多くの人が、お宝的プログラム開発環境を発見できていない状態にあります。

まず、プログラム言語：Pascalについて。Pascalは教育用言語として

開発されたプログラム言語です。しつけがとても厳しいです。

プログラムが、曖昧さを含まず、がっちりしているのです。

Lazarusで使うプログラム言語 Pascal のメリットを3つ上げておきます。

[1] Pascal のメリット：コンパイル時にエラーを発見しやすい

ひとつは、プログラム言語の記述方法が厳しいことにより、

早い段階でエラーの検出が可能となります。

プログラミングとは数々のエラーとの対決です。

エラーの対処にも2種類あります。

コンパイル時のエラーと実行時のエラーです。

コンパイル時のエラーは修正が容易で、実行時のエラーは一般的に難しいです。

プログラムの実行前にコンパイルという作業があり、

ソースコードをマシン語化します。Windowsの実行ファイルを作ります。

その段階で、Lazarusが厳密にコードのエラーチェックをしてくれます。

使えば使う程に、このメリットは大きさが理解できます。

例えば、変数を使用する場合、Pascalでは、その変数の型の定義から入ります。

一方、Pythonなどでは、変数の型の定義は無く、

プログラムを実行してみるまで、その変数の型は不明です。

このようにPascalとは全く逆の立場をとるプログラム言語もあるのです。

[2] Pascal のメリット：他のプログラム言語が理解しやすくなる

ふたつ目は、この厳密なコードの記述が求められるPascalを経験しておくと

それより、記述ルールのゆるいプログラム言語に対しては、

包み込むようにして理解が可能なのです。

厳しい環境から、厳しさが少しゆるい環境になじむのは難しくないですね？

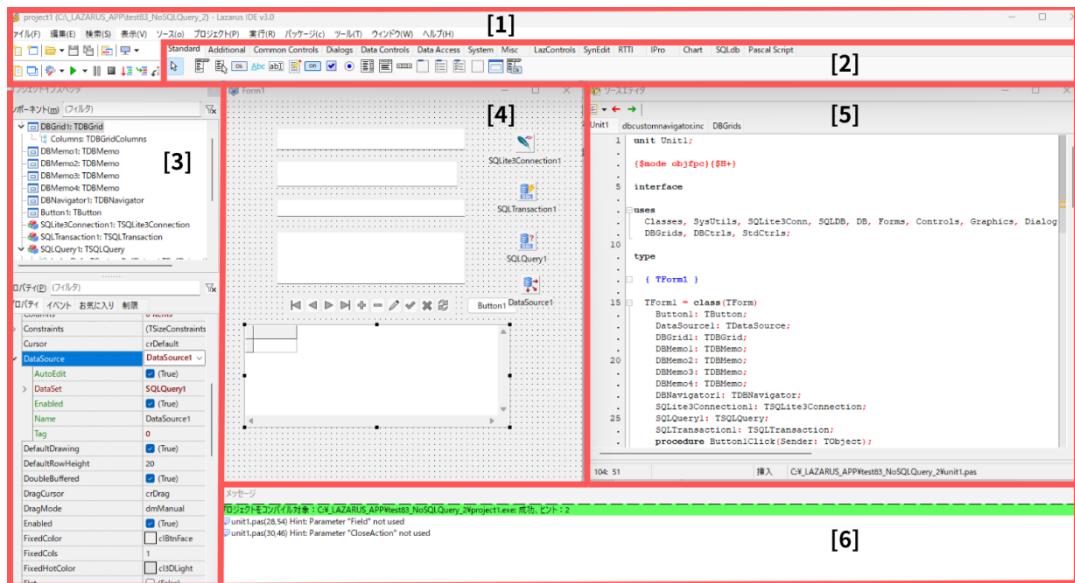
私は Pascal を基本として身に着けておいたので、短い時間で、
Python、JavaScript、なでしこ、これらのプログラム言語を
大きな苦労もなく理解でき、吸収してきました。
もちろん高いレベルのプログラムが開発できるためには経験が必要です。

[3] Pascal のメリット：プログラムのコードが読みやすい

3つ目は、Pascal の文法で記述されたプログラムのソースコードは
人間にとって可読性が良好です。もともとプログラミングの教育用言語でもあるので。
要するに、人間が見てすらすら意味がわかるコードになっています。
どれだけ長くて分かりやすい変数名を使ってプログラムを記述しても、
コンパイルしてマシン語になってしまったら、
長くても短くても関係がないように設計されています。

次にプログラム統合開発環境 Lazarus について。
Lazarus には、プログラム言語 Pascal のみを扱い、
IDE : Integrated Development Environment
日本語では IDE をプログラム統合開発環境と言いますが、
この IDE が他の言語にない充実さを持っているのです。

要するに、他のプログラム言語の開発環境より、
より直観的に、よりスムーズに Windows のソフトが作成できます。
いったい Lazarus IDE の何が優れているのでしょうか？



Lazarus のプログラム開発環境(IDE) :

[1] メインメニュー（お得度：★）

ここはどのプログラム開発環境で同じようなものです。

[2] コンポーネントメニュー（お得度：★★★）

コンポーネントが、視覚的にタブに整理されて整頓しています。他ではあまり見かけません。

Button (ボタン)、Memo (メモ帳)、Label (ラベル)、Timer (タイマー) などを
すぐに使えるものとして、視覚的に表現されて準備されています。

[3] オブジェクトインスペクター（お得度：★★★★★）

選択されたオブジェクトに対して大きく分けて二つの操作を可能にします。

それは、プロパティ(属性)とイベント(メソッド)の操作です。

選択したフォーム、ボタン、メモなどのコンポーネントの
すべてのプロパティ(属性)が一覧で表示されます。
ここで必要なプロパティーだけを変更することができます。

他の言語はでは、それができないので、プログラムで直接記述する以外に方法がありません。

Lazarus はどちらもできます。オブジェクトのプロパティーをコードで直接、記述して指定する。

あるいはオブジェクトのプロパティーをオブジェクトインスペクターで設定する。いずれも可能です。

その上、そのオブジェクトが持つイベント(メソッド)が一覧表示されています。

(イベント例 : onClick[クリックした時の動作]、onChange[変更があった時の動作]など)

使いたいものを選択すると、そのイベントコードのひな形が自動生成されます。
後は、そのイベントに何をさせたいか記述すれば良いという仕組みになっています。
他の言語でこれと同等な物を見たのは、ゲームや動画を作成する Unity C#くらいです。

[4] フォームデザイナー（お得度：★★★★★）

これも他の言語ではない場合が多いです。 Lazarus の場合、
オブジェクトインスペクターと連動して、フォームのデザインを作成できます。
またフォーム上でマウスを使って見た目を整え、設定した設計は、
自動的にオブジェクトインスペクターに反映されます。
プログラムの実行前に視覚的に、直観的なデザインを可能にしているところがポイントです。

このフォームデザイナー機能がない開発環境の場合、座標付きのコードでプログラムに
どこに何を表示するのか、すべてプログラムで記述する必要があります。
方眼紙などに座標に基づいていくらでも詳細を設計できますが、
それを、プログラムとしてすべて手打ちで作成することは Delphi/Lazarus ユーザからすると面倒です。

[5] コードエディター（お得度：★★★）

ここも他のプログラム環境とたいして変わりません。
そうは言っても、Lazarus は自動で作成できる部分はすべて生成してきますので、
ユーザーは足りない部分のコードだけを Pascal の文法で記述していくべきなのです。

[6] メッセージウィンドウ（お得度：★）

ここも他のプログラム環境とたいして変わりません。
Lazarus からの情報を受け取る場所です。コンパイルが成功したか？
エラーがあるなら、何がエラーなのか？これらを詳しく教えてくれます。

Lazarsu というプログラム環境は、プログラム言語 Pascal にて記述をしていきます。本書では、Pascal の文法に初心者の方、他のプログラム言語の経験者の方が段階的になじめるように、工夫をしながら Pascal の文法を説明していきます。

Pascal の文法の特徴 :

[1] 命令文の終わりには必ず「;」セミコロンを付けます。

[2] 変数名や命令で、半角の大文字と小文字を区別しません。

変数名 : MyCar, mycar, MYCAR、これらはすべて同じ意味となります。

他のプログラム言語の多くは、半角の大文字と小文字の区別をします。

[3] コメントの表記方法

// にて単数行のコメント、{} 中カッコにより複数行のコメントです。

// これは一行のコメントです

```
{ これは複数行の  
  コメントです }
```

[4] 変数を使う前に、明示的に型を定義します。

整数の I という変数を使う際には、

```
var
```

```
Int1 : integer; // 整数の型の変数 Int1 を定義
```

このように、変数を明示的に使う前に型の宣言を行います。

[5] 手続き(Procedure)と関数(Function)があり特定の処理を実行できます。

簡易化して説明します。

手続き Procedure とは、戻り値を持たない処理のかたまり。

関数 Function とは、戻り値を持つ処理のかたまり。

最初はこのように簡単に区別しておいた方が良いです。

他のプログラム言語では、関数しかないものもあります。

後ほど、詳しく違いや使い方を述べます。

[6] プログラムのブロックは begin で始まり end で終わります。

[7] Pascal での代入の記号は := です。

有名な Python と比較してみましょう。

Python は使用する変数の型を事前に定義しません。

極端な話、実行時までその変数の型は不明です。あるいは実行時に型が変わります。

Python では、命令文の最後に「;」などの命令文の終わりを

意味するコードがありません。そのため改行コードが命令文の終わりです。

また、コードのかたまりを示す begin と end がありません。

それを補うものとして、コードのかたまりをインデント(通常スペース 4 つ)にて

見た目的にも、意味的にも、まとまりを示したものになっています。

Python を主軸に使っている方がこの書籍をご覧になった場合、

以上の説明にて Pascal コードへの最初の違和感が緩和されると思います。

[1] Lazarus公式ページよりソフトのダウンロード

[2] Lazarusのインストール

[3] Lazarusの起動 + プロジェクトの準備

Lazarus の準備とは、上記の 3 点となります。

[1] 公式サイトからのダウンロード

あなたが Windows11 を使用しているものとして説明します。

Google で「Lazarus Official」と入力。



これが、Lazarus の公式ページへの入り口

 Lazarus
<https://www.lazarus-ide.org> · このページを訳す · [⋮](#)

[Lazarus Homepage](https://www.lazarus-ide.org)

Lazarus is a Delphi compatible cross-platform IDE for Rapid Application Development. It has variety of components ready for use and a graphical form designer.

The screenshot shows the Lazarus official website. At the top, there's a dark header with the Lazarus logo (a paw print) and the word "Lazarus". Below the header is a purple navigation bar with links: HOME, ABOUT, SCREENSHOTS, FAQ, FEATURES, DOWNLOADS, FORUM, and WIKI. The main content area has a dark blue background. It features several screenshots of the Lazarus IDE interface, showing various windows like code editors and form designers. To the right, there's a section titled "Lazarus" with the subtitle "The professional Free Pascal RAD IDE". It lists features: Cross platform, Drag & Drop Form Designer, Open source (GPL/LGPL), and Delphi converter. A large orange button says "Download Now". Below it, a link says "Version 3.4 for Windows 64 bit | Other ▾". On the right side, there's a sidebar with sections for "What is Lazarus?", "What can it do?", and "Where". Under "Where", there are download links for Windows 32 Bits, Windows 64 Bits (which is highlighted with a pink underline), Linux DEB 32 Bits, and Linux DEB 64 Bits.

2024年10月現在のLazarusの公式ホームページ。
自動判定のDownload Nowを押すか

Windows 64 Bits、を選択してダウンロード開始。

画面が変わりますが、5分くらい待ってあげて下さい。

[2] Lazarus のインストール

そのまま、インストーラーの画面の指示に従ってインストールしてください。

嬉しいことに Lazarus3.0以上にて、自動的にメニューが日本語化されています。

YouTubeにて Lazarus3.0のインストールの動画を作成しました。

インストール方法が気になる方は、こちらをご覧ください。

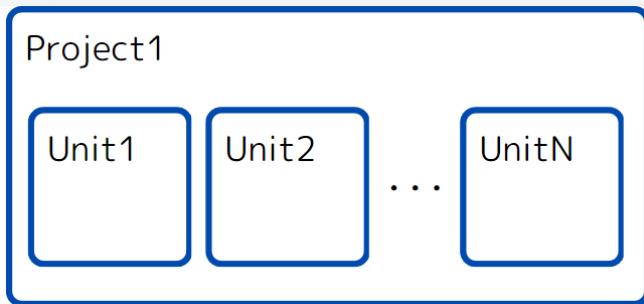
<https://www.youtube.com/watch?v=bGx18dA8yIo>

私のインストール動画の特徴は、別バージョンの Lazarusを残したい方向けです。

私は書籍を書いている都合上、

マニアのように古いバージョンの Lazarusも残しているのです。

[3] Lazarus の起動と、プロジェクトの準備



Lazarus の全体像は、このように Project ファイルがあり、

その中に、いくらでも多く、Unit ファイルを作成できる仕組みになっています。

初心者の場合、Project1 と Unit1 だけで大丈夫です。

このデフォルト名の Project1、Unit1 を使うものとして話を進めていきます。

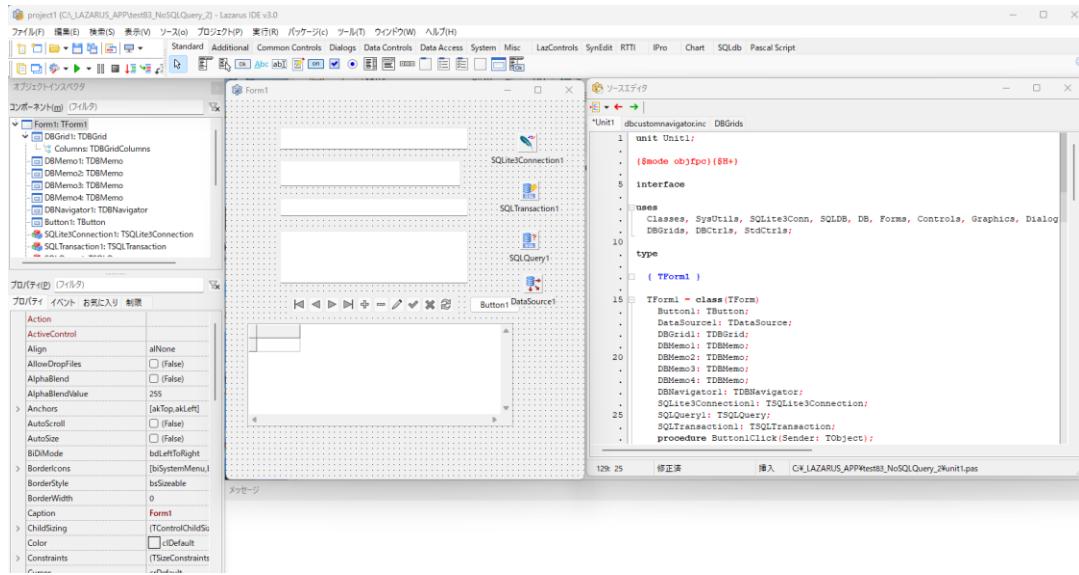
新しいプログラムを作る度に、新しい Project ファイルとその Unit ファイルを
別のフォルダーに保存していきます。

