Bernhard Wurm

Inkl. Downloads

Mit Syntax-Highlighting!

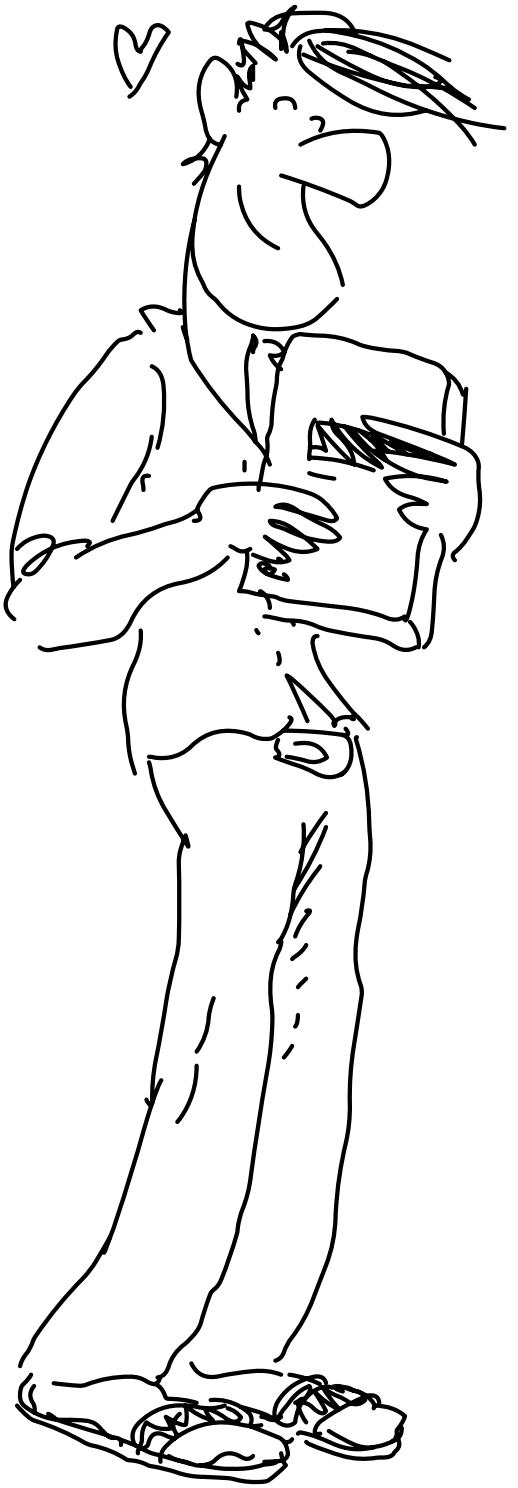
Schrödinger programmiert C#

Das etwas andere Fachbuch

- Von den Sprachgrundlagen über XAML bis zur komplexen Anwendung
- Ob LINQ, Generics oder C# 13: Hol dir die Juwelen aller Versionen!
- Durchblicken, mitmachen und genießen!

VIERTE AUFLAGE

Rheinwerk Computing



MIT TALENT, KATZEN-
PHOBIE UND LÄSSIGEM
SCHUHWERK BESTACH
SCHRÖDINGER DIE
RHEINWERK-JURY.
FÜR IHN GEHT JETZT
EIN TRAUM IN ERFÜLLUNG.

Liebe Leserin, lieber Leser,

DU HAST DIR WAS VORGENOMMEN:

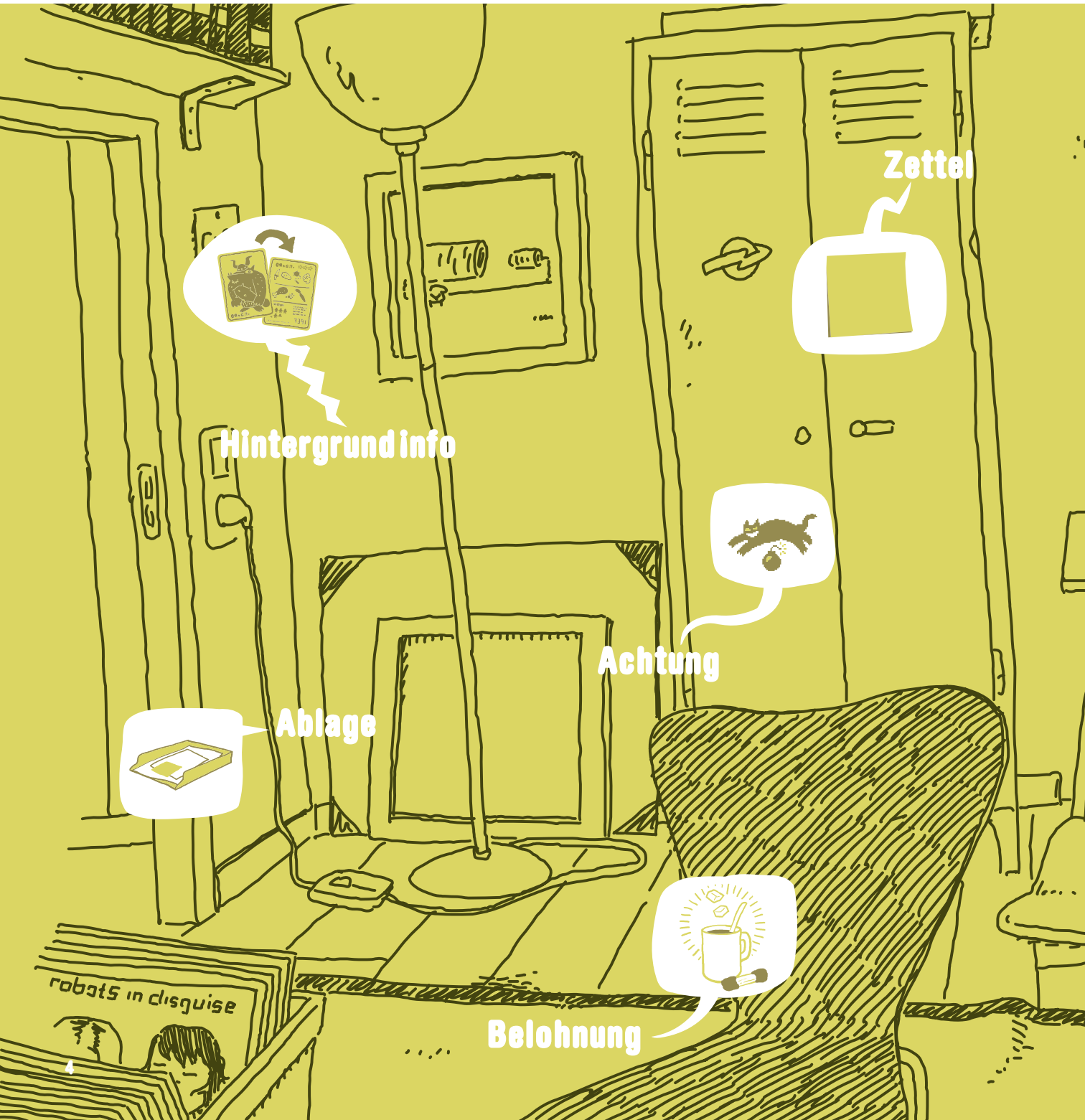
C#.

Da sind wir dabei.

