Delphi Programming Cheatsheet

A concise reference for Delphi developers, covering essential syntax, data types, control structures, and object-oriented programming concepts.

Syntax and Basic Data Types

Program Structure

```
program ProjectName;

uses
  // List of units (libraries) used by the program
  Forms, Dialogs;

{$APPTYPE GUI}

begin
  // Program execution starts here
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Explanation:

- `program ProjectName;` - Declares the program name.
- `uses` - Specifies units (libraries).
- `{$APPTYPE GUI}` - Compiler directive for GUI applications.
- `begin ... end.` - The main program block.

Data Types

`Integer`	Signed integer. Common subtypes: `Shortint`, `Smallint`, `Longint`, `Int64`, `Byte`, `Word`, `Longword`.
`Real`	Floating-point number. Subtypes: `Single`, `Double`, `Extended`, `Currency`.
`Char`	Single character (e.g., `'A'`).
`String`	Sequence of characters. `AnsiString` (default), `UnicodeString`.
`Boolean`	`True` or `False`.
`Variant`	Can hold any data type; late binding.

Variable Declaration

```
var
  VariableName: DataType;
  AnotherVariable: Integer;
  Name: String;
```

Explanation:

- `var` - Keyword to declare variables.
- `VariableName` - The name you choose for the variable.
- `DataType` - The data type of the variable (e.g., `Integer`, `String`, `Boolean`).

Control Structures

Conditional Statements

If-Then-Else:

```
if condition then
begin
  // Code to execute if the condition is true
end
else
begin
  // Code to execute if the condition is false
end;
```

Case Statement:

```
case variable of
  value1: // Code to execute if variable = value1;
  value2: // Code to execute if variable = value2;
else
  // Code to execute if variable doesn't match any value
end;
```

Looping Constructs

`For` loop

```
for i := startValue to endValue do
begin
  // Code to execute in each iteration
end;
```

`While` loop

```
while condition do
begin
  // Code to execute while the condition is true
end;
```

`Repeat-Until` loop

```
repeat
  // Code to execute
until condition;
```

Break and Continue

- `Break`: Exits the current loop.
- `Continue`: Skips the rest of the current iteration and proceeds to the next.

```
for i := 1 to 10 do
begin
  if i = 5 then Continue; // Skips when i is 5
  if i = 8 then Break;    // Exits when i is 8
  ShowMessage(IntToStr(i));
end;
```

Procedures and Functions

Procedure Declaration

```
procedure ProcedureName(Parameter1: DataType;
Parameter2: DataType);
begin
  // Procedure body
end;
```

Explanation:

- `procedure` - Keyword to declare a procedure.
- `ProcedureName` - The name of the procedure.
- `(Parameter1: DataType)` - Parameters passed to the procedure.

Function Declaration

```
function FunctionName(Parameter1: DataType;
Parameter2: DataType): ReturnType;
begin
  // Function body
  Result := ReturnValue; // Assign the return
value
end;
```

Explanation:

- `function` - Keyword to declare a function.
- `FunctionName` - The name of the function.
- `(Parameter1: DataType)` - Parameters passed to the function.
- `ReturnType` - The data type of the value returned by the function.
- `Result` - Special variable to assign the return value.

Parameters

`Value` parameters	Passed by value; changes to the parameter inside the procedure/function do not affect the original variable.
`Var` parameters	Passed by reference; changes to the parameter inside the procedure/function affect the original variable.
`Const` parameters	Passed by constant reference; optimized and prevents modification of the original variable.
`Out` parameters	Like `var`, but the initial value of the parameter is discarded. Used for returning values.

Object-Oriented Programming

Class Declaration

```
type
  TMyClass = class(TObject)
  private
    // Private fields (data)
  protected
    // Protected fields and methods
  public
    // Public fields and methods
    constructor Create;
    destructor Destroy; override;
    procedure MyMethod;
  end;
```

Explanation:

- `type` - Keyword to define a new type.
- `TMyClass` - The name of the class (convention: start with `T`).
- `class(TObject)` - Inherits from `TObject` (base class).
- `private`, `protected`, `public` - Access specifiers.
- `constructor Create;` - Constructor.
- `destructor Destroy; override;` - Destructor (override required).

Methods

Constructor

```
constructor TMyClass.Create;
begin
  inherited Create;
  // Initialization code
end;
```

Destructor

```
destructor TMyClass.Destroy;
begin
  // Clean-up code
  inherited Destroy;
end;
```

Method

```
procedure TMyClass.MyMethod;
begin
  // Method implementation
end;
```

Inheritance

```
type
  TMyDerivedClass = class(TMyClass)
  // ...
  end;
```

Explanation:

- `TMyDerivedClass` inherits from `TMyClass`.
- Derived classes inherit fields and methods from the base class.