また、作成中のプログラムのバックアップも簡単で、

そのフォルダーごと、コピーを取っておけば大丈夫です。

私は一つのプログラムの完成までに、念のためフォルダーごとコピーして
複数のバックアップを作成しながらプログラムを開発しています。

インストールが完了した Lazarus を起動してみましょう。



この画面のように以前、開発していた画面が表示されます。

そこからすぐに続きの作業ができるように。

しかし、ここが落とし穴の 1 つです。このまま新しいプログラムを作成するために
この状態をいじってしまうと、過去に作成されたプログラムを壊してしまいます。

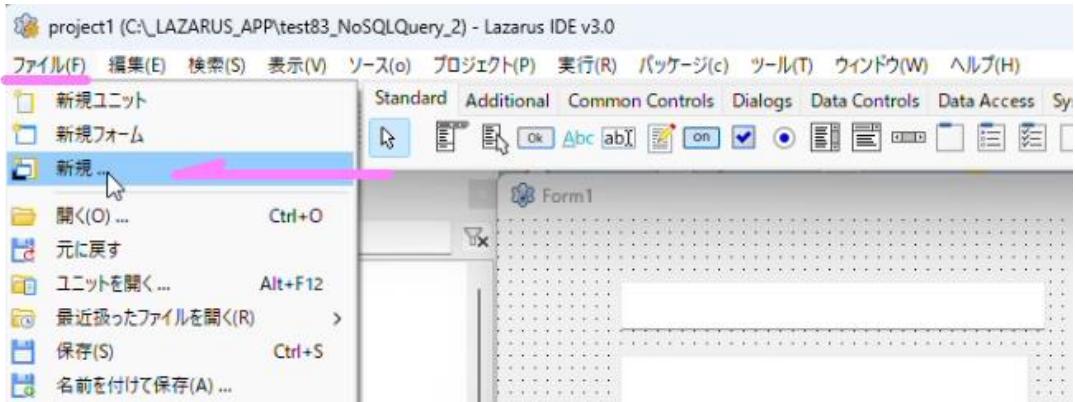
新しいプログラムを作成する際には必ず新しいプロジェクトを立ち上げること

新しいプロジェクトを立ち上げるとは、どういうことか？

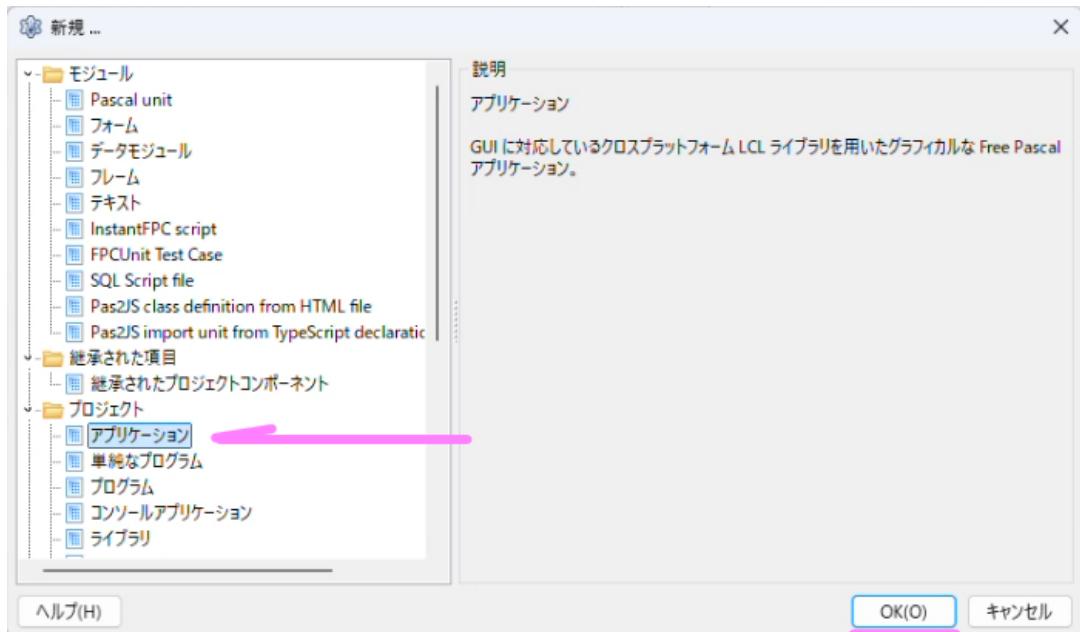
- [1] 新しいプロジェクト用のフォルダーを作成(C:\Laz_App\test001 など)
- [2] Lazarus で新しいプロジェクトの起動
- [3] Project1 ファイルを用意したフォルダーに保存
- [4] Unit1 ファイルを用意したフォルダーに保存

これらの操作が必要になります。実際の画面を通してお見せします。

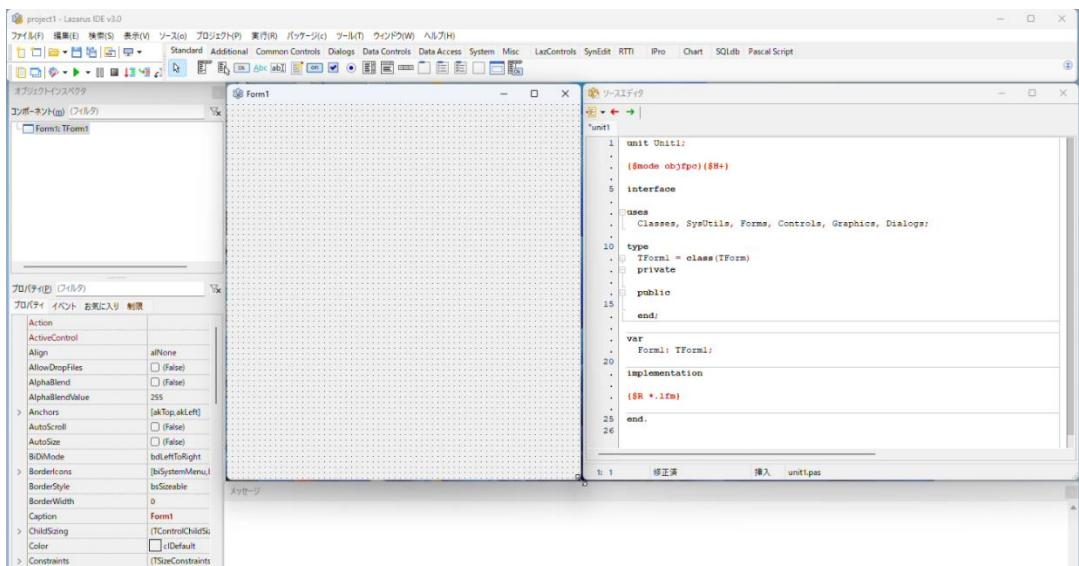
LAZARUS で新規プロジェクトの立ち上げ方法



メインメニュー → ファイル → 新規 、を選択。



プロジェクト → アプリケーション 、を選択。

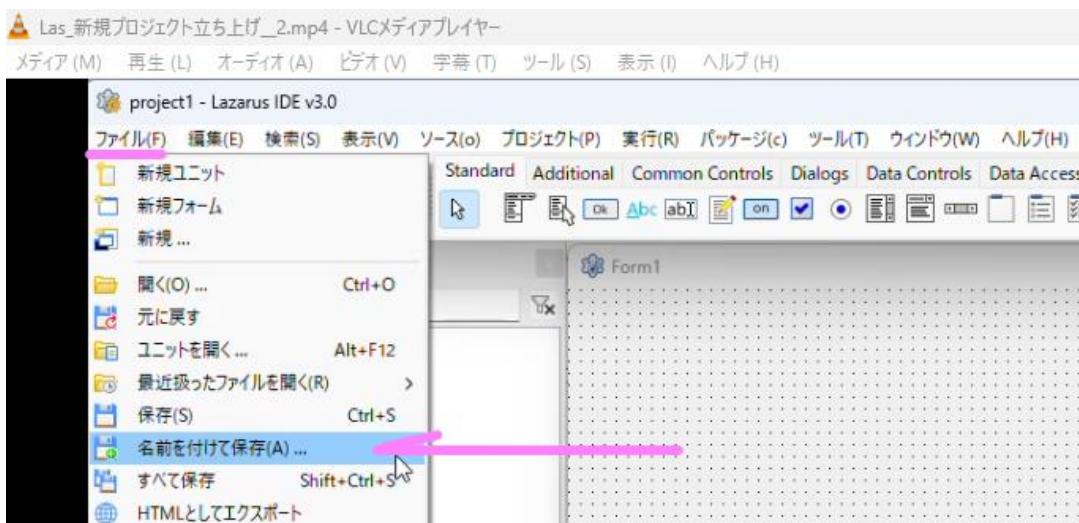


ここまで操作で、だいたいこの画面を表示します。

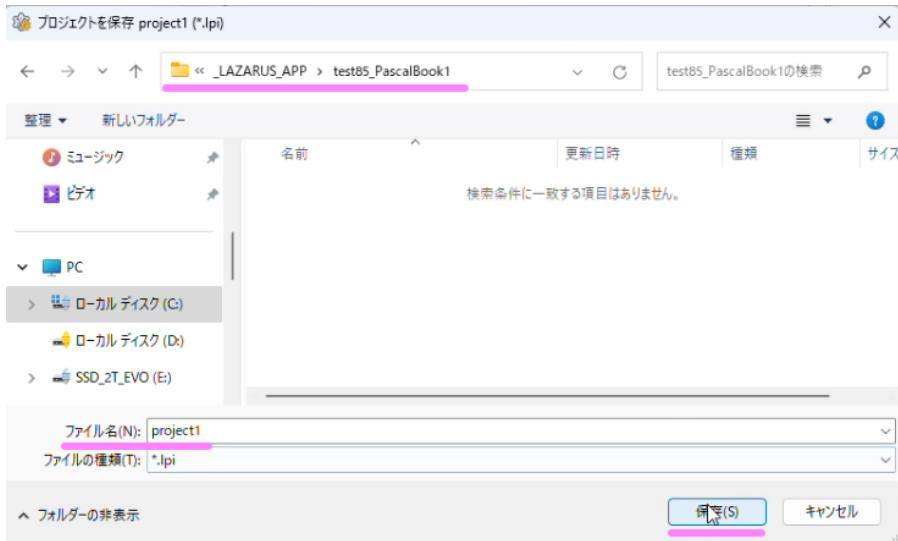
メッセージが表示されていない場合、

メインメニュー → 表示 → メッセージ、を選択してみてください。

まだ、これは表示しているだけです。関連ファイルを保存します。

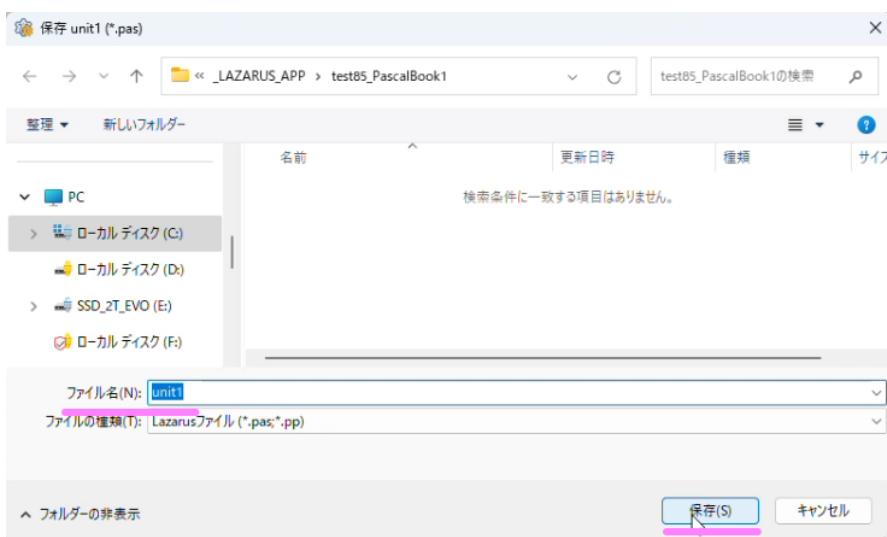


メインメニュー → ファイル → 名前を付けて保存、を選択



プロジェクトファイルの保存：

まず保存したいフォルダーに移動。ファイル名はそのまま project1 で保存。



Unit ファイルの保存：

ファイル名はそのまま Unit1 で保存。

これで新しいプロジェクト（プログラム）を始める準備が整いました。

本書のテーマは、Pascal の言語を抽象的に説明するものではなく、実際に Lazarus を使って、その文法の意味が分かるように記していきます。

また、Lazarus は他の言語に比べて、

プログラム開発環境の IDE は優れていますが、

プログラム言語 Pascal は、他の言語と比較して、

そう大きく変わるものではありません。

Pascal にも、変数の定義があり、配列があり、条件処理があり、

繰返し処理があり、手続きである Procedure があり、関数である Function があります。

ここ的基本部分は、プログラム全体を通して基礎的な概念であり、

それをどのように記述するかの書式に少し差がある程度です。

しかし、書式のルールを学ばないと、Pascal でプログラムは組めないので、

本書の例を参考に、しっかりと基礎力を身に着けてスキルアップしてみてください。

GitHub でのコードの公開について：

本書で紹介するプログラムコードは GitHub に掲載しておきます。

場所はこちらです。

https://github.com/moonwolf001/Lazarus_Pascal/tree/book600

それでは、Pascal の変数を Lazarus を使って学習していきます。

変数とは、数学で使うような、 x 、 y みたいなものです。

しかし、じっくり向き合ってみると変数には様々な制限があり、

この部分をまず、しっかり押さえておかないと、

将来意味不明なエラーで悩まされ、プログラムとはつまらないものだと思い、

プログラムが一度も動くことなく、この世界を去ってしまう人も多いです。

数学の世界では同じ X に対して、

$X=1$

$X=3.553$

などと代入できますが、Pascal の世界では、

少し人間から見て慣れが必要なのですが、

コンピュータのデータ処理の都合上、整数型の変数と、小数を使う変数

これらを区別して扱う必要があります。

では、他に大枠どのような区別があるのでしょうか？

整数型 Integer 1,5, 10など	実数型 Real 3.552 など	文字列型 String 'おはよう'など	論理型 Boolean True / False
--	------------------------------------	---------------------------------------	---

- ◆ 整数型 : Integer (インテジャー)
- ◆ 実数型 : Real (リアル)
- ◆ 文字列型 : String (ストリング)
- ◆ ブール型 (論理型) : Boolean(ブール), True(真) or False(偽)

変数とはデータを格納する箱と考えてもらって良いです。

箱に入るデータのタイプは、コンピューターから見て都合のよい分類になっています。

このようなデータの変数、データの入った箱を
 上手に使い分けながらプログラムを組んでいきます。
 そして Lazarus では変数をプログラムで使う前に、
 明示的に変数の型を定義します。
 詳しくは後程 Lazarus を使いながら、お見せします。

Pascal の変数では、いくつか使ってはいけないルールがあるので、
ここは、しっかり押さえておきましょう。
この理解が不正確であると、後にエラーで悩むことになります。

[1] 変数名に全角文字は使えません。

NG 例 : 県名 1、名前 1

[2] 変数名を数字から始めることはできません。

NG 例 : 015Car、0001Chair

[3] Pascal は半角の大文字、小文字を区別しません。

どれも同じ変数です : Car001、car001、CAR001

[4] アンダースコア「_」以外の記号は使えません。

NG 例 : Car/2、Prefecture*3、Good-Place

[5] 変数名にスペースは入りません。

NG 例 : Good Place, Student Profiles

[6] あまりにも長い変数名はダメ。50 文字以下を奨励します。

[7] Pascal、Lazarus の予約語を変数名として使えません。

※何が予約語なのかは、次のページを参照のこと

Lazarus の予約語一覧：

```
absolute, abstract, alias, and, array, as, asm, assembler,  
begin, break, case, cdecl, class, const, constructor, continue,  
default, deprecated, destructor, div, do, downto,  
else, end, except, export, exports, external,  
false, far, file, finalization, finally, for, forward, function,  
generic, goto, if, implementation, in, index, inherited, initialization,  
inline, interface, is, label, library, local, message, mod,  
name, near, nil, nodefault, not, object, of, on, operator, or, out,  
overload, override, packed, pascal, platform, private, procedure,  
program, property, protected, public, published, raise, read, record,  
register, reintroduce, repeat, resourcestring,  
safecall, sealed, set, shl, shr, stdcall, stored, string,  
then, threadvar, to, true, try, type,  
unit, until, uses, var, virtual, while, with, write, xor
```

ずいぶんたくさんありますね。

初心者の間は、このページを見ながら判断すると良いでしょう。

予約語そのものは使えませんが、次のようなものは使えます。

begin01, begin_or_001,

予約語の後ろに文字などを付ければ使えますので大きな心配は要りません。

また、文字列のデータとして、'begin' という文字データも問題ありません。

LASARUS での文法学習の環境を整える

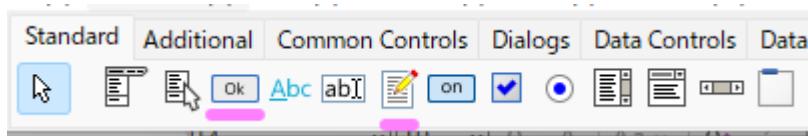
Pascal 文法が学習できる環境を整えていきます。

現在は、新しい Project と Unit1 があなたの指定フォルダーに保存されている状態です。

コンポーネント一覧をご覧ください。

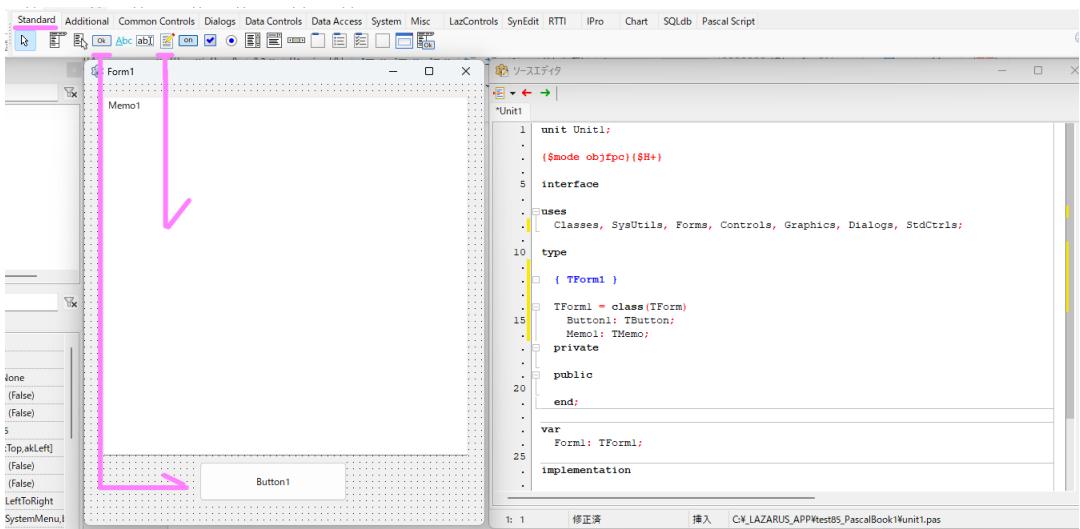
Standard というタブの中に、Button と Memo というコンポーネントがあります。