Lernen musst du zwar selbst, aber wir geben unser Bestes, um dich zu unterstützen:

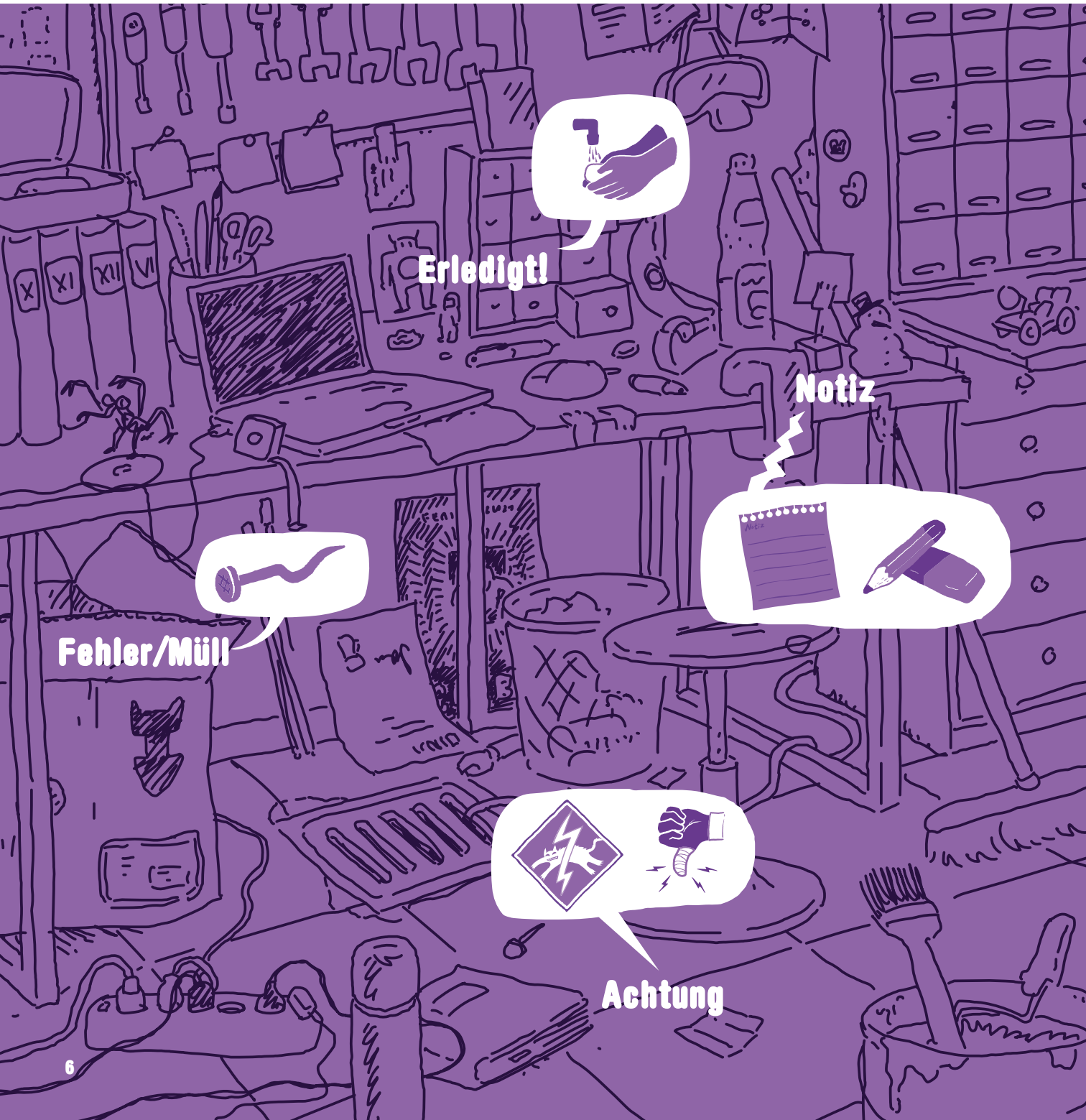
Na dann: Viel Erfolg!