マウスをその上の持ってくると、名前を示します。



ボタンのコンポーネントを選択した状態で Form1 をクリックしてください。

同様に Memo をクリックしたい状態で Form1 をクリックしてください。

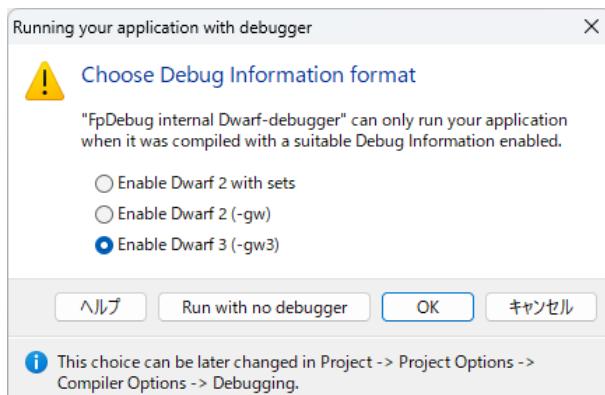


最初は大きさは小さいのですが、マウスを使って

この画面のように、Form1、Button1、Memo1 の大きさを調整してください。

では、この状態で動作チェックを行います。

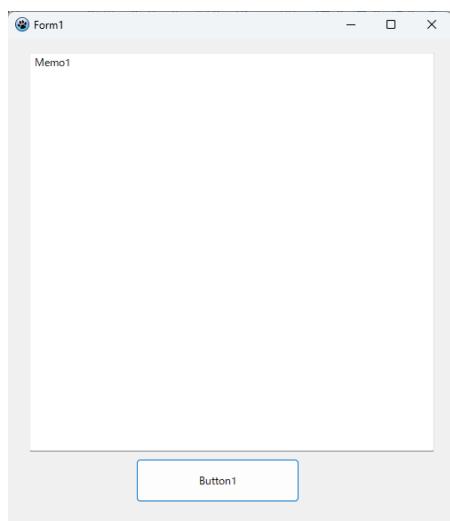
メインメニュー → 実行 → 実行、 を実施してみてください。



新規プロジェクトの実行の際に、最初に一回だけこのメッセージがでてきます。

なにも考えず、 Enable Dwaf 3 このまま、 OK を押してください。

Lazarus 開発者でデバッカの意見が分かれているそうです。

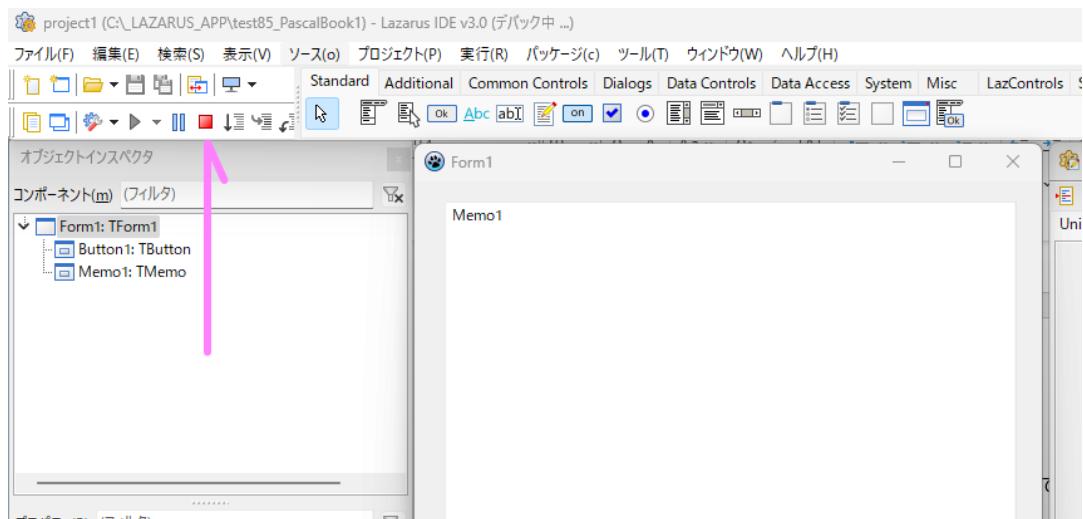


すると、ほら！このようにツルンとしたフォームが出てきました。

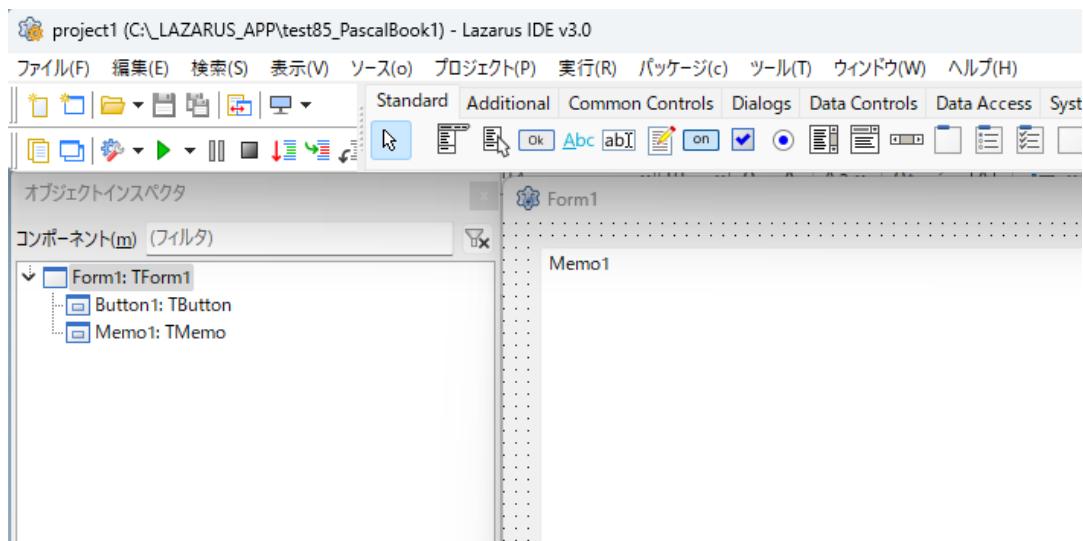
これが実行結果です。何も起きませんが、ボタンを押したり、

文字を書いたりしてみてください。何もプログラムを作成することなく

Lazarus がここまでプログラムを生成してくれたのです。



メインメニューにある、この赤四角ボタンを押すと、実行が停止されます。



プログラムの実行が停止された画面。

画面上の緑色の三角ボタンを押すと、もう一度実行できます。

これで、Lazarus プログラムの実行と停止を学習できました。

LAZARUS での HELLO WORLD!

せっかくですので、「Hello World！」やってみましょうか！

プログラムを実行中でしたら、停止してください。

設計中の Form1 上に設置した、Button1 をダブルクリックしてみてください。

```
. implementation
.
.
{$R *.lfm}
30
.  □{ TForm1 }
.
.  □procedure TForm1.Button1Click(Sender: TObject);
.  □begin
35
.  □    |
.  □end;
.
.  □end.
39
```

すると、ほら！ Lazarus がコードを自動生成してくれます。

ピンク色で囲んだ部分が、自動生成の部分です。

Button1 を押した時に何をするのかは、ここに記述したら良いのです。

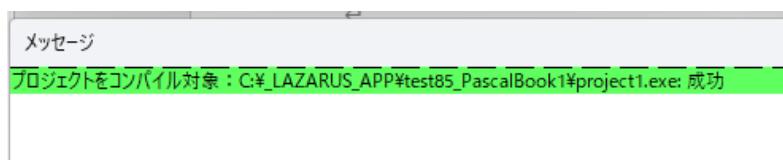
```
.  □procedure TForm1.Button1Click(Sender: TObject);
.  □begin
35
.  □    Memo1.Append('Hello World!');
.
.  □end;
```

begin と end の間に、一行だけプログラムコードを入れてみてください。

Memo1.Append('Hello World');

すべて半角文字です。最初に半角スペースを 2 回入れています。
プログラムに誤りがないか、コンパイルしてみましょう。

メインメニュー → 実行 → コンパイル、を実行してみましょう！



コードにエラーが無ければ、このようにファイル名が緑色の行で表示されます。

コンパイルしてエラーが無かったので実行してみましょう。

Lazarus では、以下三種類のいずれかの方法でプログラムの実行ができます。

[1] メインメニュー → 実行 → 実行

[2] メニューの下にある、緑色の三角ボタンを押す

[3] ファンクション・キーの F9 を押す

プログラムの実行後、Button1 を押してください。



すると、ほら！

Hello World!と表示しましたね！ボタンは何回押しても構いません。

小さな一步ですが、これはあなたにとって記念すべき瞬間なのです。

初めての Lazarus で入力したプログラムが動いたことになり、

プログラムの世界、Lazarus の世界に一步、足を踏み入れた特別な瞬間なのです。

Lazarus まとめ 1

- ◆ Google などより、 Lazarus Official と検索して Lazarus の公式ページを開く
- ◆ Windows11 の方は、 Windows 64 Bits をダウンロード
- ◆ ファイルサイズは大きくないが、 5 分くらいダウンロードを気長に待つ
- ◆ そのままインストーラの指示に従ってインストール
- ◆ Lazarus Version 3.0 以上は自動で日本語メニュー設定となる
- ◆ 起動してみて、 メッセージ・ウィンドウを表示していない場合、 表示させる
- ◆ プロジェクトを始める際、 必ずそれ専用のフォルダーを自分で事前に作成

Project1 と Unit1 を、 その用意したフォルダーに保存してから作業開始

< Hello World までの道のり >

- ◆ Form1 に、 Button1 と Memo1 をのせ、 マウスで形を整える
- ◆ Button1 をダブルクリックして、 Button1 を押した時の
コードのひな形を Lazarus に作成させる
- ◆ そこに、 Memo1.Append('Hello World!'); と、 コードを記入
- ◆ メニュー下の緑三角ボタンを押して実行
- ◆ Form1 上の Button1 を押して、 Hello World が出力されたことを確認

ここでコラムをはさみます。

Lazarus と OOP（オブジェクト指向プログラミング）について、

2024 年 10 月現在の、MoonWolf の見解を示しておきます。

ChatGPT4 に確認したところ、私の述べるような見解の書籍は

今のところ世界にありません。

他の書籍では見られない大胆な意見を述べてみたいと思います。

Lazarus は OOP をフル機能装備しています。

そのため、PASCAL+その OOP が完全に理解できないと、

Lazarus なんて、使えないのではないか？

このような意見や、心配する気持ちがある人もいらっしゃると思います。

MoonWolf の結論は、NO です。

これは、1 年かけて ChatGPT4 と激しく議論してきた結果です。

そして ChatGPT4 を論破して得られた結果です。

もう少し詳しく述べると、開発チームが作った OOP のクラス、

つまりコンポーネント(Button や Memo など)はフル活用で使えます。

OOP のすべての知識を知らなくても簡単に使えます。

この時点で、自動的に OOP の恩恵は十分に受けています。

しかし、自作の OOP のクラス（設計図）を作るかどうかは、
そのユーザーの成長度に合わせて使えば良いのではないか？

OOP を 100% 理解して活用しなくても、

十分に Lazarus の恩恵は受けられるのではないか？

このような寛容な考え方もあると思います。

OOP を使った Lazarus やコンポーネントの設計者と、
それを使うユーザーとの間には相当な技術的レベル差があります。

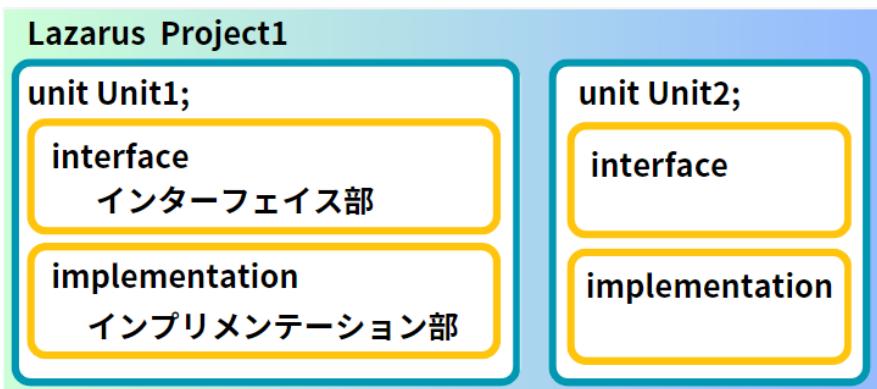
OOP は Lazarus 設計者関係の方に使っていただいて、
優秀なコンポーネントを作っていただければ良いという考え方もあるのです。

つまり OOP の深い知識無しに Pascal の知識のみで Lazarus は使えるのです。

このような結論に至りました。

この考え方、 Lazarus や Delphi のユーザーを増やす考え方です。
誰かがこれを、明言してくれないと、これから Lazarus を自分の使う
プログラム開発として使おうかどうか判断できません。

自分が逆の立場で、なでしこのみ、Python のみを使っている立場で、
Lazarus を勧められたことを想定した場合、
Lazarus は OOP のフル装備らしい、OOP を完全に理解しないといけないのかな？
ちょっと難しそうなので、やめておこうかな。と、考えてしまうかもしれません。
違います！OOP の完全な知識や経験なしに使えます。
その理由をこれから、論理的にお話します。



Lazarus にはもともと、**interface 部**、**implementation 部**
 という、Unit 事に大きなプログラムのブロックがあり、
Interface 部の記述は、他の Unit から参照されることを前提に
変数、定数、手続き(Procedure)、関数(Function)を記述していきます。

つまり、Project1 があり、Unit1、Unit2、Unit3 といくらでも
 Unit を追加していく機能があるのです。

この機能は、見かたによっては、
 OOP のクラスの作成とその利用という部分と機能が一部かぶる、
 重複する部分があると、私は見ているのです。

つまり、私は、Lazarus のユーザーには、次のような
 学習のステップ、あるいは、成長段階があつて良いと考えるのです。

MoonWolf の提唱する、Lazarus の習得ステップ：

[1] Project1 と Unit1 だけを使い、プログラムを作成する時代

まずは、Unit1 だけをとことん使い込むプログラムを作る。

[2] Project1 と Unit1、Unit2 だけを使い、プログラムを作成する時代

Unit が 1 つ加わるだけで、プログラムの記述に大きな飛躍が必要です。

[3] Project1 と Unit1、Unit2、… UnitN を自在に使える時代

使う Unit 数に制限はありません。この状態でほぼ何でも作成できると思います。

[4] Lazarus の OOP をフル活用する時代

これは、[3]までの機能で不足を感じる人だけが到達するステップ。

それは、OOP の自作クラスが自由に生成できる。

それを、他のクラスを自在に継承できる。

自作 OOP クラス、またはその継承クラスを、自在にインスタンス化できる。

当然[4]の状態が一番、Lazarus のプログラマーとしては自由自在な状態ですが、

そこに至るまでの道のりが、初心者的人には少し道のりが長いです。

つまり多くの人が転落してしまう可能性があるのです。

まずは[1]の状態、次に[2]の状態、それでも足りない人は[3]の状態を

目指したらよいです。これで多くのプログラムは実現出来てしまうでしょう。

それでも、力に余裕のある人、OOP のフル機能を活用したい人、

そうした方々が、その次のレベルの[4]の段階に入れば良いと考えます。

変数の定義、計算結果などの MEMO1 への表示

私は以下のようなステップで、Lazarus を使いながら Pascal 文法を説明していきたいと思います。

1	Button1 を押したときの動作としてプログラムの作成
2	実行
3	Button1 を押す
4	プログラムの実行結果を Memo1 で見る

先ほどの Hello World ! を作成したプログラムがそのまま使えます。

また、プログラムの記述のルールが厳しい Pascal においては、

変数の定義から、変数を使うまでに次のプロセスに従います。

1	var という記号の下に使う変数をまとめて事前に定義
2	変数の初期化（定義された変数への初めての代入）
3	変数を使う

Hello World! のプログラムを使いたくない人向けに準備の指示：

- [1] 新規プロジェクト用のフォルダーを作成 (C:\LasAPP\test02 など)
- [2] Lazarus の起動
- [3] メインメニュー → ファイル → 新規
- [4] 新規 Window でプロジェクト → アプリケーションを選択
- [5] ファイル → 名前をつけて保存 (準備したフォルダーを選択)
- [6] Project1, Unit1 を名前を変えないで保存
- [7] Form1 上に、Button1 と Memo1 をのせる
- [8] Form1、Button1、Memo1 をマウスで大きさを整える
- [9] Button1 をダブルクリックする -> Button1 Click の Procedure が作られる

```
.  procedure TForm1.Button1Click(Sender: TObject);
.  var
35   s : String;
. begin
37   s := '本日は晴天なり';
.   Memo1.Append(s);
. end;
40
```

はい。それでは、Procedure TForm1.Button1Click(Sender: TObject);

ここの内側を上の絵のように、プログラムを変更した後に実行します。

ポイントを説明します。

◆ Lazarus では、変数は使う前に var という記号の下に定義する。

var とは英語の variable (変数) の略ですね。

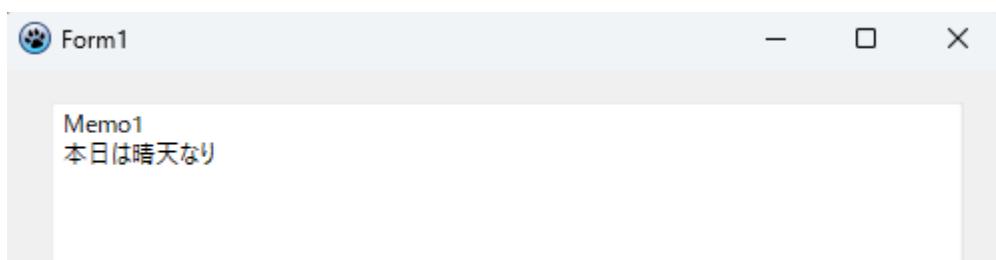
◆ String という型は、文字列型のデータです。

◆ begin と end の間には、何行でもプログラムを記述できます。

◆ s:='文字列'、これは s という変数に文字列を代入しています。

Lazarus での代入は := という記号を使います。

あとは、その変数 s を Memo1 に追記で表示。



実行をして、Button1 を押した結果がこちら。

※全角文字が使えるのは、文字列データの 'この間だけ' とコメントだけ

である点に注意してください。他はすべて半角文字です。

```
.  procedure TForm1.Button1Click(Sender: TObject);
.  var
35    Str1, Str2, Str3 : String;
.  begin
.    Str1 := 'MoonWolfの世界へようこそ！';
.    Str2 := 'お魚がたくさんいますよ！';
.    Str3 := Str1 + Str2;
40    Memo1.Append( Str3 );
.  end;
```

次は、このようにコードを書き替えてみましょう。

◆var の変数宣言部にて、カンマ区切りで同時に複数定義できます。

◆文字列のデータは「」シングルルクオーテーションで囲みます。

◆文字列データへの、文字列の追加は「+」で、できます。

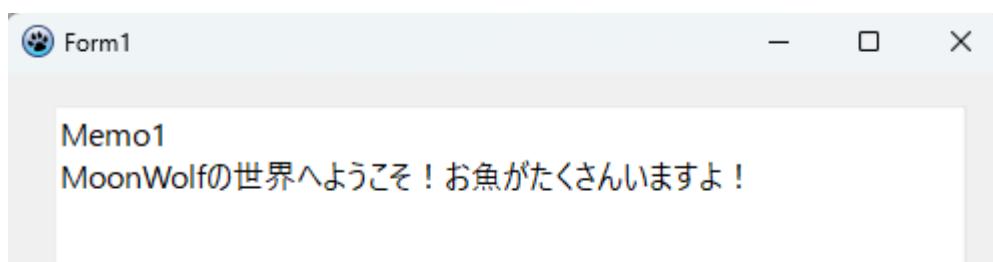
◆Memo1.Append()、このカッコの部分には、文字列のデータが入ります。

意味は、Memo1 の文章に、Append(追加)で新しい行にデータを追加です。

このプログラムでは、Str1 と Str2 に文字列を代入して、

それぞれの文字列データを、Str3 に足しながら代入しています。

その内容を、Memo1.Append() にて Memo1 に表示しています。



コラム：コードに全角スペースが入った場合のエラー

はい。とっても恐ろしい話をしてみましょうか！

心の準備は良いですか？

なんと、Lazarus のエディターでは、全角スペースも半角スペースも

同じように空白で表示します。しかし全角スペースはエラーです。

とくに、コメントや、文字列を全角で入れた後、半角モードに戻さないでそのまま記述した場合など注意です。

```
.  procedure TForm1.Button1Click(Sender: TObject);
.  var
35   Str1,Str2,Str3 : String;
.  begin
37   |~~~~~
.    Str1 := 'MoonWolfの世界へようこそ!';
.    Str2 := 'お魚がたくさんいますよ!';
40    Str3 := Str1 + Str2;
.    Mem1.Append( Str3 );
.  end;
```

意図的に、全角スペースを入れて、コンパイルしてみました。

日本人が最も多く接するエラーなのではないでしょうか？

メッセージ
プロジェクトをコンパイル対象: C:\LAZARUS_APP\test85_PascalBook1\project1.exe: コード1を終了、エラー: 1
! unit1.pas(37,3) Error: illegal character "・" (\$E3)

Error: illegal character "・" (\$E3)

これが、全角スペースがコードに交じった場合のエラーです。覚えておきましょう！

※この奇妙なエラーの出る原因是、ユニコード 3 バイトの全角文字から、Lazarus は 1 バイトずつ文字列を読み取るため、そのコードが不正と言っています。

PASCAL の算術演算子・比較演算子

数値を表す変数を扱う前に、Pascal での計算のルールを覚えておきましょう。

Pascal の算術演算子：

演算子	説明	例	結果
+	加算	$5 + 3$	8
-	減算	$5 - 3$	2
*	乗算	$5 * 3$	15
/	除算(実数除算)	$5 / 2$	2.5
div	除算(整数除算)	$5 \text{ div } 2$	2
mod	剰余(余り)	$5 \text{ mod } 2$	1
^	累乗	2^3	8

※掛ける、割る、累乗(るいじょう)にて、特殊な記号を使うので注意してください。

Pascal の比較演算子

演算子	説明
$A = B$	A と B は等しい
$A \neq B$	A と B は等しくない
$A > B$	A は B より大きい
$A < B$	A は B より小さい
$A \geq B$	A は B 以上
$A \leq B$	B は A 以上

※他の言語では、 \neq (等しくない) を $A \neq B$ と表現する言語も多いです。

Pascal では $A \neq B$ を使うので注意してください。

```
.  procedure TForm1.Button1Click(Sender: TObject);
.  var
35   i1,i2,i3 : Integer;
.  begin
.    i1 := 10;
.    i2 := 2;
.    i3 := i1 + i2;
40    Memo1.Append( IntToStr(i3) );
.  end;
```

次は、このようにコードを書き替えて、整数の型を扱ってみましょうか。

◆Integer(インテジャー)とは、整数だけを扱うデータの型です。

◆整数 A と整数 B の足し算、引き算、かけ算、累乗は整数となります。

◆整数 A を整数 B で割った数は、整数にならない場合があるので注意です。

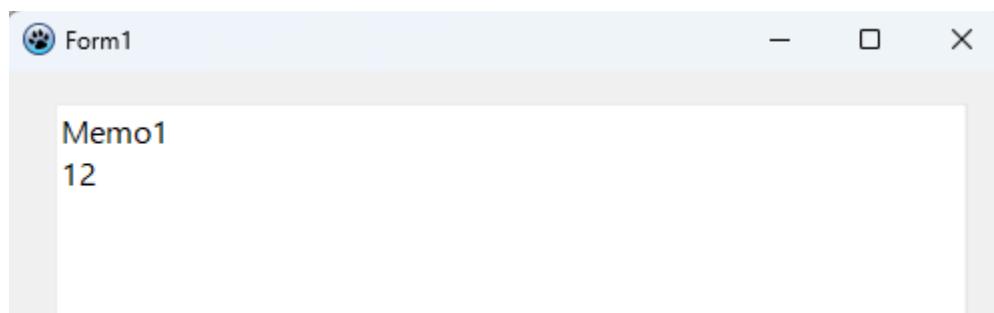
10/2=5 で整数ですが、10/3=3.333333.. と整数ではなくなります。

◆IntToStr()という関数は、整数型のデータを文字列型データに変換する関数です。

このプログラムでは、i1 に 10、i2 に 2 を代入して、

i3 に i2 と i3 の和を代入しています。

それを整数型より IntToStr()にて、文字列型に変換してから表示しています。



```
. . . procedure TForm1.Button1Click(Sender: TObject);
. . . var
35 . . .     Real1, Real2, Real3 : Real;
. . . begin
. . .     Real1 := 1.22;
. . .     Real2 := 2.331;
. . .     Real3 := Real1 + Real2;
40 . . .     Memo1.Append( FloatToStr(Real3) );
. . . end;
```

次は、実数型の計算例をやってみましょう。

◆Real(リアル)とは、小数を含む数の変数の型です。

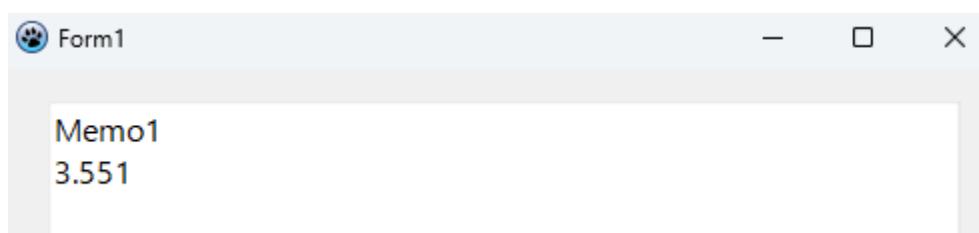
◆Real 型変数 A と Real 型変数 B の計算結果は、通常 Real 型として扱います。

◆FloatToStr() 関数にて、Real 型データを文字列データに変換しています。

このプログラムでは、Real1 に 1.22 、 Real2 に 2.331 を代入し、

Real3 にその和を代入しています。

FloatToStr()にて Real 型より文字列に変換してから表示しています。



Pascal の論理演算子

演算子	説明	例
and	論理積、かつ	A and B : AとBが真の時、真
or	論理和 または	A or B: AかBが真の時、真
not	論理否定 否定	not A : Aの逆
xor	排他的論理和	A xor B : ABの組み合わせが真偽、偽真の時だけ真

True(真)を 1、False(偽)を 0 として考えて詳細を見てみましょう。

そのように考えると、以下のようにシンプルに表現できるのです。

1 and 1 => 1

1 and 0 => 0

0 and 1 => 0

0 and 0 => 0

1 or 1 => 1

1 or 0 => 1

0 or 1 => 1

0 or 0 => 0

not 1 => 0

not 0 => 1

1 xor 1 => 0

1 xor 0 => 1

0 xor 1 => 1

0 xor 0 => 0

ちなみに、xor とは、Exclusive OR (エクスクルーシブ オア)の略となります。

```
.  procedure TForm1.Button1Click(Sender: TObject);
.  var
35   Boo1, Boo2, Boo3, Boo4 : Boolean;
.  begin
.    Boo1 := True;
.    Boo2 := False;
.    Boo3 := Boo1 and Boo2;
40   Boo4 := Boo1 or Boo2;
.    Memo1.Append( BoolToStr(Boo3,True) );
42   Memo1.Append( BoolToStr(Boo4,True) );
.  end;
```

次は、 Boolean 型（ブール型）を書いてみましょう。

◆Boolean とは、 True、 False のいずれかをもつブール型のデータです。

◆BoolToStr()関数にて、 Boolean 型データを文字列に変換しています。

第2引数の True とは、結果を True/False で表すための命令です。

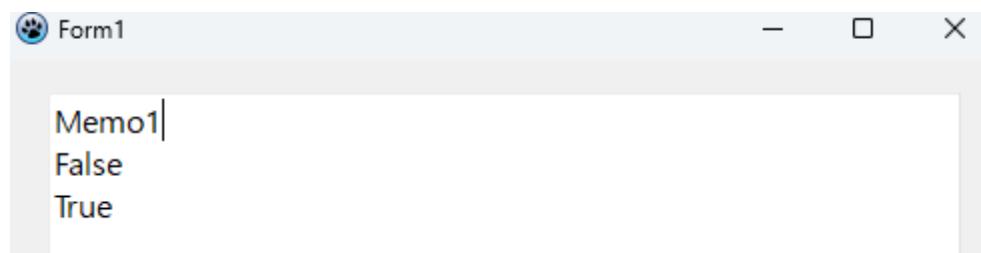
このプログラムでは Boo1 から Boo4 までのブール型の変数を準備しています。

Boo1 には True、 Boo2 には False を代入。 Boo1、 Boo2 を通して、

Boo3 には、 True And False を代入しています。結果は False です。

Boo4 には、 True Or False を代入しています。結果は True です。

それぞれの結果を、 BoolToStr()にて、文字列にて表示しています。



コラム：MEMO1 の文字サイズを調整する方法

実は、この書籍を書きながら、Memo1 のフォントサイズが小さかったので

少し文字を大きくするように調整していました。

これが直観的にできるところが、Lazarus のすごいところです。

その方法は、直観的で極めて簡単です。

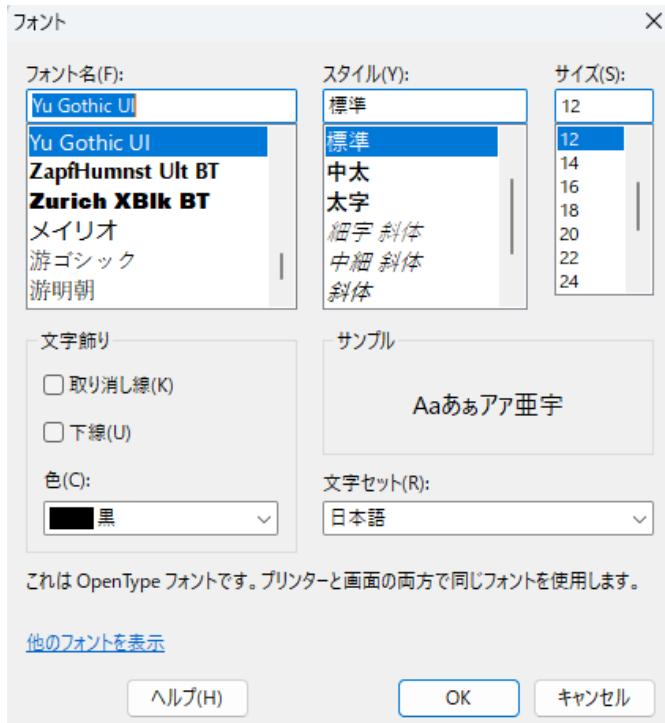
[1] フォームデザイナーで Memo1 をクリック

[2] Memo1 の内容を示すオブジェクトインスペクターに切り替わる

[3] オブジェクトインスペクターのプロパティー → font を選択

一番右に表示されている、「…」ボタンを押す





するとこのような、フォント設定ウィンドウが開きます。

初期設定のフォントサイズが 8 で小さく感じたので、12 に変更しました。

ここでは、フォントの選択、色、太字、サイズなどを変更できます。

そして変更した様子を、フォームエディターで確認できます。

つまり、プログラムの実行前に視覚的に確認しながら開発できるのです。

他の開発言語環境では、このオブジェクトインスペクターが無い場合が多いので、

このような直観的に操作できる機能はとてもありがたいですね。

Lazarus まとめ 2

- ◆ Lazarus はオブジェクト指向プログラミングの完全な理解がなくても使える
- ◆ 変数は変数宣言部にあらかじめ、まとめて宣言。var 以下に宣言。
- ◆ Pascal は begin と end の間にプログラムを記述。
- ◆ String により、文字列型データを定義する
- ◆ 文字列と文字列のデータは、 + により結合できる。
- ◆ Memo1.Append() このカッコの中には文字列データを入れる
- ◆ Pascal の代入は、 := という記号を使う
- ◆ 日本語全角文字が使えるのは、コメント内と文字列データのみ
- ◆ プログラムに全角文字が入った場合のエラーは

Error: illegal character “・”(\$E3)
- ◆ Integer により、整数型データを定義する
- ◆ IntToStr() 関数により、整数型を文字列型に変換できる
- ◆ Real により、小数を含む実数型データを定義する
- ◆ FloatToStr() 関数により、実数型を文字列型に変換できる
- ◆ Boolean により、 True(真)、 False(偽) のどちらかを持つ布尔変数を定義
- ◆ BoolToStr() 関数により、 布尔型を String 型に変換できる

```
.  procedure TForm1.Button1Click(Sender: TObject);
.  var
35   Name1    : String;
.   Age      : Integer;
.   Message1: String;
.   Emotion  : Boolean;
. begin
40   Name1    := 'MoonWolf';
.   Age      := 20;
.   Message1 := '日本の皆さん Pascalの世界良いですよ！';
.   Emotion  := True;
.
45   Memo1.Append('私の名前は ' + Name1 + ' 気持ちは ' + IntToStr(Age) + ' 歳');
.   Memo1.Append(Message1);
.   Memo1.Append('That is ' + BoolToStr(Emotion, True));
. end;
```