Schrödingers Büro



Die nötige Theorie, viele Hinweise und Tipps

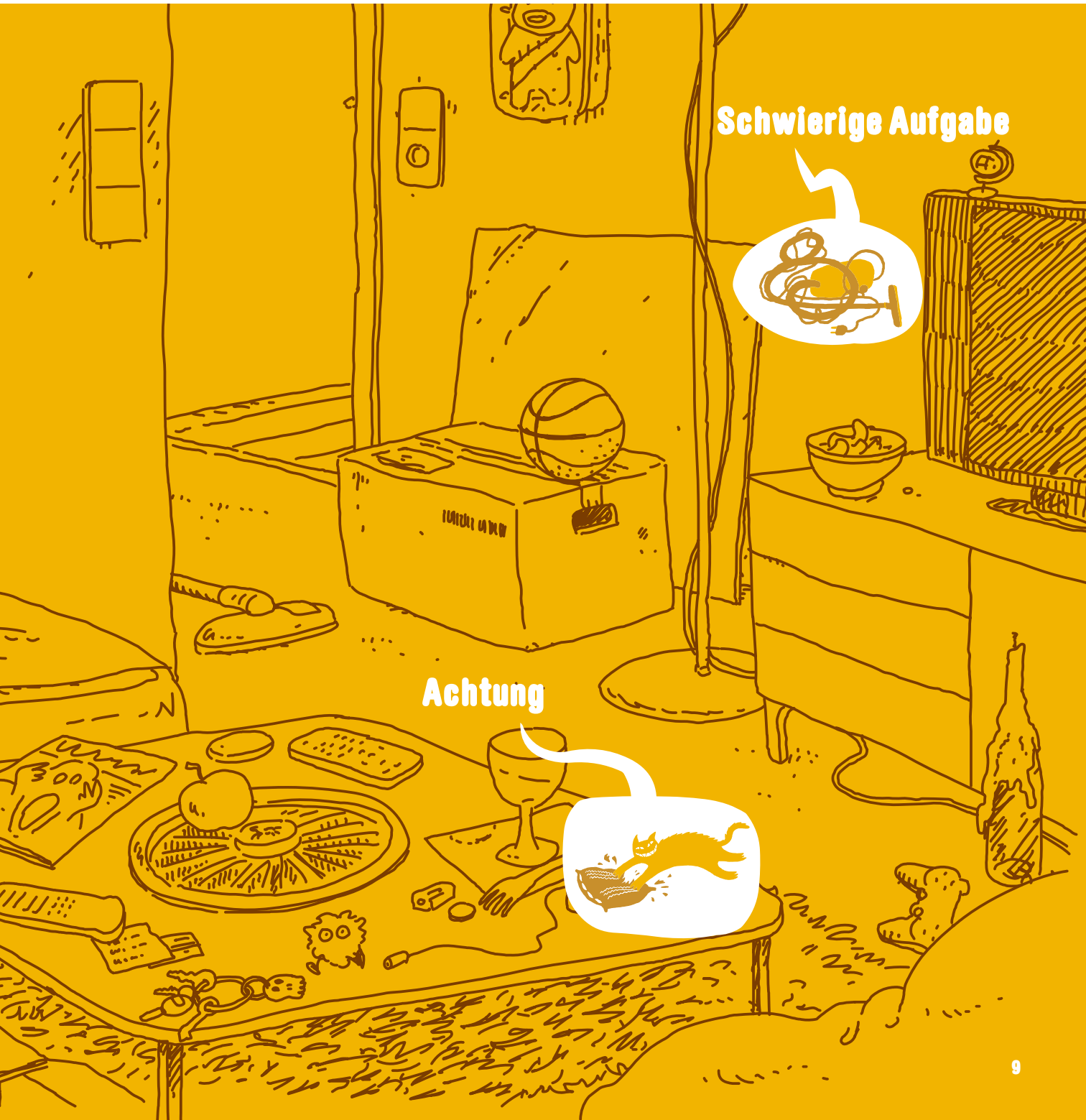
Schrödingers Werkstatt



Unmengen von Code, der ergänzt, verbessert und repariert werden will

Schrödingers Wohnzimmer

Viel Kaffee, Übungen und die verdienten Pausen



—EINS—

Compiler und Entwicklungs-umgebungen

Ein guter Start ist der halbe Sieg

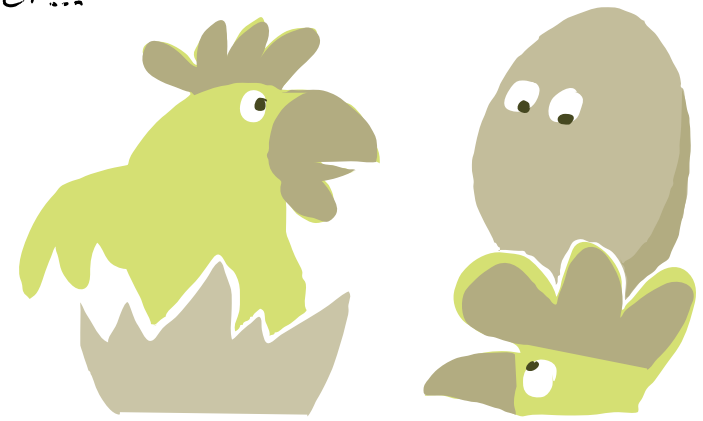
Schrödinger steht am Anfang seines neuen Jobs als C#-Entwickler. Sein Problem: Er kann noch gar kein C#. Sein erster Schritt zur Lösung: Er hat sich Hilfe geholt. Und er hat richtig Lust auf die Sache. Beste Voraussetzungen also. Jetzt ist der zweite Schritt an der Reihe: Installieren! Aber was?

Compiler und Compiler

Hallo Schrödinger, es ist schön, dass es dir schon unter den Nägeln brennt. Und wir werden auch gleich loslegen mit dem großen Spaß. Doch wie du weißt, braucht ein guter Handwerker auch ein **gutes Handwerkszeug**. Also besorgen wir dir jetzt gleich mal ein paar Dinge, die du als angehender C#-Programmierer brauchst.

Das wichtigste Programm, um mit einem C#-Programm richtig loszulegen, ist der **Compiler**. Dieser übersetzt deinen verhältnismäßig gut lesbaren Programmcode in eine andere, für den Computer einfacher verständliche Sprache.

Ich brauche also ein Programm, um mein Programm für den Computer verständlich zu machen? Das ist ja wie mit der Henne und dem Ei...



Fast noch schlimmer:
Es ist erstmal noch kein Maschinencode. Dein Programmcode wird in die **Intermediate Language** (IL-Code genannt) übersetzt. Erst **beim Ausführen des Programms** wird nun der IL-Code in **Maschinencode** übersetzt. Dies übernimmt ein weiterer Compiler – der sogenannte **Just-in-time-Compiler** (JIT-Compiler genannt).

Warte mal. Ich brauche einen Übersetzer von C# in den IL-Code und dann wieder einen von IL-Code in Maschinencode? Das ist ja umständlich!

Das wirkt vielleicht so, aber du musst wissen, **dass C# eine von mehreren Programmiersprachen ist, die auf der Entwicklungsplattform .NET laufen**. Es gibt auch andere Programmiersprachen, wie zum Beispiel Visual Basic.NET, F# usw., die ebenfalls nicht direkt in den Maschinencode übersetzt werden, und somit kannst du – sofern du so etwas willst – einfach zwischen verschiedenen Programmiersprachen wechseln und auch Bausteine von anderen Sprachen verwenden.

C#, F#, Visual Basic, dann gibt es auch noch C, C++, Java. Glaubt denn jetzt jeder, dass er eine **eigene** Programmiersprache entwickeln muss?

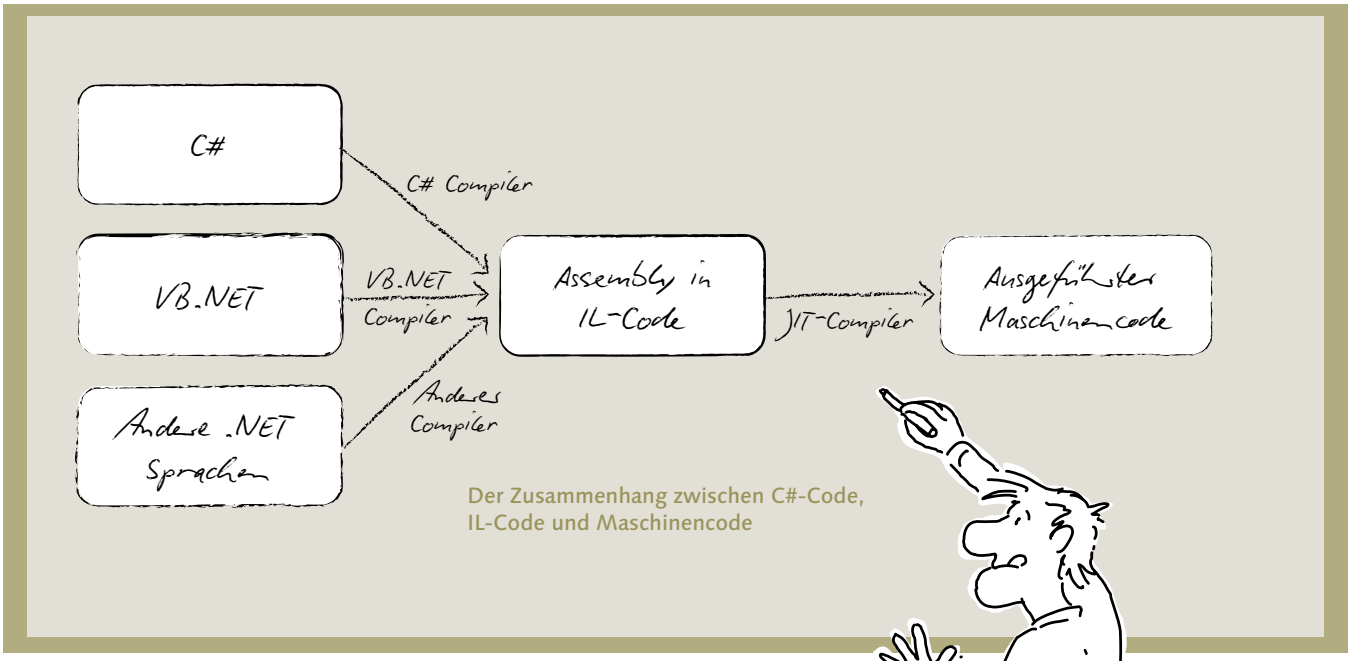
Nachdem du also mithilfe des C#-Compilers dein Programm kompiliert hast, erhältst du eine Datei, die IL-Code beinhaltet. Diese Datei nennt man **Assembly**. Ein Assembly kann, muss aber keine ausführbare EXE-Datei sein. Es kann auch eine Bibliothek – also eine DLL-Datei – sein, die verschiedene Funktionen beinhaltet, die wiederum von anderen Dateien verwendet werden können.

Aber ich dachte immer, in einer EXE-Datei steht Maschinencode? In echt steht aber IL-Code drin, und trotzdem kann sie ausgeführt werden?