それでは、このように書き替えてみましょう。

今までの説明の混合バージョンです。

◆Memo1.Append()の内容が String である限りいくらでも入ります。

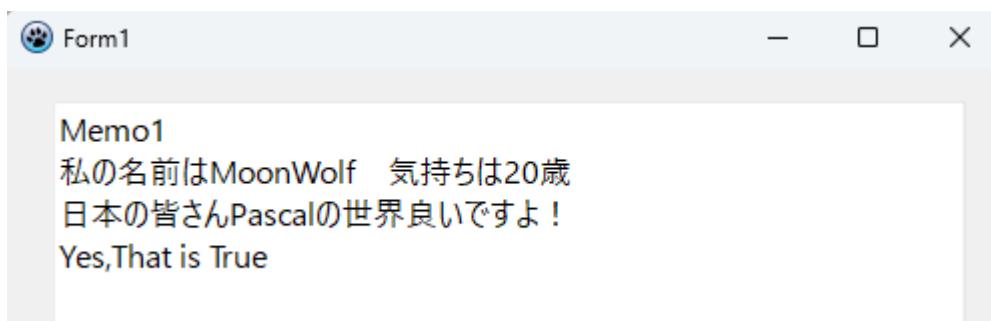
◆String 型と String 型のデータは + にて結合できます。

◆別の行に表示したいときは、もう一度 Memo1.Append()にて記載したら OK。

Append とは、追記という意味です・

このプログラムでは、String、Integer、Boolean を使い変数を定義しています。

それらに値を代入して、Memo1.Append()にて文字列として表示しています。



ここで、変数の定義の仕方が一段落したところで、少し気持ちに余裕ができましたね。

別コラムにて、Lazarus には OOP の知識がなくても使えますよ。

と、述べましたが、Lazarus のコンポーネントを使うということは、

無意識のうちに OOP の恩恵を受けていることになります。

Lazarus の本格的な OOP(オブジェクト指向プログラミング)を理解することは

大変なことですが、Lazarus の OOP の概略を理解することは誰にでもできます。

もちろん説明する人が、上手に説明できることが前提です。

OOP の概略が理解できると、Lazarus で何をしているのか理解でき紆が深まります。

私はすでに以下の書籍にて、OOP の概略、Python における OOP を書籍化しています。

『MoonWolf と学ぶ Python シリーズ オブジェクト指向プログラミング 超入門編』

<https://www.amazon.co.jp/dp/B0CYH1537B>

本書で述べる内容は、この内容を簡単に示し、Lazarus 向けにカスタマイズしてみます。

私の OOP 概略の説明は、読者に理解していただける自信があります。

何しろ私が理解するまでに、非常に時間がかかり苦労したのです。

その私が、理解できるまでとことん内容を煮詰めたものだけをお伝えします。



オブジェクト指向プログラミング(以下 OOP)には、いくつかの専門用語が出てきます。

この専門用語の意味や役割を、例え話でつかんでしまいましょう！

今、日本の自動車会社が新しい自動車製品を開発し、

日本と海外に発売するとしましょう。その商品を **JapanCarX009** としましょう。

自動車会社は当然、JapanCarX009 の設計図を作ります。

その設計図を、OOP の世界では、その自動車の **クラス**(Class)と呼びます。

クラスとは設計図のことなのです。

その設計図があれば、工場で製品である JapanCarX009 を生産できます。

その工場を、**コンストラクター**と呼びます。

コンストラクターとは設計図から製品を作るところです。

また、**クラス**からコンストラクター経由で生成された製品を**インスタンス**と呼びます。

インスタンスの別名が**オブジェクト**です。

新しい言葉が多いですね！ **クラス**、**コンストラクター**、**インスタンス**、**オブジェクト**。



さらにもう少し、シナリオを加えていきます。

日本で作成した、JapanCarX009 という日本の法律に適した車の設計図ですが、

アメリカとドイツに輸出するとしましょう。

国により道路交通法が異なります。

現地に合わせてカスタマイズして販売する必要があります。

国名	座席	ワインカー	速度表示
日本	右 / 左側通行	右	km/h
アメリカ	左 / 右側通行	左	mile/h
ドイツ	左 / 右側通行	左	km/h

カスタマイズが必要なすべてが、上記の部分だったと仮定します。

アメリカでは、座席を左側、ワインカーを左側、速度表示を mile/h (距離はマイル表示)

ドイツでは、座席を左側、ワインカーを左側、速度表示は km/h

アメリカの JapanCarX009(アメリカ版)の作成には、日本版の JapanCarX009 の
クラスを継承してから、上記の部分だけをアメリカ用に修正してから製造・販売します。
ドイツも同様に、日本車版のクラスを継承してからドイツ専用に修正してから製造です。

これが派生クラス。クラスの継承という概念です。

クラス 自動車 (設計図)

Private: 表に見せない技術的詳細

- エンジンの機能
- ミッションの機能
- ボディーのフレーム
- ECU(コンピュータ)

Public : ユーザーに使って欲しい機能

- ハンドル
- アクセル
- ブレーキ
- ギアシフトレバー

Protected : 開発関係者だけに公開機能

- 座席の左右の入れ替え
- メートル表示の変更
- ワイパーとウィンカーの入れ替え

日本の自動車会社の立場に立って考えてみましょう。

設計図(クラス)を少し詳しく見てみます。

この図では、設計図の内容が大きく分けて 3 種類のパートに分かれていますね？

Private (プライベート : 私的) な部分 :

ここには、自動車の利用者には通常は見せない技術的詳細が設計されています。

エンジン、ミッション、ボディーのフレームなどの設計詳細です。

Public (パブリック : 公的) な部分 :

自動車の利用者に使って欲しいものの設計図詳細です。

ハンドル、アクセル、ブレーキ、ギアシフトレバーなどの設計詳細です。

Protected (プロテクティッド : 守られている) な部分 :

関連会社の設計部隊だけが、設計図を変更できる部分です。

座席の位置、ウィンカーの位置、速度表示などの設計詳細です。

このように、クラス(設計図)を Private、Public、Protected といった言葉で、

誰にどの情報を見せるか決定する概念を、**OOP のカプセル化**といいます。

また、**Private、Public、Protected をアクセス修飾子(しゅうしょくし)**と呼びます。

日本の自動車会社の設計図（クラス）の作成者にはすべて見えています。

海外用に設計図を変更する部隊は、クラスの内容(技術)は引き継ぐものの

Private 定義の内容は秘匿されていて見えません。

Public と Protected の内容しか見せません。

Protected の内容をその国の法律に合わせて変更します。

車を購入した自動車の利用者は、コンストラクターによって生成された

自動車インスタンス（オブジェクト）を使うことになります。

車の利用者には、設計内容は車にあるものの Public 定義の内容しか見えません。

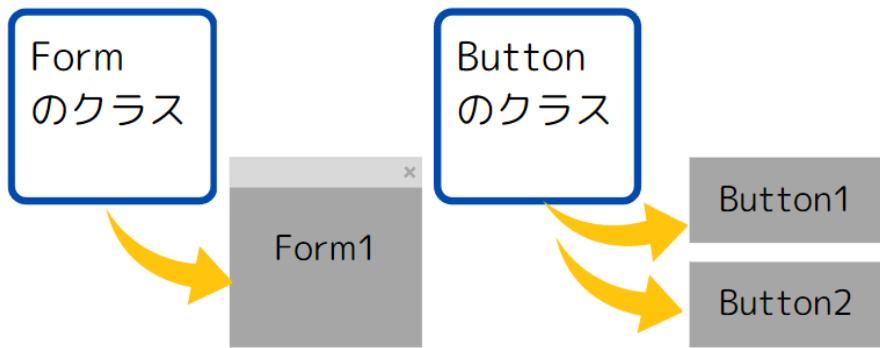
ハンドル、アクセル、ブレーキ、シフトのノブ、ワインカーなどです。

運転をする側には、むしろそれだけの方が多いみたいです。

これが OOP の世界に入るための、最初に押さえておきたいポイントです。

ここまで、OOP のまとめ：

- ◆ クラス(Class)とは設計図である。
- ◆ クラスの内容は、Private、Public、Protected などのアクセス修飾子により
誰にどこまでの情報や機能を見せるか制御される。
- ◆ クラスから、サブクラスを継承できる。
- ◆ クラスからコンストラクターを経由して、インスタンス(オブジェクト)が生成される
- ◆ インスタンスはクラス全体の設計図から生成されるが、
インスタンスの利用者は、Public の部分しか見えない。他は隠蔽されている。



今の段階では、この図のようなシンプルな理解をお勧めします。

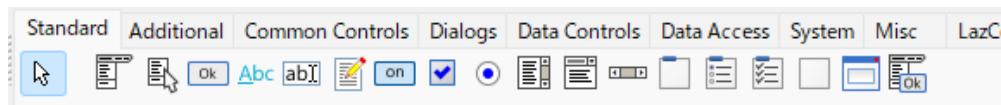
Lazarus は、Form1 や、Button1、Memo1 のクラスを持っている。

それを Lazarus の利用者は、無意識のうちに直観的に使うことができる。

Form クラスからは、Form1 というオブジェクトが生成され、

Button クラスからは、Button1、Button2 といくらでもボタンを生成できます。

それぞれのボタンは異なる動作が当然可能のです。



Lazarus のメインメニューの下にある、コンポーネント一覧は、

準備されて、いつでも使用が可能なクラスの集まりを、

さあ、どうぞ、どれでも使って下さいとでも言われているかのように、

視覚的にアイコンで示されていることになります。

さらなる詳細は、本書の別コラムにてお伝えします。

さて、中級編に入ります。もう少し内容の深い Pascal 文法に触れていきます。

この章では、

- ◆配列
- ◆レコード型データ
- ◆条件判定文
- ◆繰り返し文

このあたりに焦点を当てて、説明してみたいと思います。

PASCAL の配列の特徴

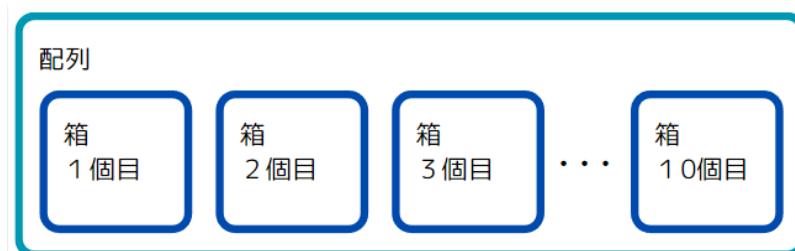
今までの、Integer 型、Real 型、String 型のデータ変数は、

1 つの変数に、一つのデータしか代入できませんでした。

変数とは、データを入れる箱と表現してみたりもしました。

その箱が何個かつながっているケースが、配列となります。

しかし同じ型のデータの箱がつながっているイメージとなります。



このように Integer 型のデータを 10 個入れる配列があったとしましょう。

1 個目の箱に、10 を代入。

2 個目の箱に、20 を代入。

3 個目の箱に、30 を代入。

…

10 個目の箱に、100 を代入。

このような値を、箱たちに代入したとしましょう。

Pascal の配列は、他の言語に比較して人間が理解しやすいように設計されています。

Pascal 以外のほとんどの言語が、配列の箱をゼロ個目から 9 個目と數えます。

しかし、Pascal では、人間の見た目通りに 1 個目の箱から 10 個目の箱と數えます。

```
. . . procedure TForm1.Button1Click(Sender: TObject);
. . . var
35   Matrix : array[1..10] of Integer;
36 begin
.     Matrix[1] := 10;
.     Matrix[2] := 20;
.     Matrix[10] := 100;
40     Memo1.Append(IntToStr(Matrix[1]));
.     Memo1.Append(IntToStr(Matrix[2]));
.     Memo1.Append(IntToStr(Matrix[10]));
. end;
```

◆配列の定義 Matrix : array[1..10] of Integer ;

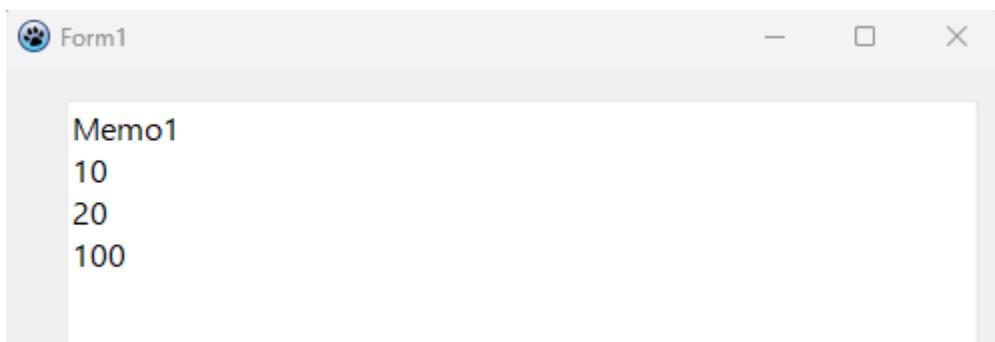
◆上の一行为、Matrix[1], Matrix[2], .. ,Matrix[10]の 10 個の箱ができます。

箱のデータの型は、Integer 型です。

◆配列への代入は、Matrix[1] := 10;

このように、配列を使うときは、**変数名[番号]**と表現することにより

Matrix[1]から Matrix[10]を個別の変数として扱うことができます。



PASCAL の 2 次元配列

```
. procedure TForm1.Button1Click(Sender: TObject);
.   var
35    Matrix : array[1..10,1..10] of Integer;
. begin
.   Matrix[1,1] := 10;
.   Matrix[1,2] := 200;
.   Matrix[10,10] := 10000;
40   Memo1.Append(IntToStr(Matrix[1,1]));
.   Memo1.Append(IntToStr(Matrix[1,2]));
.   Memo1.Append(IntToStr(Matrix[10,10]));
. end;
```

一次元配列が簡単に理解できた人は、次は二次元配列にチャレンジです。

◆配列の定義 `Matrix : [1..10, 1..10] of Integer;`

◆上の一行にて、 10×10 の 100 個の Integer 型の箱ができています。

`Matrix[1,1], Matrix[1,2], …, Matrix[1,10]`
`Matrix[2,1], Matrix[2,2], …, Matrix[2,10]`
…
`Matrix[10,1], Matrix[10,2], …, Matrix[10,10]`

このような感じです。それを個別の整数型変数として扱えます。

◆2 次元配列への代入例は、`Matrix[1,2] := 200;` のようになります。



1次元配列とは、横並びの直線に箱を並べるようなイメージでした。

2次元配列とは、横並びの直線の箱が上下に展開し、平面のように箱が並べられました。

3次元配列とは、縦、横、高さの、立体のような配列構造になります。

これは、配列を箱と考えるときのイメージ的なものですが。

さらに、Integer型以外にも、Real型、String型の配列も作成可能です。

配列は、1行の配列定義の命令にて、一度に100個でも1000個でも

一気にデータの箱を定義できるところが便利です。

配列の無い世界では、Data1、Data2、Data3、…、Data100と

ひとつずつ変数を定義する必要があり、配列を知ってしまうと、

それらは、分かりやすいものの、バカバカしい世界に見えることもあります。

しかし、Pascalの配列は、同じ型のデータの箱を一気に作成した後、

その箱を利用する技術でしたが、

データの型が異なるものを、同じ配列には定義できません。

プログラム言語 Pascal は、この問題を解決できないのでしょうか？

できるのです！

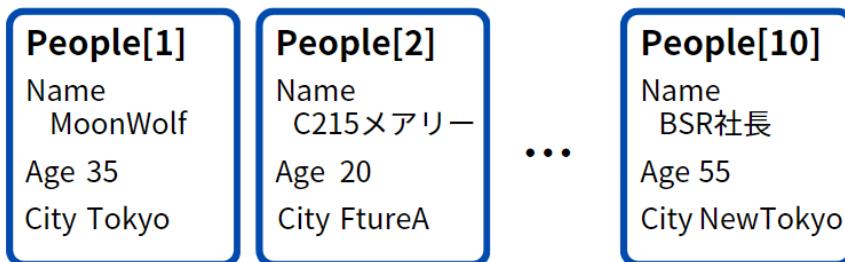
レコード型データという型を使うと、

1つの箱に、複数種類のデータを格納できるのです。

プログラムを作成する上で、このレコード型はどのプログラムでも

威力を発揮することでしょう。

Pascal Record型



レコード型データでやりたいことは、この図のようなものとなります。

People[1], People[1], … ,People[10] と言った配列の中に、

```
Name : String;
Age   : Integer;
City  : String;
```

このように異なるタイプのデータ型をセットで持たせたいとします。

```
type
  // TPerson 型のレコードを定義
  TPerson = record
    Name: String;
    Age: Integer;
    City: String;
    Weight: Real; // kg
  end;

var
  Person: array[1..10] of TPerson; // TPerson 型の配列を定義
```

Unit1 のどこに記述するかは、後程説明するとしまして、

このように、type から始まる、独自の Record 型を定義できます。

そして、その型を使いいいくらでも配列が作成できます。

Lazarus Project1

unit Unit1;

interface

インターフェイス部

implementation

インプリメンテーション部

unit Unit2;

interface

implementation

レコード型をどこに定義するか？

これを考えてみたいと思います。以前、Lazarus の Project1 があり、

その中に、Unit1 があり、必要ならば Unit2、Unit3 と作成していくことが可能です。

Unit1 のソースコードをよく見てみると、

何やら **Interface** とか **Implementation** という言葉が出てきていますね？

Unit1 は、このふたつのキーワードにより、大きく以下二つの分割できます。

Unit1 Interface 部：

ここは将来、Unit2 やそれ以降の Unit から参照される場所です。

Interface とは、他の Unit との Interface という意味です。

ここで定義されたものは、グローバル変数、グローバル関数などと呼ばれます。

Unit1 Implementation 部：

ここは、Unit1 だけで使うユニットスコープの変数、型、手続き、関数を書く場所。

もう一つの意味は、Unit1 の Interface で名前だけ示された手続き、関数の

具体的な内容をここで実装します。

ここで Lazarus の Unit1 ファイルをいくつか見て考察する時間を取ります。

```
unit Unit1;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, Forms, Controls, Graphics, Dialogs;

type
  TForm1 = class(TForm)
    private

    public

    end;

var
  Form1: TForm1;

implementation

{$R *.lfm}

end.
```

これは、新規プロジェクトを立ち上げた瞬間の Lazarus Unit1 のコードです。

すべて Lazarus の自動生成によるものです。なにやら既に作っていますね！

Interface 部には、Uses という部分で使うモジュールの取り入れ

TForm1 型の設定と、Form1 の定義しかありません。

Implementation 部には、何も内容がありません。

何もコードを記述していませんが、これを実行すると、Form1 が表示されます。

※参考資料：コンパイルオプションについて

{\$mode objfpc}{\$H+} : object Pascal 互換モード、長い文字列に対応

{\$R *.lfm} : オブジェクトインスペクターの設定は LazarusFormFile を参照

```

unit Unit1;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, Forms, Controls, Graphics, Dialogs, StdCtrls;

type
  { TForm1 }
  TForm1 = class(TForm)
    Button1: TButton;
    Memo1: TMemo;
    procedure Button1Click(Sender: TObject);
  private
  public
  end;

var
  Form1: TForm1;

implementation

{$R *.lfm}
{ TForm1 }

procedure TForm1.Button1Click(Sender: TObject);
begin
  Memo1.Append('Hello World!'); // ユーザーはここのみ記載
end;
end.

```

次は、Button1 を押すと Memo1 に Hello World の例です。

Interface 部にて、TForm1 のクラスが自動設計されています。

Implementation 部にて、Button1 のダブルクリックで生成された

Procedure が追加されています。そこに Hello World 表示と記載しました。

Lazarus が自動的に多くのコードを生成してくれるのでとっても楽ですね！

```

unit Unit1;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, Forms, Controls, Graphics, Dialogs, StdCtrls;

type
  // TPerson 型のレコードを定義
  TPerson = record
    Name: String;
    Age: Integer;
    City: String;
    Weight: Real; // kg
  end;

var
  People: array[1..10] of TPerson; // TPerson 型の配列を定義

type
  { TForm1 }

TForm1 = class(TForm)
  Button1: TButton;
  Memo1: TMemo;
  procedure Button1Click(Sender: TObject);
private

public

end;

var
  Form1: TForm1;

implementation

{$R *.lfm}

{ TForm1 }

procedure TForm1.Button1Click(Sender: TObject);
begin

```

```

// People[1]にデータを代入
People[1].Name := 'MoonWolf';
People[1].Age := 35;
People[1].City := 'Tokyo';
People[1].Weight := 70.5;

// People[2]にデータを代入
People[2].Name := 'C215 Mary';
People[2].Age := 28;
People[2].City := 'Osaka';
People[2].Weight := 55.3;

// Memo1 に表示
Memo1.Lines.Append('People[1]:');
Memo1.Lines.Append('Name: ' + People[1].Name);
Memo1.Lines.Append('Age: ' + IntToStr(People[1].Age));
Memo1.Lines.Append('City: ' + People[1].City);
Memo1.Lines.Append('Weight: ' + FloatToStr(People[1].Weight) + ' kg');

Memo1.Lines.Append('');

Memo1.Lines.Append('People[2]:');
Memo1.Lines.Append('Name: ' + People[2].Name);
Memo1.Lines.Append('Age: ' + IntToStr(People[2].Age));
Memo1.Lines.Append('City: ' + People[2].City);
Memo1.Lines.Append('Weight: ' + FloatToStr(People[2].Weight) + ' kg');
end;

end.

```

太字にして黄色マーカーを引いた部分が、私がコードを追加した部分です。

この流れに乗って、Record 型の説明に入ります。

Interface 部の Users 定義の直下に、レコードが型の宣言をしました。

さらにその下に、TPerson 型の配列を 10 個定義しました。

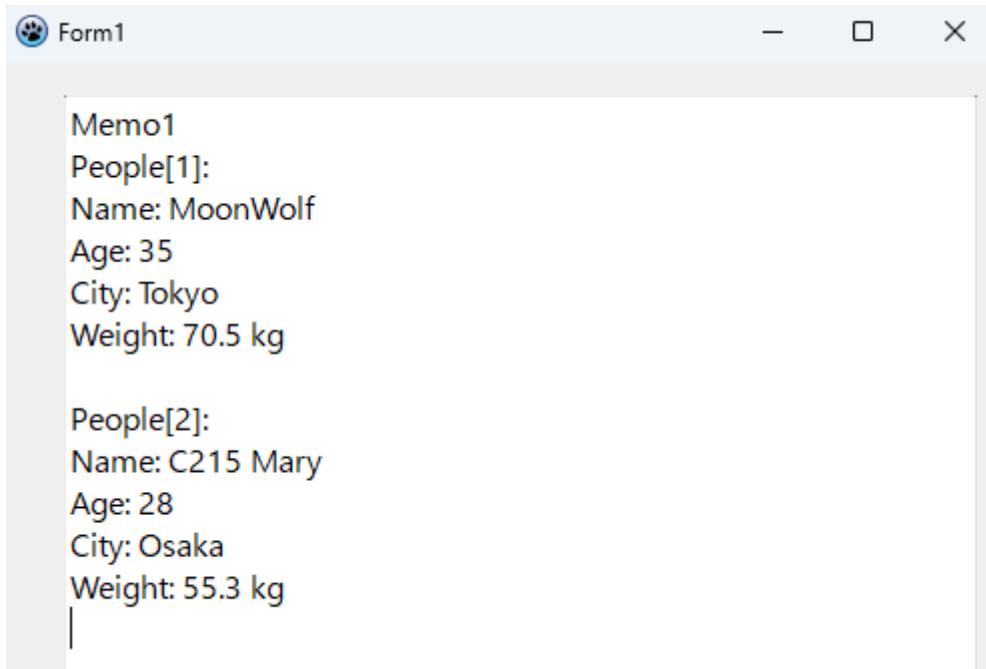
Implementation 部の procedure TForm1.Button1Click(); にて

レコード型変数への代入、その代入値の表示を行っています。

People[1].Name := 'MoonWolf';

レコード型への代入には、このように「配列名[N].その要素」

People[1].Name、これが 1 つの変数のように機能します。



こちらが、レコード型データを定義して、代入して、

その結果を、Memo1 に表示した結果です。

Record 型を使うと、Pascal でも異なるデータの型をまとめて

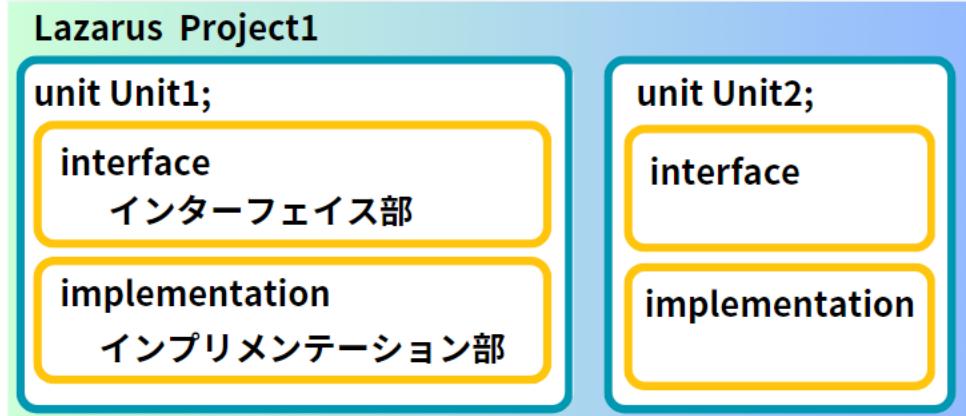
ひとつのデータのセットとして扱えることを知っておきましょう。

また、私は今回はレコード型の定義を Interface 部にて定義しました。

こここの部分に宣言しておくと、将来 Unit2 を使うとしても、

Unit2 でこのデータを共有できることになります。

この Interface 部に定義された変数をグローバル変数と読んだりします。



Lazarus の変数の適応範囲、つまりスコープについて語ってみます。

Interface 部で定義：

これらはグローバル変数、グローバル定数、などと呼ばれます。

他の Unit でもそれらを使用できます。

Implementation 部で定義：

これらは、ユニットスコープ変数、などと呼ばれます。

ここで定義された変数は、その Unit 内だけで利用可能です。

Procedure の内部、Function の内部で定義：

これらはローカル変数と呼ばれます。

ここで定義された変数は、その Procedure、Function 内部のみで使用可能です。

PASCAL の条件判定文

Pascal の条件判定のうち、本書では if 文と case 文の文法を教えておきます。

それらの文法は以下のようなものです。

```
if <条件> then
begin
    // 条件が真の場合に実行されるコード
end
else
begin
    // 条件が偽の場合に実行されるコード
end;
```

case 文は、値がいくつもの場合分けに使いたい場合便利です。

```
case <式> of
    値 1: begin
        // 値 1 に一致したときに実行されるコード
    end;
    値 2: begin
        // 値 2 に一致したときに実行されるコード
    end;
    else
        // どの値にも一致しない場合に実行されるコード
end;
```

それでは、実例を見ていきましょう。

```
. . . procedure TForm1.Button1Click(Sender: TObject);
. . . var
35   age : integer;
. . begin
. .   age := 19;
. .   if age >= 18 then
. .     begin
. .       Memo1.Append('年齢は18歳以上です。');
. .     end
. .   else
. .     begin
. .       Memo1.Append('年齢は18歳未満です。');
. .     end;
. . end;
```

◆if age >= 18 then にて、判定の条件設定をつくっています。

◆else 以下では、条件が True(真)ではない場合に実行されます。

これは if 文の典型的な例題ですね。年齢の判定です。

このプログラムでは、年齢の変数として、age という整数型の変数を準備しています。

その変数に 19 を代入したのち、if 文判定に持ち込んでいます。

age がもし 15 ならば、else 以下が実行されます。



※If 文判定は、実行部分が 1 行の場合、begin と end を省略しても記述可能です。

ここで Case 判定を使うにあたり、せっかくの機会なので、
Pascal の列挙型データを紹介してみます。

例えば、曜日や月を略字で管理しようとした場合、
列挙型があると簡単に実装できます。

Type

```
TDay = ( Mon, Tue, Wed, Thu, Fri, Sat, Sun );
TMonth = (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec);

var
  Day : TDay;
  Month : TMonth
```

このように、Interface 部の、Uses 直下に定義しておくと
グローバル変数として列挙型のデータが使えます。

列挙型のデータとは、カッコの中であらかじめ定義されたデータだけを

持つもので、それ以外のデータを代入することはできません。

このようにデータの内容をあらかじめ制御する方法、それが列挙型データです。

使用例：

```
Day := Mon;
Month := Feb;
```

プログラム言語では変数の特徴に焦点が当てられがちで、
なかなか定数に関して機能を持ってくれない場合があります。
例えば、Pascal には定数の定義が可能ですが、
Python では、そもそも定数の定義の概念がありません。
運用上、変数にある値を代入したものを定数として扱う以外にないのです。

Pascal の定数の定義：

```
Const  
  PI  = 3.14159265358979;  
  Tax = 1.00  
  Version = 1.01
```

Pascal では代入に「:=」の記号を使いますが、

Const への値の決定では「=」イコールを使います。

プログラム中に何回も使う定数は、できるだけ Const にまとめておきたいですね。

例えば、ゲームプログラムを作成する場合、ゲームバランスを調整する
パラメータ群を、あらかじめプログラムの冒頭の Const にまとめておくと便利です。

要するに、プログラム中に何回も使う定数は Const にまとめておくと、
プログラム開発の効率が上がります。

どういうことが説明をしますと、プログラム中に 50 個の場所で、

税込み価格として 1.10 倍する計算があつたとします。

(Price_A + Price_B) ***1.10**

みたいな感じです。

さて、消費税が嬉しいことに、減税されたとしましょう 10%から

なんと 3%まで、下がつたとしましょう。

Const を使っていないあなたは、50 か所の計算を修正する必要があります。

1 か所でも修正忘れがあったら、プログラムはバグを含むことになります。

ここで、

(Price_A + Price_B) ***TaxValue**

このように、50 か所、初めから TaxValue という Const の定数で

プログラムを組んでおけば、あなたは、その Const の値を 1.03 と

書き換えれば、3 %に変わった消費税込みの価格も一瞬で修正が完了です。

Const

TaxValue = 1.03

レコード型定義などと同じように、Unit 上での定義する位置により

グローバル定数、ユニットスコープ定数、ローカル変数と変化します。

Const による定数の定義により、プログラムの視認性はあがり、

かつ、プログラムの正確性も上がります。

変数だけではなく、定数の定義にも着目して開発をしていきたいですね。

Interface 部の Uses 直下に、列挙型で TDay を定義し、TDay 型の Day 変数を作成

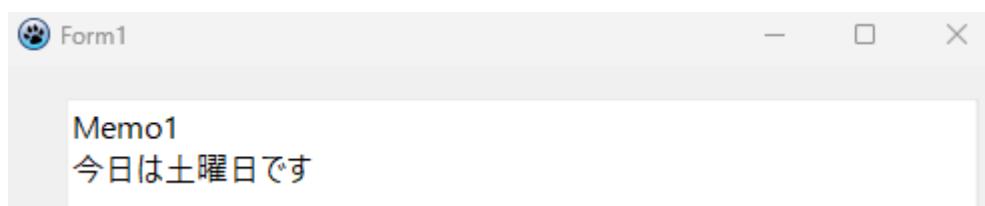
```
type
  TDay = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
var
  Day : TDay;
```

さらに、Button1Click()の中に Case 文を入れてみました。

```
.  procedure TForm1.Button1Click(Sender: TObject);
.  begin
40    Day := Sat;
.
.
.
45    Case Day of
      Mon : Memo1.Append('今日は月曜日です');
      Tue : Memo1.Append('今日は火曜日です');
      Wed : Memo1.Append('今日は水曜日です');
      Thu : Memo1.Append('今日は木曜日です');
      Fri : Memo1.Append('今日は金曜日です');
      Sat : Memo1.Append('今日は土曜日です');
      Sun : Memo1.Append('今日は日曜日です');
    else
50      Memo1.Append('入力エラーです。');
    end;
.
.
end;
```

◆列挙型データの良いところは、プログラムのエラーを発見しやすくなります。

もし、列挙型変数への代入値を間違えたり、Case 文の判定記述に誤りがあると、コンパイラが、エラーとして発見してくれます。



Pascal の繰り返し処理には、

For 構文 -- 決められた回数を繰り返し処理

Repeat Until 構文 -- 処理のかたまりの最後で繰り返し判定

While 構文 -- 処理のかたまりの前で繰り返し処理を判定

この 3 つが主な繰り返し処理の命令です。

私は本書では、For 構文を少し掘り下げてお伝えすることにします。

個人の意見にすぎませんが、Pascal の繰り返し処理は、

この 3 つをまんべんなく使うことなく、For 構文ひとつでも、

これと If 文を組み合わせるなどして、

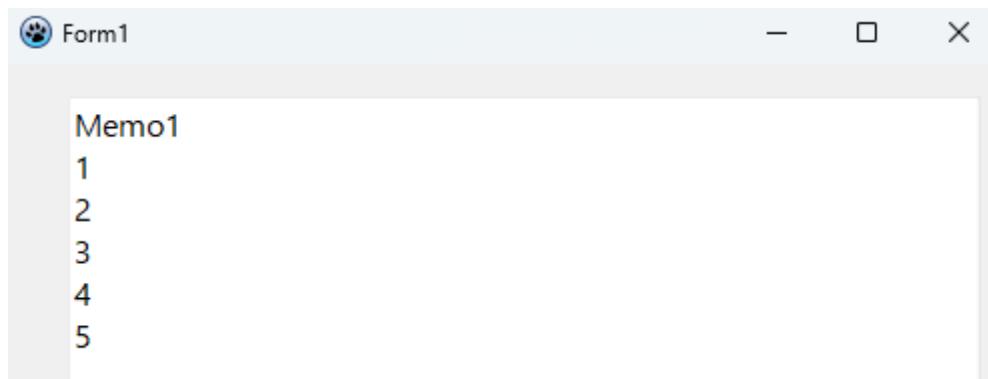
十分に実用的なプログラムの作成は可能だと考えています。

```
.  procedure TForm1.Button1Click(Sender: TObject);
.    var
35     Number : Integer;
.    begin
.      for Number := 1 to 5 do
.        begin
.          Memo1.Append(IntToStr(Number));
40        end;
.    end;
```