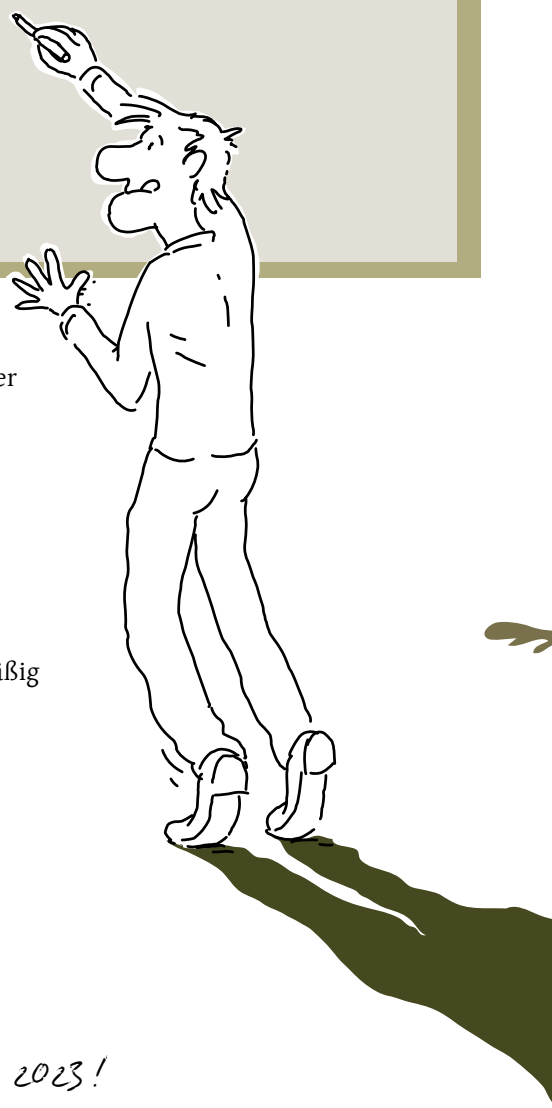
Du kannst dir das so vorstellen, dass der erste Befehl in dieser EXE-Datei ein Verweis auf die Common Language Runtime (kurz CLR) ist. Diese startet den JIT-Compiler, der den Code direkt, noch bevor dieser ausgeführt wird, in richtigen Maschinencode übersetzt und ihn dann ausführt.

[Hintergrundinfo]
Immer, wenn Code eines Assemblys ausgeführt werden soll, tritt der JIT-Compiler in Aktion. Der ist so optimiert, dass immer nur der aktuelle Codeblock kompiliert wird und nicht mehr. Das kompilierte Ergebnis wird zwischengespeichert und nur noch dieser schnelle Code ausgeführt.

Außerdem hat der IL-Code den Vorteil, dass er nicht nur in Windows funktioniert, sondern beispielsweise auch unter Linux und unterschiedlichen Rechnerarchitekturen (x86 oder ARM) laufen kann, da er **plattformunabhängig** ist und **erst bei der Ausführung in den Maschinencode übersetzt** wird.



Der Zusammenhang zwischen C#-Code, IL-Code und Maschinencode



Du hast übrigens bereits einen alten Compiler auf deinem Windows-Rechner installiert.

Der Compiler ist die **`csc.exe`** (csc steht für C-Sharp-Compiler) und in dem Verzeichnis **`C:\Windows\Microsoft.NET\Framework64\v4.0.30319`** zu finden.

> Je nach Windows-Version unterscheidet sich auch die standardmäßig installierte .NET-Framework-Version etwas.

Na, dann lass uns loslegen!

Schrödinger, du willst doch das Aktuellste lernen, das es gibt, oder?

Natürlich, ich bin doch nicht von 2023!

[Hintergrundinfo]

Die erste Version des .NET Frameworks und C# wurde 2002 veröffentlicht. 2016 veröffentlichte Microsoft eine Neuentwicklung mit dem Namen .NET Core. Dieses legte den Fokus auf die plattformübergreifende Entwicklung, sollte effizienter sein und Altlasten entfernen. Alle Neuentwicklungen werden seither dort gemacht und nur mehr teilweise im .NET Framework nachgezogen.

Wir werden uns also auf die aktuelle .NET-Version stützen und alles, was .NET Framework heißt, ignorieren.

Core wurde inzwischen wieder aus dem Namen entfernt.

[Ablage]

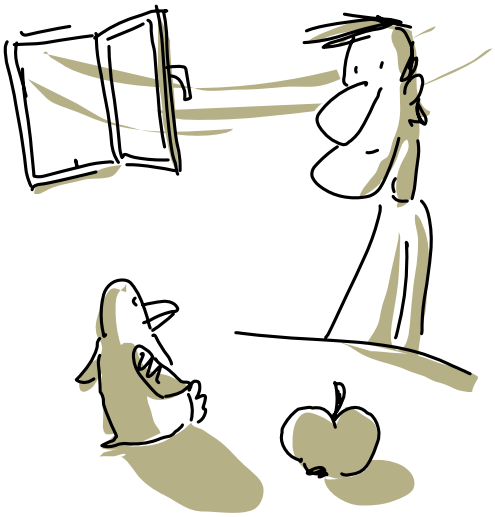
Die aktuelle .NET-Version erhältst du kostenlos unter *https://dotnet.microsoft.com* oder mit der Installation der Entwicklungsumgebung Visual Studio unter *https://visualstudio.microsoft.com*.

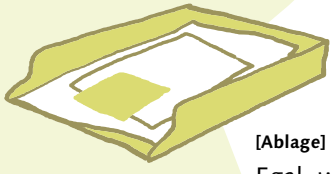
Du brauchst eine IDE!

Obligatorisch ist zwar nur der Compiler, aber in einem x-beliebigen Texteditor macht programmieren keinen Spaß. Auch das Kompilieren kann komfortabler sein als auf der Kommandozeile. Die Entwicklungsumgebung – auch gerne IDE genannt – unterstützt dich bei deinen Projekten.

[Begriffsdefinition]
IDE steht für Integrated Development Environment, also eine integrierte Entwicklungsumgebung. Dieses Programm unterstützt dich bei der Entwicklung und ermöglicht auch das Kompilieren deiner Programme.

Es gibt selbstverständlich verschiedene Entwicklungsumgebungen, mit denen du C# programmieren kannst. Und du hast natürlich die Auswahl. **Selbst unter Linux** kannst du, wenn du willst, mit MonoDevelop C#-Programme entwickeln. Eine ebenfalls gute Alternative für die .NET- Entwicklung unter Windows, Mac und Linux ist **Visual Studio Code** von Microsoft. Um Anwendungen für den Windows 11-Store entwickeln zu können, benötigst du selbst Windows 11, und am besten eignet sich hierbei natürlich die Entwicklungsumgebung **direkt von Microsoft**.
Daher schlage ich dir vor, du konzentrierst dich auf **Visual Studio**. Auch hier gibt es eine kostenlose Edition – die sogenannte **Community Edition.**





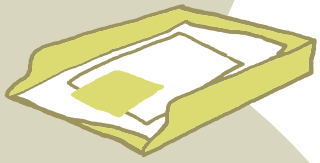
[Ablage]
Egal, welche Entwicklungsumgebung du verwendest, C# ist C# und bleibt C# – die Syntax ist also überall ident. Die IDEs unterscheiden sich im Allgemeinen bei Komfortfunktionen für den Entwickler.

Visual Studio Community Edition

Visual Studio von Microsoft gibt es in verschiedenen Versionen. Die großen, kostenpflichtigen Enterprise-Versionen integrieren sämtliche Funktionalitäten, die das Entwicklerherz begehrt und auch einige, die du nie im Leben brauchen wirst. Es steht aber auch eine sehr gute kostenlose Versionen zur Verfügung: die **Visual Studio Community Edition**. Auch sie bietet alles was du brauchst, um **Webanwendungen, Desktopanwendungen oder Universal Windows Platform Apps** zu erstellen. Sogar wenn du für Android oder für den Apfel entwickeln möchtest, ist dies mit der Community Edition möglich. Dies ist in Wahrheit eine kostenlose Professional Edition, die für **kleine Teams, Open Source Projekte und die Wissenschaft** gedacht ist.

Noch bin ich eine Ein-Mann-Armee, das ist klein genug.