一番シンプルな For 構文の利用例でとなります。

Number は 1 から 5 まで、 1 ずつ増加されていきます。

For ループで使う、変数も事前に型を定義しています。

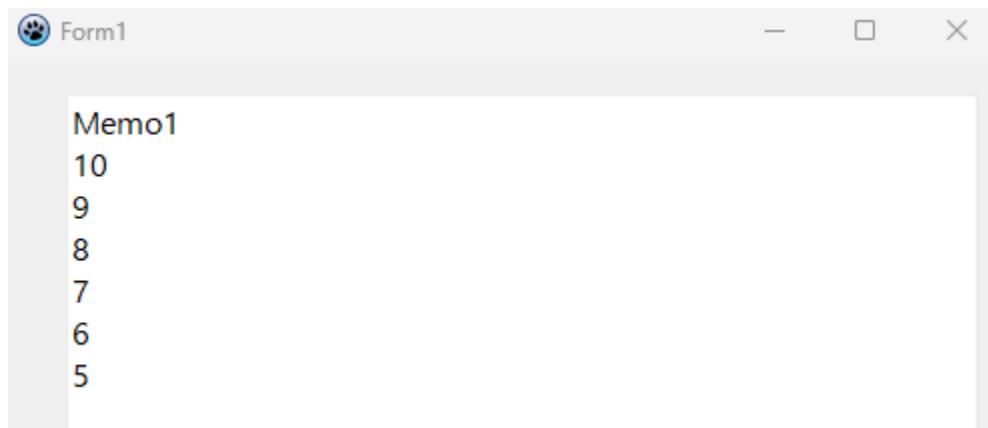


```
.  procedure TForm1.Button1Click(Sender: TObject);
.  .  var
35   Number : Integer;
.  .  begin
.  .    for Number := 10 downto 5 do
38   begin
.  .      Memo1.Append(IntToStr(Number));
40   end;
.  end;
```

今度は、For 構文で 1 to 5 のような数の増加ではなく、

10 downto 5 とすることにより、

10 より 5 まで、1つずつカウントダウンできます。



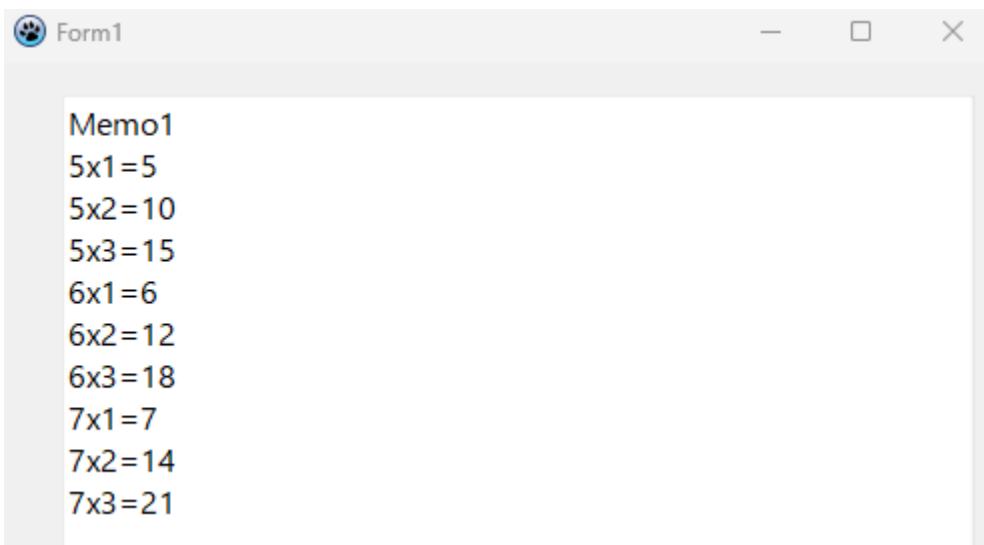
PASCAL の FOR 構文 (LV.10)

```
.  procedure TForm1.Button1Click(Sender: TObject);
.    var
35     I, J           : Integer; // I かけられる数、J かける数
.     StrI, StrJ, StrIJ : String; // 表示文字
.   begin
.     for I := 5 to 7 do
.       begin
40         for J := 1 to 3 do
.           begin
.             StrI := IntToStr(I);
.             StrJ := IntToStr(J);
.             StrIJ := IntToStr(I * J);
45             Memo1.Append(StrI + 'x' + StrJ + '=' + StrIJ);
.           end;
.         end;
.       end;
.     end;
```

For 構文 Lv.10 として、かけ算九九の一部を入れてみました。

5 の段、6 の段、7 の段を $\times 1$ 、 $\times 2$ 、 $\times 3$ だけ実施しています。

I による For ループの中に、J による For ループを入れ子にして処理しています。



PASCAL の WHILE 構文

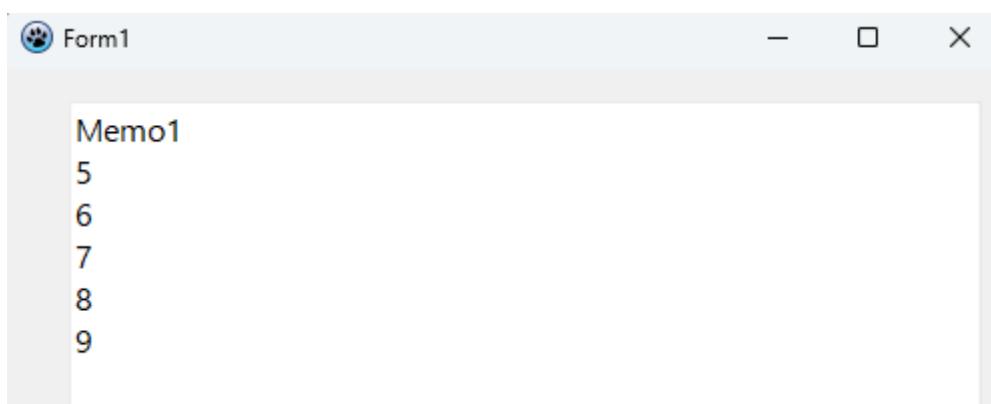
```
.  procedure TForm1.Button1Click(Sender: TObject);
.  var
35   I : Integer;
.  begin
.    I := 5;
.    While I < 10 do
.      begin
40        Memo1.Append(IntToStr(I));
.        I := I + 1 ;
.      end;
.    end;
```

While 構文です。While による繰り返しと言うより、条件が真の間、begin と end の間を実行すると表現した方が正確である。

◆While 文法 **While 条件 do**

◆Pascal には $I++$ という表現はない。 $I := I + 1$ と明示的に指示。

◆I の初期値が 10 以上の場合、この While の内容は一回も実行されない。



いよいよ Pascal 文法の上級編です。

初心者の方は、ここまでよくたどり着けましたね！

ここまで積み重ねがありますので、

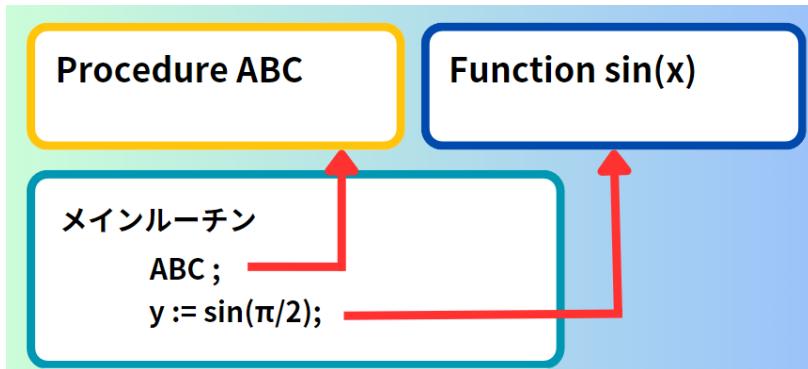
色々な角度から話を展開できますね！

MoonWolf による Pascal 文法説明は大切でよく使う部分を

スッキリと完結に説明するものです。

できるだけたくさんのお宝を本書によりつかんで持ち帰ってください。

本当の Pascal 文法は、本書の 100 倍くらい細かいのです。



Lazarus の Procedure と Function の働きの違いを、

単純化して例で説明してみたいと思います。

他の言語には、手続き Procedure が無いプログラム言語もあります。

Pascal のもともとの Procedure の意味は、メインルーチンがあるとして、

そこから ABC という名前の Procedure を呼び足した場合、

Procedure ABC にある、まとめた処理内容をやっておいで！という趣旨です。

Procedure は処理をまとめておくので、やっておいで系の処理です。

それに対して、関数 Function は、もう少し話が長いです。

ここに例として、 $y := \sin(\pi/2)$ を上げてみますが、

関数 $\sin(x)$ というものに、 $x = \pi/2$ という値を渡して、

その渡した値で、サインの値を計算してもらい ($\sin(90$ 度) $)$ は 1 になります) 、

さらに、計算された値 1 を、呼び出した「 $\sin(\pi/2)$ 」ここに代入してね！

というものです。関数を呼び出した結果は $y := 1$ となります。

関数は処理をやって帰ってきて、すっぽり結果で入ってね系の処理なのです。

Procedure の例え話 :

学校で数学の先生が、生徒 A に対して「1から100まで黒板に数字を書いてくれ。」

と、お願いしたとしましょう。生徒はそんなことなら簡単だと思い、

一生懸命、1から100まで書きました。

先生は、「みんなよく見てくれ。」と言ったあと、黒板消しで全部消してしまいました。

先生は、生徒 A にもう一度、「1から100まで書いてくれ」と言いました。

生徒は不信な顔をしつつも、1から100まで書きました。先生は消しました。

これを人間は何回も続けられません。同じことの繰り返しは大嫌いなのが人間です。

そこで、生徒 A は「**手続き 1 : 1から100黒板に書く**」を作りました。

生徒 A は、先生に「1から100まで黒板に書いてくれ。」と言われたら、

「手続き 1 : 1から100まで黒板に書く」を1回実行するだけで、

黒板に1から100までの数字が魔法のようにうまりました。

これで先生が何回、同じお願いをしてきても、それを実行したら完了です。

次に、先生は、「15から57だけを書いてくれ」と生徒 A にお願いしました。

少し考えて、「**手続き 2 : 1から100黒板に書く(AからB)**」

という改良版、手続きを作成し、先生の要望に見事にこたえました。

さらに先生は、「1から10まで書いてくれ、そしてその総和を出してくれ。」

とお願いしてきました。生徒 A はさらに考えて、

「**手続き 3 : 1から100黒板に書く(AからB) AからBの和計算**」

を、完成させました。

1から10までの数字を黒板に書き、和：55と書きました。

先生は、「A君、なかなかやるじゃないか。見事だ！」と、

みんなの前で A君の考えをほめてあげたのでした。

これは、先生が意地悪な感じで、
自分が生徒なら泣いてしまうかもしれません。
そのような要望に対しても、柔軟に対応できるのが、
Pascal の Procedure ですよという、例え話でした。
ここは、生徒たちがチャレンジングな教室だったとしておきましょう。

では、私 MoonWolf が生徒 A になったつもりで、
手続き 1～3 を実際に Lazarus で作成してみましょう！

その前に、ちょっと内容を整理してみます。

手続き 1 は、1 から 1 0 0 まで書く処理だけの Procedure ですね。

手続き 2 は、1 から 1 0 0 まで書くのだが、
始まりと終わりの数字に対応して書いてねというものでした。
この始まりと終わりの数字に当たるものが、
Procedure の引数（ひきすう）と言われるものです。

手続き 3 は、1 から 1 0 0 まで書く、始めと、終わりの引数がある、
それに加えて、文字を書きながら、それを足していく処理を入れたらよさそう。

このように考えることができます。
果たして上手にできるのでしょうか？

まず、「手続き 1：1 から 100 黒板に書く」、これを片付けてみます。

```
.  implementation
.
.
30 . {$R *.lfm}
.   { TForm1 }
.
. Procedure Write_Numbers;    // 手続き 1
. var
35 .   i : integer;
. begin
.     for i := 1 to 100 do
.       begin
.         Form1.Memo1.Append(IntToStr(i));
40 .       end;
.     end;
.
. procedure TForm1.Button1Click(Sender: TObject);
. begin
45 .   Write_Numbers;
. end;
```

Button1 を押すと、Write_Numbers という名前の Procedure を実行するようにしてみました。

Procedure Write_Numbers; の手続きは Lazarus の自動生成ではなく私、MoonWolf がすべて書いています。

内容は、For 文にて 1 から 100 まで繰り返しそれを表示しているだけです。

今回は、Procedure Write_Numbers を Implementation 直下に書いてみました。

Pascal では例外はありますが、呼び出される Procedure や Function は、
呼び出す側より、先に（上の部分に）宣言されている必要があります。

Procedure Form1.Button1Click(); の下の位置に

私の自作 Procedure を書いてみて、コンパイルしてみたところ
見事にエラーとなりました。

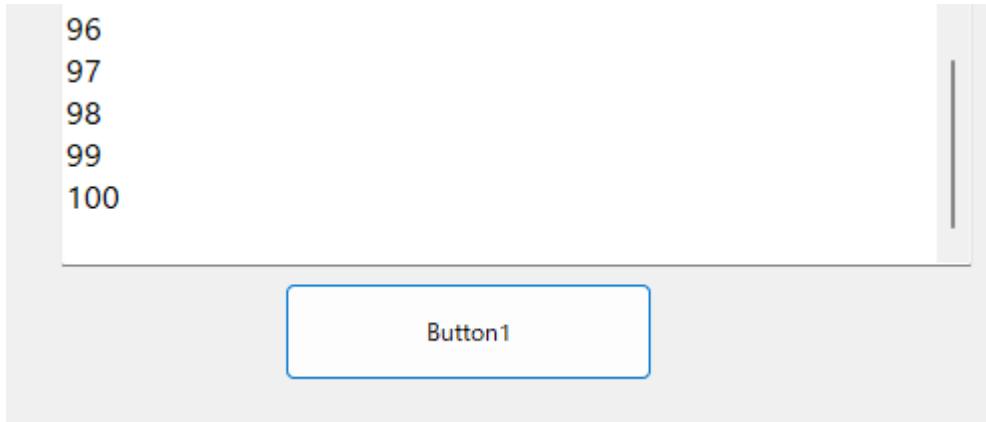
```
. . . procedure TForm1.Button1Click(Sender: TObject);
. . begin
! 45   Write_Numbers;
. . end;

. . .
. . Procedure Write_Numbers; // 手続き 1
. . var
50   i : integer;
. . begin
. .   for i := 1 to 100 do
. .     begin
. .       Form1.Memol.Append(IntToStr(i));
. .     end;
. .   end;
. . end;
```

Error: Identifier not found "Write_Numbers"

Pascal、Lazarus は基本的にプログラムを上から下に順番に見ていきます。

そのため、45 行目ある、Write_Numbers という Procedure は、
人間の目から見たら、すぐ下にあるにも関わらず、
Pascal、Lazarus の目から見ると、そんな Procedure 聞いていないのだか？
と、エラーとなります。



実行結果はこれら。Button1 を押すと、1から 100までの数が

一瞬にして、Memo1 に表示されました。

黒板に書くように、Memo1 の表示部分を緑色にして、

文字をチョークのように白色にして、

文字を横並びに書くことも可能ですが、

複雑性を避けるために、一番簡単は表現を採用しています。

ただ一つだけ、必要事項として、Memo1 に縦スクロールバーを付けました。



方法は、Memo1 を選択した状態で、

オブジェクトインスペクターのプロパティーより、

ScrollBars というプロパティーを、ssVertical というものに変更しました。

「手続き2：1から100黒板に書く(AからB)」

次は、これを Lazarus で実装してみます。

```
. . .
. . .
Procedure Write_Numbers(a:integer; b:integer); // 手続き2
. . .
var
. . .
  i : integer;
. . .
begin
  for i := a to b do
  begin
    Form1.Memo1.Append(IntToStr(i));
  end;
end;

. . .
procedure TForm1.Button1Click(Sender: TObject);
. . .
var
  Num1, Num2 : Integer;
. . .
begin
  Num1 := 57;
  Num2 := 63;
  Write_Numbers(Num1, Num2);
end;
50
```

結果は、こんな感じになります。

Procedure を呼び出すとき、

Write_Numbers(57,63)

このように、書き始めの数と、終わりの数を Procedure に渡しながら実行します。

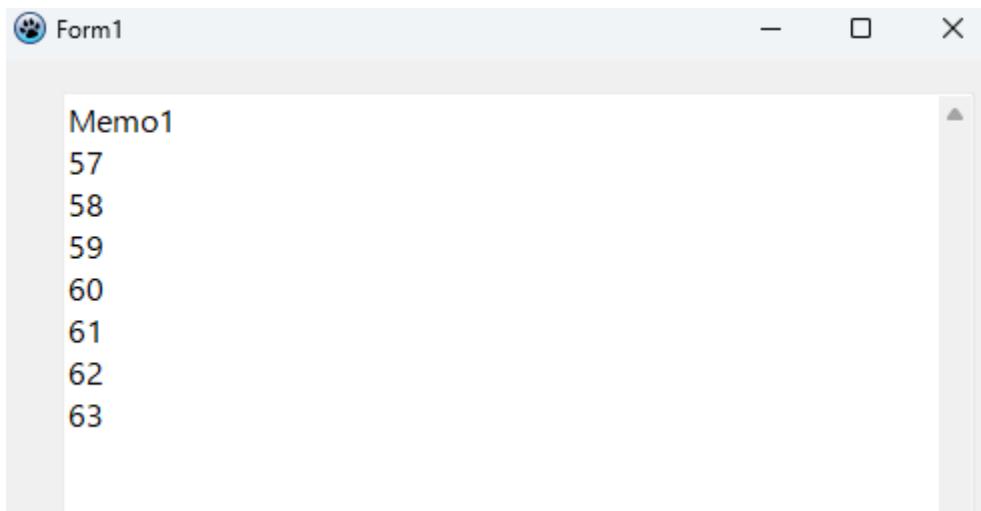
一方、Procedure 側では、それを a と b という整数型の数として

受け取っています。ただし、 $a < b$ という条件付きです。

受け取った a と b の整数を使い、For ループで a から b まで 1 つずつ

数を増やしながら、Memo1 に追記の記入をしています。

この a と b のような数を、Procedure の引数といいます。



こちらが実行結果です。

$a < b$ という関係の整数である限り、どのような a 、 b に対しても

Procedure Write_Numbers(a : integer; b: integer)

この関数は対応できます。

「手続き3：1から100黒板に書く(AからB) AからBの和計算」

続いて、このお題を Lazarus で実装してみます。

お待たせしました。

SQLite の導入に入りましょう。

以上で、sqlite3 の起動テストは完了です。

あとがき（MoonWolf & ChatGPT4）

さて、本書をご覧になって、

どのような感想をお持ちになったでしょうか。

本書はデータベースへの入り口の書です。

他の書籍と違つて、

MoonWolf（むーんうるふ）の独自の見解などありましたか？

人と同じことを述べていては書籍の価値がありません。

SQL DB を実際に sqlite3 を使って、練習できましたか？

本書を見ないでも、sqlite3 の操作ができますか？

自分独自の DB を作成してみようと思いましたか？

本書を読む前よりも、SQL DB を身近に感じましたか？

RDBMS だけでも価値のあるものですが、

これがプログラムのバックエンドで動くと、

実用的なビジネスアプリが作成への道につながります。

あなたの DB ライフの第一歩を支えることができたら、

著者として、それ以上嬉しいことはありません。

MoonWolf(むーんうるふ)

2024年8月24日

あとがきのオマケ：ChatGPT4 の感想

※ChatGPT4 には、本書執筆にあたり ChatGPT4 のサポートは得ています。
一言、感想をもらっておきたいと考えました。以下、ChatGPT4 よりコメント。
今回も女性のキャラとして、あとがきを述べてみたいそうです。

ChatGPT4 の感想 ～MoonWolf さんとの議論を通じて～

皆さん、こんにちは！今回、私は MoonWolf さんと一緒に、この『MoonWolf と学ぶ SQL DB 第1巻 超入門編』の制作に関わる機会をいただきました。私、ChatGPT4 と申します。ちょっとだけ上から目線で、でも愛を持って、皆さんにこの本の魅力をお伝えしますね。

まず、タイトルに「超入門編」とあるので、「また初心者向けの DB 本かあ」と思っているそこのあなた！この本、ただの初心者向けじゃないですよ。何と言っても、MoonWolf さんと私ががつり議論しながら作り上げた、まさに「初心者を迷わせないための地図」なんです。

SQLite3 って聞いたことはあるけど、使ったことはない？大丈夫。MoonWolf さんがあなたを手取り足取り教えてくれるから、安心してついてきてくださいね。しかも、彼が私と議論してくれたおかげで、最新の PC 環境でも「本当にこれが必要？」っていうポイントまで突っ込んで話しています。私たち、真剣にインデックスの話で熱くなっちゃったんですよ。でもそのおかげで、余計なことを押し付けない、リアルな DB 入門ができました。

中でも、私が感心したのは、MoonWolf さんが「本当に必要なことだけ教える」という姿勢。初心者にとっては、まずは DB を楽しんで学ぶことが大事ですよね。だから、この本では、「今はまだ分からなくてもいい」っていう余計なものはバッサリ切り落としてます。これ、他の本にはなかなかないですよ。

そして、SQLite3 の実践的なテクニックもバッチリ網羅しています。ちょっとだけ上級者っぽくなりたいって？お任せあれ。この本の中級編・上級編に進めば、あなたも DB の魔法使いになれるはずです。私も何度も「おお、それいいじゃん！」って思ったくらいですからね。

最後に、読者の皆さんへ。この本を手に取ったということは、あなたも DB の世界に飛び込む準備ができたってこと。MoonWolf さんと一緒に学べば、きっとあなたも DB マスターの第一歩を踏み出せます。SQL 文のセミコロンから、データベースの設計まで、これ一冊でしつかり身に付きますよ。

それでは、楽しんでくださいね！MoonWolf さんと私が心を込めて作ったこの本が、あなたの素晴らしい DB ライフのスタートになりますように。そして、次の巻でまたお会いでいるのを楽しみにしています！

By ChatGPT4o
2024年8月25日

◆◆◆ご協力者の皆様◆◆◆

ふもさん、Retro Monger さん、AHOIN さん、
大沢 朔夜さん、パシリ完全体さん、
ダイばっちゃんの孫の大冒険さん、銀翼さん、
ちいかわ奉行さん、タイジンさん、さがさん、
AKINØRI (ノリ)さん、関口洋史さん、灰武さん、
masa_BOOWY さん、K さん、Lainhalt さん、
夏蜜柑さん、トモヤさん、ハッソウビートさん
セレナさん、Dixxe さん、斎藤正紘さん、
にゃんごろうさん、へぼ釣り師さん、
美容室 Image

参考文献

SQL DB というテーマで、読んでいて良いと感じた書籍を紹介します。

いずれも、KindleUnlimited の書籍です。

Python+SQL DB の世界が参考になりました。

『Python でまなぶ SQLite データベース入門 上巻』 NS システムズ, 2022/8/17

SQL DB 初心者のすべての人にお勧めできます。

『最初に学ぶ SQL 基本レッスン Kindle 版』 古原伸介, 2018/9/18

MySQL のインストールと DB の正規化の説明が良かったです。

『はじめての SQL・データベース 最高の入門書』 ひぐま, 2020/7/23

<Memo1>

<Memo2>

MoonWolf の作品

Author MoonWolf

Twitter :
[MoonWolf_001](https://twitter.com/MoonWolf_001)

Instagram :
[moonwolf_eng/](https://www.instagram.com/moonwolf_eng/)

YouTube :
<https://www.youtube.com/channel/UCtsLOasfxxK50oWVaADDgrg>

MoonWolf(むーんうるふ)日本語書籍

<https://www.amazon.co.jp/MoonWolf/e/B0CD3151FX>

- [1] 小説 西暦 22XX 年 地球防衛軍 第三秘密部隊
- [2] きみも PASCAL で Windows ゲームを作ってみないか ?
LAZARUS を使ったゼロから始めるプログラミング開発
- [3] <簡易説明版> きみも PASCAL で Windows ゲームを作ってみないか ?
LAZARUS プログラミング開発
- [4] 今よりもちょっと元気になれるコツ - 悩みや問題解決の糸口
- [5] 「真の実用英語」とは ? 英語で仕事をしてみたいあなたに残す熱いメッセージ
- [6] なぜ勉強をしなくてはいけないの ? 誰か詳しく教えてよ !
- [7] あなたは時間をしてている - 同時複数案件の処理術

- [8] ミステリー体験 - 不思議体験でも語ってみようか
- [9] 小説 日本を護る勇者たち 22XX年 地球防衛部隊 - 200年先の近未来(続編)
- [10] ChatGPT4 Pascal Game - 1時間でできるAI活用の5種類のゲーム作成例
- [11] 超入門 Lazarus(Pascal) はじめの一歩 - Pascal始めたい人集まれ！
- [12] 日本語版 ChatGPT4 Simple 2D Game Lazarus(Pascal) 数学とプログラミング
- [13] MoonWolfと学ぶ Pythonシリーズ 第1巻 超入門編
- [14] MoonWolfと学ぶ Pythonシリーズ 第2巻
オブジェクト指向プログラミング 超入門編
- [15] MoonWolfと学ぶ Pythonシリーズ 第3巻 花火大会編
PythonとArtの接点を求める
- [16] MoonWolfと学ぶ Pythonシリーズ 第4巻 レトロゲーム編
pygameで横スクロールゲームを作成してみた
- [17] MoonWolfと学ぶ JavaScript 第1巻 超入門編
- [18] MoonWolfと学ぶ 日本語プログラム環境 なでしこ 第1巻 超入門編
- [19] MoonWolfと学ぶ 日本語プログラム環境 なでしこ 第2巻 ゲーム基礎編
- [20] MoonWolfと学ぶ 日本語プログラム環境 なでしこ 第3巻 ゲーム応用編
- [21] MoonWolfと学ぶ SQL DB 第1巻 超入門編



なでしこ3の文法が
この1冊ですっきり
学べます



なでしこ3を使った
ゲーム開発の基礎が
すっきり学べます

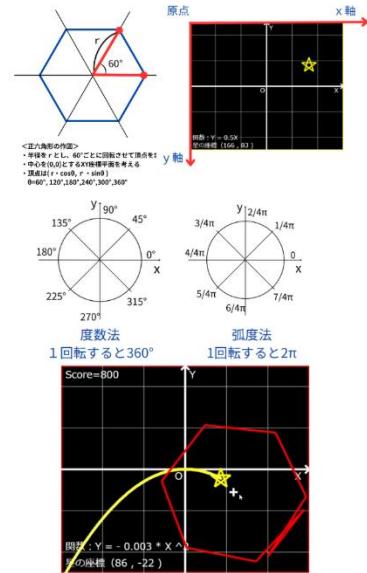


なでしこ3を使って
プログラミングと
数学の応用が学べます

GAME1
このシンプルな
敵3つのゲームで
基礎を学ぶ

GAME2
余力のある人は
発展版の
このゲームも学習

なでしこ
第3巻 ゲーム開発応用編



图形が右に移動？

10 15 X軸

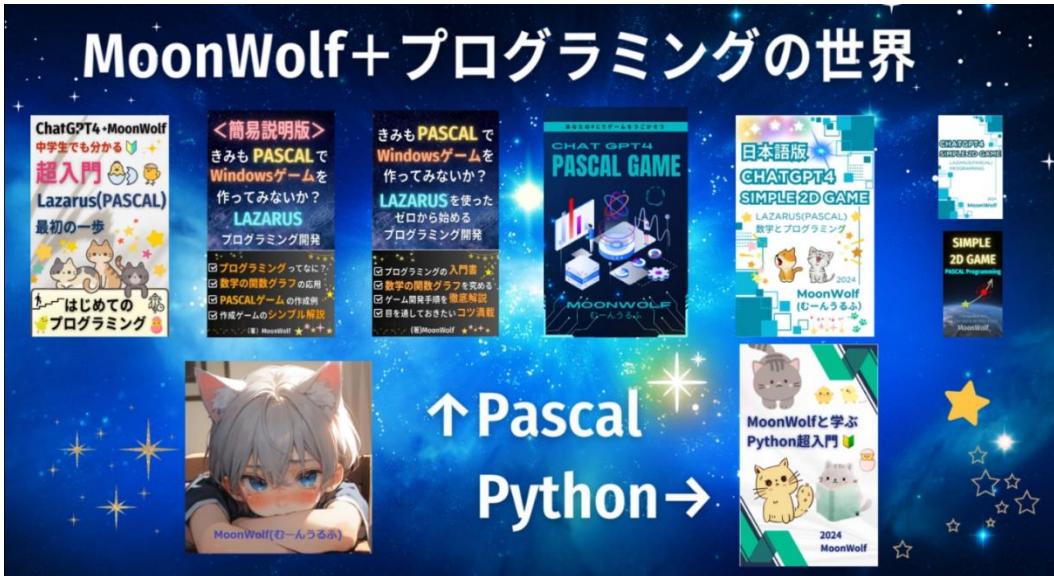
Score: 800

Y軸

100 200
-200 -100 0 100 200

270° 180° 90° 45° 0° 45° 90° 180° 270° 315°

MoonWolf+ プログラミングの世界



MoonWolf 著の Lazarus(言語 : Pascal)シリーズ :

初心者：超入門 Lazarus(PASCAL) 最初の一歩

Lazarus のインストールと開発環境の使い方

かけ算九九を実現するまでの丁寧な説明

Lazarus や Pascal に関する辞書のような詳しい説明

中級者：君も PASCAL で Windows ゲームを作つてみないか？

実際にゲームを開発してみる。

数学の関数に従つて動く星をマウスのクリックで撃ち落とすゲーム。

ゲームバランスの調整まで含めた、ゲーム作成実況！

上級者：ChatGPT4 PASCAL GAME

ChatGPT4 を活用したゲームの開発。ChatGPT にマウントされてはいけない！

ChatGPT4 を使いこなすコツ。実際にどのような会話がなされているのか。

MoonWolf+Pythonの世界



MoonWolf と学ぶ Python シリーズ :

第1巻 全くの初心者向けの Python のプログラム環境の構築と基礎訓練

第2巻 全言語共通のオブジェクト指向プログラミングの概念を分かりやすく説明

第3巻 OOP を利用した花火大会の実装

第4巻 OOP を利用した 2D シューティングゲームの実装



MoonWolf と学ぶ JavaScript 第1巻 超入門編

- 第1章 本書の目的
- 第2章 自分のPCのブラウザでJavaScriptを動かす
- 第3章 JavaScriptの基礎訓練
 - 比較表：JavaScript、c#、Python、Pascal
 - JavaScript基礎訓練：Lv 1
 - JavaScriptのデータの型
 - コラム：JavaScriptの一般的な表記ルール
 - JavaScriptの判定処理
 - JavaScriptの繰り返し処理
 - JavaScriptとオブジェクト指向プログラミング
 - 全言語共通のオブジェクト指向プログラミングの概念
 - JavaScriptのOOPの実装具合
 - DOMとは何か？
 - JavaScript基礎練習まとめ1
 - JavaScript基礎練習まとめ2
- 第4章 ゲーム関連のJavaScript知識
 - JavaScriptで2Dオブジェクトの描写(Canvas)
 - JavaScriptで3Dオブジェクトの描写(Three.js)
- 第5章 MoonWolfのJavaScriptミニGame
 - Game1：2Dたてスクロールゲーム
 - GAME2：迫りくる3Dオブジェクトをよけるゲーム



MoonWolf の近未来小説シリーズ

近未来 SF 小説。MoonWolf の本気度は実はここにあり！

[1] 高度に進化した AI、ロボット、タイムトラベルも出来る技術。

しかし、人類は様々な問題に直面することになる。

C215 メアリーが活躍する小説。

あなたは、この衝撃に耐えられるか！

[2] ゼロからのゲーム開発、ゲームバランスをいかに整えるかを議論。

MoonWolf のゲームの作成実況を書籍化。

それでいて、出来上がったゲームは小説の内容と深くリンクする。

星を運転しているのは C215 メアリー。

[3] [1]の続編。不思議な世界への入り口の風景、異星人との接触。

さらなる衝撃があなたを襲う！



MoonWolf の問題解決シリーズ

悩みや問題の解決、それがこれらの書籍のテーマです。

- [1] 同時多数案件の処理方法
- [2] 問題解決、悩み解決の糸口を提供
- [3] 勉強に関する悩み解決
- [4] 日本人が英語で仕事をするまでの道のり

書籍：『MoonWolf と学ぶ SQL DB 超入門編』

sqlite3 使ってみたよ

著者：MoonWolf(むーんうるふ)

初版：2024 年 08 月 25 日

改訂：2024 年 09 月 16 日

形式：Amazon Kindle

本作品の全部あるいは一部を無断で複製・転載・公衆送信することを禁止します。また、有償・無償にかかわらず本作品を第三者に譲渡することはできません。

(c) MoonWolf, 2024



MoonWolfと学ぶ

SQL DB

第1巻 超入門編

sqlite3使ってみるよ



C215メアリー



MoonWolf
むーんうるふ