[Hintergrundinfo]
Der lange Weg der Namensfindung zu **Universal Windows Platform Apps** startete mit dem Begriff Metro Apps, der nach einem Rechtsstreit fix zu **Windows Store Apps** wurde. Mit Windows 8.1 kamen die sogenannten **Universal Apps**, die eine teilweise gemeinsame Codebasis für Windows Store und Windows Phone erlaubten. **Windows 10** und die Windows Universal Platform perfektionieren dieses Ziel: Es läuft die absolut gleiche App auf dem Raspberry Pi wie auch auf einem Tablet oder Desktop – und heißt zu Recht **Universal** Windows Platform App.



[Ablage]
Die Entwicklungsumgebungen kannst du unter **www.visualstudio.com** herunterladen.

Installiere dir bitte die **Visual Studio Community Edition**, und dann geht es los.

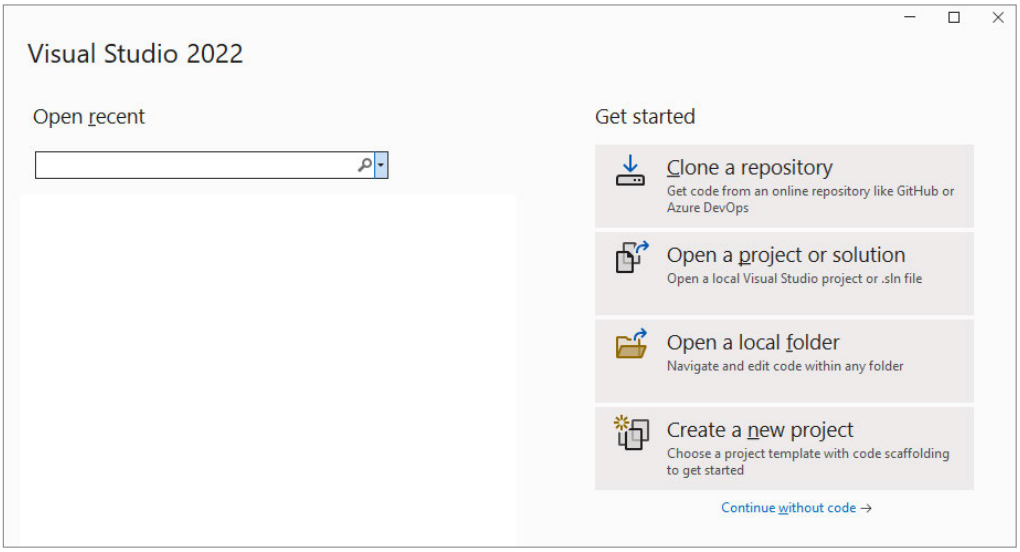
Der Spaß geht los!

Die Software ist wie gesagt kostenlos. Du musst dich jedoch mit deinem Microsoft-Account an Visual Studio anmelden, damit es länger als 30 Tage funktioniert. Das kannst du gefahrlos machen. Einfach anmelden bzw. kostenlos registrieren, falls du noch keinen Account hast, und der Spaß kann losgehen.

Ist ja klar, dass die wieder alle meine Daten haben wollen. Aber gut – kostet ja nichts.

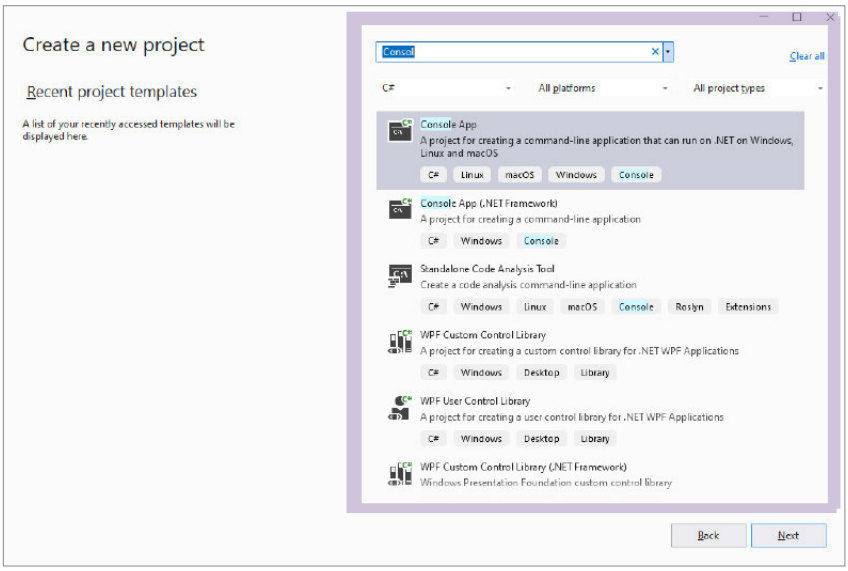
Nachdem du die Installation abgeschlossen hast, kannst du dein erstes Projekt starten.

Anmelden mit dem Microsoft-Account



Visual Studio – Startbildschirm

Durch einen Klick auf **`Neues Projekt`** erhältst du eine Auswahl mit den möglichen Projekttypen für diese Visual-Studio-Version. Diese Version beinhaltet alles, was du für klassische Desktopanwendungen benötigst, also ältere Windows-Forms-Anwendungen, modernere WPF-Anwendungen, Konsolenprogramme und auch Klassenbibliotheken, die dir ermöglichen, Funktionen in DLL-Dateien zu packen und projektübergreifend zu verwenden.



Auswahl der Projekttypen von Visual Studio

Dein erstes Projekt

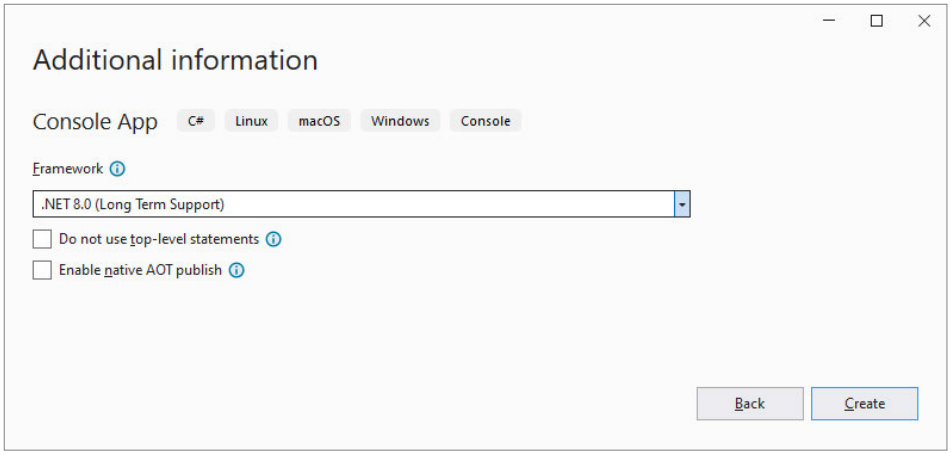
Am besten beginnst du zunächst mit einer **Konsolenanwendung**. Die eignet sich gut für die ersten Schritte, da du nicht von schönen, bunten Oberflächen abgelenkt wirst.

»Console App« gibt's da zweimal.

Viele Projekttypen gibt es einmal mit und einmal ohne den Zusatz ».NET Framework«. Wir verwenden ausschließlich die **ohne Zusatz**. Dafür sind wir auch mit Linux etc. kompatibel und modern unterwegs. Dein Projekttyp ist also einfach **Console App**.

Genau, lassen wir das alte Zeug einfach weg. Find ich gut.

Nachdem du den Projektnamen, zum Beispiel **`HalloSchrödinger`**, und den Speicherort ausgewählt hast, wirst du aufgefordert, eine .NET-Version zu wählen. Denn auch von der neuen Implementierung gibt es natürlich unterschiedliche Versionen.



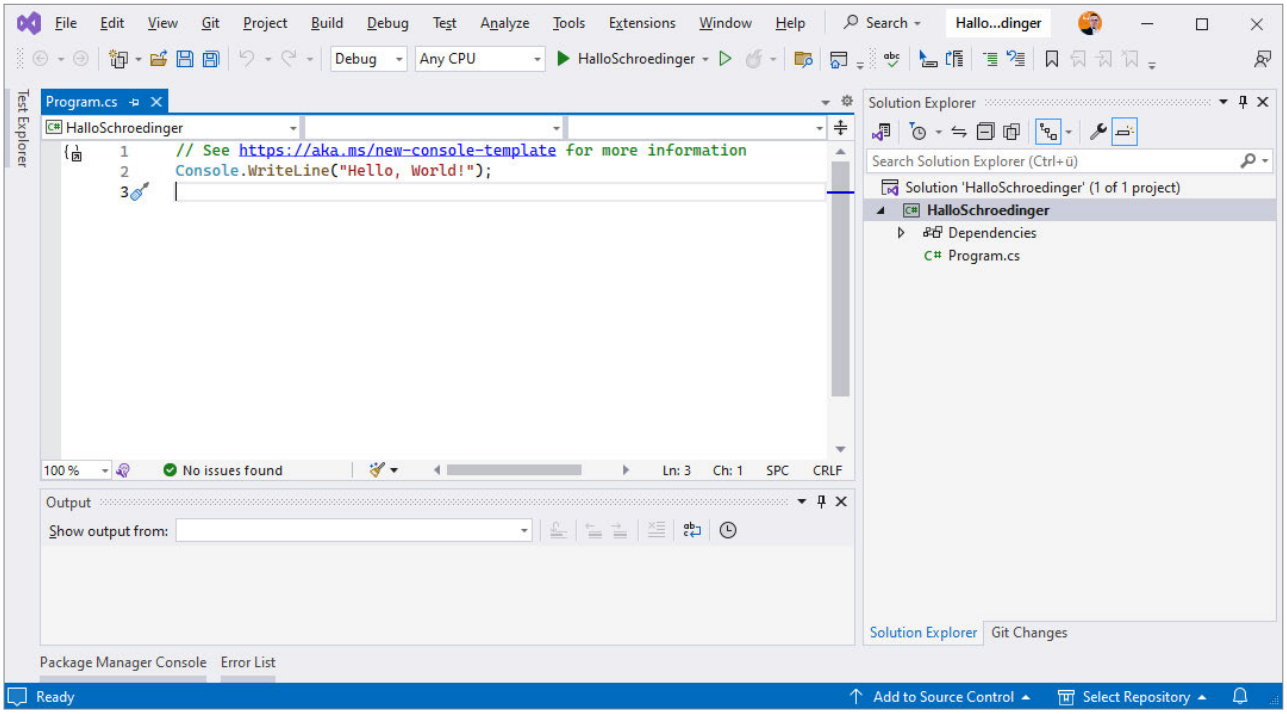
Framework-Auswahl

[Achtung]

Long Term Support vs. Standard Term Support

Etwa alle zwei oder drei Versionen wird eine Version mit **Long Term Support (LTS)** veröffentlicht. Diese hat den Vorteil, dass es offiziell eine längere Unterstützung gibt, nämlich 3 Jahre. Bei Versionen mit Standard Term Support (STS) endet diese bereits nach 18 Monaten. Das bedeutet auch, dass du diese Version früher upgraden musst und beispielsweise in der Azure Cloud etc. ohne Upgrade relativ frühzeitig nicht mehr betreiben kannst.

Im großen Hauptfenster siehst du deinen Programmcode. Das ist dein Spielplatz, wo du an deinem Code arbeitest, ihn verfeinerst, Fehler suchst und möglicherweise manchmal fast verzweifelst. Im rechten Bereich findest du den Projektmappen-Explorer oder den sogenannten Solution-Explorer, wenn du Visual Studio auf Englisch installiert hast. Dieser zeigt dir alle Dateien in deiner Projektmappe an.



Grundgerüst einer Konsolenanwendung

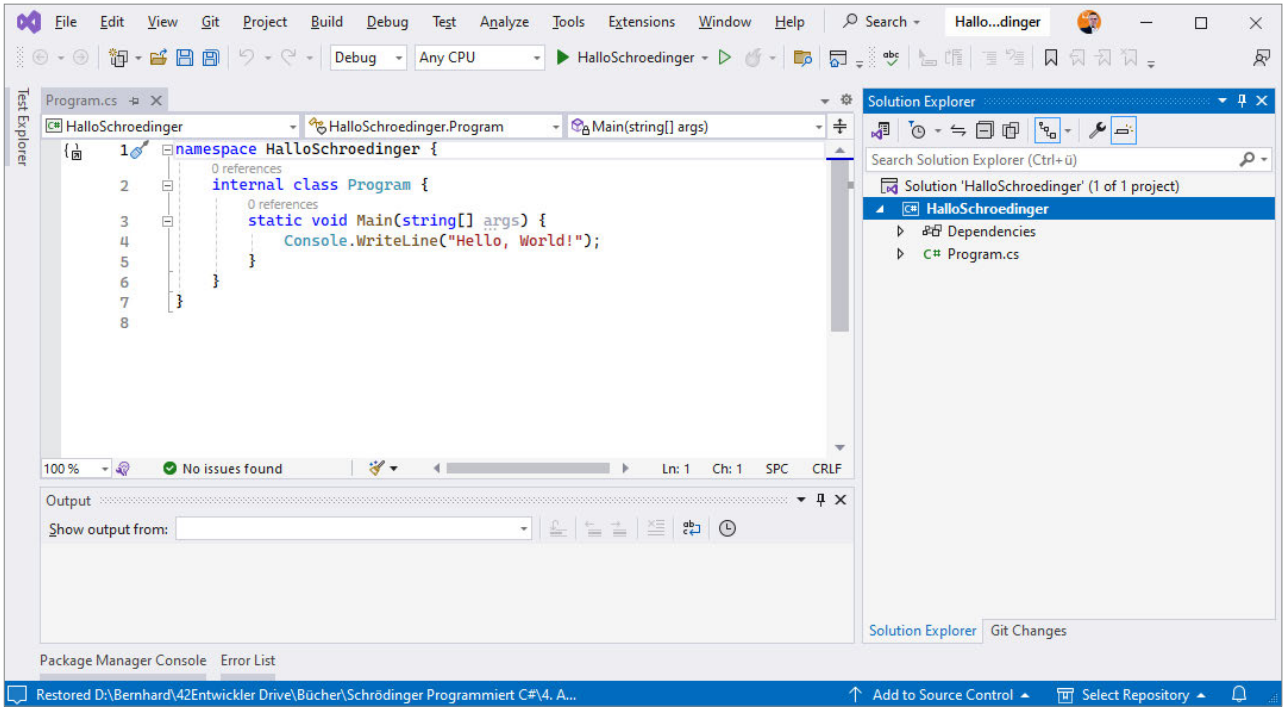
Eine Software ist wie ein **großer Schrank**. Er sieht zunächst wie ein großes kompaktes Etwas aus, aber wenn du ihn öffnest, hast du viele verschiedene Schubladen und Fächer mit Dingen darin.

Eine Software besteht ebenfalls aus vielen einzelnen Teilen und oftmals aus **verschiedenen Projekten**. Die Projektmappe besteht also aus Projekten und ein Projekt wieder aus vielen verschiedenen Dateien und Programmcodes.

Am Ende ergibt das alles zusammen deine Software.

[Notiz]

Jedes Programm startet automatisch an einem bestimmten Einstiegspunkt. Dieser ist im Allgemeinen als **`Main`**-Funktion bekannt. Inzwischen wird dir dieses Codegerüst rundherum nicht mehr angezeigt und du startest direkt in der ersten Zeile mit deinem Code. Wenn du die **`Main`**-Funktion sehen möchtest, wähle bei der Framework-Auswahl das Häkchen **Do not use Top-Level statements**.



Gleicher Programmcode ohne Top-Level-Statements

Was bedeutet das Ganze hier?

Lass uns einfach starten
und das Feature Top-Level-Statements verwenden!

Ja, einfach ist gut!

```
// See https://aka.ms/new-console-template for more information
```

*1 Das ist nur ein Kommentar, der wird vom Compiler ignoriert. Einzeilige Kommentare starten mit `//`.

```
Console.WriteLine("Hello, World!");
```

*2 Der Text **`"Hello World"`** wird mithilfe der Funktion **`Console.WriteLine`** in der Konsole angezeigt.

[Notiz]
Texte (sogenannte Strings) sind in C# immer innerhalb von Anführungszeichen zu setzen, ansonsten weiß der Compiler nicht, dass es sich um einen Text und nicht um irgendetwas anderes handelt.

```
Console.WriteLine("Bitte Namen eingeben:");
var name = Console.ReadLine();
Console.WriteLine("Hallo " + name);
```

*3 Natürlich kannst du mit den Elementen, die der Benutzer eingibt, auch arbeiten. Dazu musst du eine Variable definieren. Die Variable ermöglicht dir, Werte im Arbeitsspeicher des Computers zu halten und damit zu arbeiten. Damit der Compiler weiß, dass es sich um eine Variable handelt, schreib einfach **`var`** vor den Variablennamen, den du frei wählen kannst. Später werde ich dir die verschiedenen Variablentypen zeigen, aber dazu gleich mehr. Der **Variablen** kannst du nun mittels **`Console.ReadLine`** einen Wert zuweisen, und zwar genau den Wert, den der Benutzer eingibt.

Wie du schon gesehen hast, kannst du mittels **`Console.WriteLine`** Text ausgeben. Es gibt aber auch ein **`Console.ReadLine`**, womit du Text einlesen kannst, den der Benutzer eingibt. Durch **`Console`** kannst du auf das Konsolenfenster zugreifen und entsprechend Eigenschaften setzen, wie zum Beispiel den Fenstertitel, oder auch Funktionen aufrufen, etwa zur Ein- oder Ausgabe.

Das scheint ja einfach zu sein, aber warte kurz, das will ich gleich mal ausprobieren.

Gerne. Drück einfach auf **`Start`**, um das Programm zu kompilieren und auszuführen.

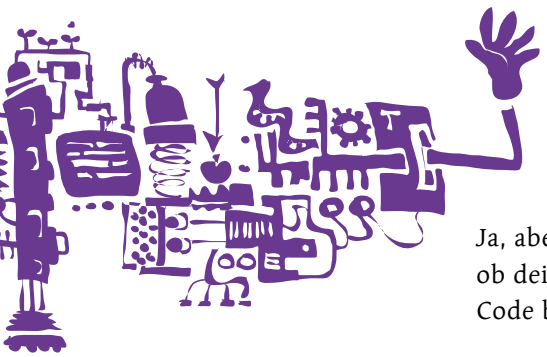
Bravo, Schrödinger. Du hast dein erstes Programm geschrieben.
Und dein Programm hat dich gleich begrüßt.

Toll!

Wenn du in Visual Studio auf den **`Start`**-Knopf drückst, wird automatisch der Compiler angeworfen. Außerdem wird das Programm gleich ausgeführt, und der JIT-Compiler springt zu deiner **`Main`**-Funktion, kompiliert diese und führt deinen Code nativ als Maschinencode aus.

Nativ?

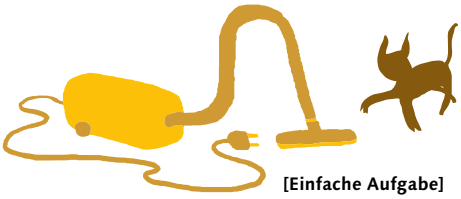
[Notiz]
Nativ bedeutet, der Code wird direkt in Maschinencode von der CPU ausgeführt und nicht über weitere Umwege oder interpretierte Zwischensprachen.



Puh, der Computer treibt ganz schön viel Aufwand für eine einfache Begrüßung.

Ja, aber der Ablauf ist immer der gleiche, unabhängig davon, ob dein Programm aus 1.000.000 Zeilen oder lediglich drei Zeilen Code besteht.

Theorie und Praxis



[Einfache Aufgabe]

Zeig mir doch mal, ob du alles verstanden hast. Verbinde die Satzteile mit einem Bleistift.

Der C#-Compiler erzeugt …

… eine IDE.

Der JIT-Compiler erzeugt …

… gibt etwas auf der Konsole aus.

Visual Studio ist …

… liest eine Zeile von der Konsole.

Console.ReadLine …

… Maschinencode.

Console.WriteLine …

… IL-Code.

Lösung:

Der C#-Compiler erzeugt IL-Code.
Der JIT-Compiler erzeugt Maschinencode.
Visual Studio ist eine IDE.
Console.ReadLine liest eine Zeile von der Konsole.
Console.WriteLine gibt etwas auf der Konsole aus.

Wunderbar, Schrödinger. Die Theorie hast du verstanden. Und dass du die Praxis auch verstanden hast, kannst du anhand der nächsten Aufgabe beweisen:

[Schwierige Aufgabe]
Schreibe doch ein kleines Programm, das Vor- und Nachnamen nacheinander abfragt und anschließend den Namen vollständig ausgibt.

Wenn ich **Console.Readline()** schreibe, startet das Programm nicht und es ist rot unterkringelt.

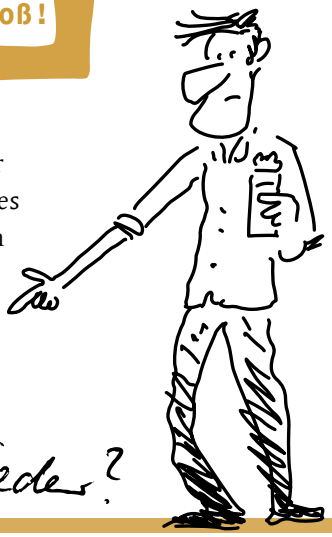
Was läuft da falsch?

[Achtung/Vorsicht]
C# ist Case-Sensitive. Achte also auf die Groß- und Kleinschreibung der Befehle und Funktionsaufrufe.

```
Console.Readline() X
```

Das L in ReadLine() gehört groß!

Das Programm startet nur, wenn der Compiler keinen Fehler findet, nur dann kann er es übersetzen und starten. Dennoch kann es sein, dass dein Programm nicht tut, was es soll, weil du zwar keinen Fehler in der Syntax (im Satzbau) hast, doch einen logischen Fehler. In dem Fall hilft dir Debugging weiter.

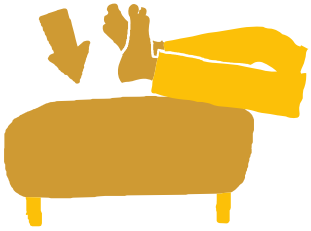


Was ist jetzt Debugging wieder?

Debuggen hilft dir bei der **Fehlersuche**. Du kannst Punkte setzen, an denen das Programm einfach stehen bleibt, damit du dir Variablenwerte ansehen kannst, usw.

Die Lösung:

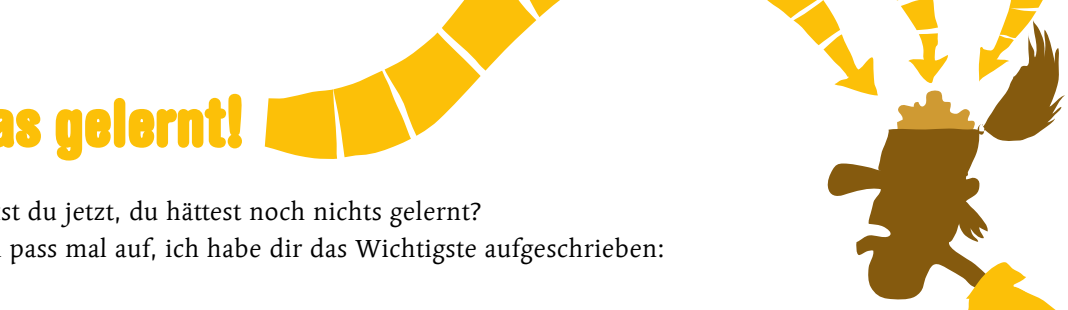
```
Console.WriteLine("Vorname eingeben:");
var vorname = Console.ReadLine();
Console.WriteLine("Nachname eingeben:");
var nachname = Console.ReadLine();
Console.WriteLine(vorname + " " + nachname);
```



[Belohnung]
Wenn dein Programm funktioniert, dann kannst du jetzt gerne ein bisschen von deiner Karriere als Spieleentwickler bei Blizzard träumen.

Was gelernt!

Denkst du jetzt, du hättest noch nichts gelernt?
Dann pass mal auf, ich habe dir das Wichtigste aufgeschrieben:



- C# ist eine der vielen Programmiersprachen, die auf der .NET-Plattform basieren. Auch Visual Basic, F#, Python.NET etc. basieren auf .NET, wobei C# nach wie vor das Flaggschiff dieser Plattform ist.
- Die aktuellen .NET-Versionen sind Open Source und laufen auch auf macOS oder Linux.
- Die Intermediate Language ist die Zwischensprache, in die jede .NET-Sprache übersetzt wird. Egal, ob du C# oder Visual Basic programmierst, nach dem Kompilieren kommt IL-Code heraus.
- Visual Studio ist eine von mehreren möglichen Entwicklungsumgebungen, mit denen du C# programmieren kannst. Wenn du willst, kannst du sogar in der Kommandozeile kompilieren.
- Die LTS-Versionen von .NET haben mindestens 3 Jahre Support, während die STS-Versionen lediglich 18 Monate Support genießen.
- Mit **F5** startest du den Debugger. Der hilft dir bei der Fehlersuche und wird noch dein Freund werden!

—ZWEI—

Datentypen und deren Behandlung

Ein netter Typ

Wie heißt es so schön, guten Freunden gibt man doch einen Kaffee—oder? Ich würde sagen, ich stelle dir ein paar freundliche Typen vor, und am Ende trinken wir eine schöne heiße Tasse Kaffee.

Das klingt gemütlich, findet Schrödinger und entspannt sich. Gemütlich? Sieben Typen allein für Zahlen? Konvertieren, kompatibel, Kommentare? Doch Schrödinger bleibt locker und lernt dabei sogar noch, mit Kamelen umzugehen.

INHALTSVERZEICHNIS

Kapitel 4: Sexy Unterwäsche – von kleinen Teilen bis gar nichts

Strings, Characters und Nullable Types

Kapitel 5: Eine endliche Geschichte

Enumerationen

Kapitel 6: Teile und herrsche

Methoden

Kapitel 7: Klassengesellschaft

Objekte, Eigenschaften und Sichtbarkeiten

Kapitel 8: Es wird Zeit für Übersicht!

Namespaces

Kapitel 9: Erben ohne Sterben

Objektorientierte Programmierung

Kapitel 10: Abstrakte Kunst

Abstrakte Klassen und Interfaces

Kapitel 11: Gleich und doch ganz anders

Records, der Star unter den eigenen Datentypen

Kapitel 12: Airbags können Leben retten

Exceptionhandling

Kapitel 13: Ein ordentliches Ablagesystem muss her

Collections und Laufzeitkomplexität

Kapitel 14: Allgemein konkrete Implementierungen

Generizität

Kapitel 15: Linke Typen, auf die man sich verlassen kann

LINQ

Kapitel 16: Blumen für die Dame

Delegaten und Ereignisse

Kapitel 17: Der Standard ist nicht genug

Extension-Methoden und Lambda-Expressions

Kapitel 18: Die Magie der Attribute

Arbeiten mit Attributen

Kapitel 19: Ich muss mal raus

Dateizugriff und das Internet

Kapitel 20: Komm zurück, wenn du fertig bist

Asynchrone und parallele Programmierung

Kapitel 21: Nimm doch, was andere schon gemacht haben

Die Paketverwaltung NuGet

Kapitel 22: Die schönen Seiten des Lebens

Einführung in XAML

Kapitel 23: Models sind doch schön anzusehen

Das Model-View-ViewModel-Entwurfsmuster

Kapitel 24: Weniger ist mehr

MVVM Community Toolkit

Kapitel 25: Funktioniert das wirklich?

Unit-Testing

Kapitel 26: Schrödingers Zukunft

Vorschau auf C# 13 und .NET9

Schrödinger programmiert C#

Design & Elektronik

„HÄTTE ES DOCH SOLCHE BÜCHER VOR 20 JAHREN SCHON GEGEBEN!“

Christian Mantey, IT-Dozent

„Überraschend gut! Sehr zu empfehlen!“

Leser-Feedback:

„Das Layout der Seiten ist genial.“

Leser-Feedback:

„Jetzt bin ich platt. Ich kann nicht aufhören zu lesen.“

c’t zur Schrödinger-Reihe

„Ein neuer Weg bei der Vermittlung von Entwickler-Fachwissen.“

DAS ALLES und noch viel mehr:

- Operatoren, Schleifen, Datentypen
- Klassen und Vererbung
- Wichtige Entwurfsmuster
- GUI-Entwicklung mit XAML
- LINQ einsetzen
- async/await verwenden
- Parallele Programmierung
- Dateizugriffe und Streams
- Mobile Apps mit MAUI

Schrödinger ist unser Mann fürs Programmieren. Er kann schon was, aber noch nicht C#. Zum Glück hat er einen Kumpel, der ihm alles in Ruhe zeigt. Erst einmal die Grundlagen von C#, dann nach und nach immer mehr Magie, und am Ende sogar richtig schicke GUIs.

Rheinwerk Computing

Vom Feinsten! Die **volle Packung C#**. Die nötige Theorie, viele Hinweise und Tipps [**im Büro**], Unmengen von gutem, aber auch schlechtem Code, der verbessert und repariert werden will [**in der Werkstatt**], viele Übungen und wohlverdiente Pausen [**zuhause im Wohnzimmer**]. Mittendrin: Schrödinger. Und natürlich du.

SCHRÖDINGER GARANTIERT:

- Gründlicher **Einstieg**
- **Profi-Konzepte** von Anfang an
- Beispiele und Aufgaben mit Lösungen
- Für Einsteiger und Umsteiger **perfekt**

Ein echtes **Fachbuch**, nur eben ganz anders!

Unser Autor: Bernhard Wurm kann sich noch gut an seine Lehrjahre erinnern und bringt selbst mit, was Schrödinger jetzt braucht: Nerven, Ausdauer, Neugierde und Spaß am logischen Denken. Chapeau! So wird aus seinem Kumpel ein richtiger C#-Entwickler, den er am Ende glatt einstellen würde.

Gedruckt in Deutschland
Mineralölfreie Druckfarben
Zertifiziertes Papier

Für Windows
Programmierung/C#
ISBN 978-3-367-10623-3

€ 49,90 [D] € 51,30 [A]