

# Inhalt

Materialien zum Buch .....	18
<b>1 Über dieses Buch</b> .....	<b>19</b>
1.1 Was Sie in diesem Buch lernen werden .....	20
1.2 Was dieses Buch Ihnen zeigen möchte .....	21
1.3 Noch mehr Informationen und Guides .....	22
1.4 Danksagung .....	24
<b>2 Die Installation, die IDE und »Hallo Rust«</b> .....	<b>25</b>
2.1 Wie Sie Rust installieren .....	25
2.2 Eine Entwicklungsumgebung wählen .....	28
2.2.1 Visual Studio Code .....	28
2.2.2 JetBrains IDEs .....	30
2.3 Das erste Programm .....	30
2.3.1 Ein Paket mit »cargo« hinzufügen .....	31
2.3.2 Die Abhängigkeit einsetzen .....	32
2.4 Wie es weitergeht .....	33
<b>3 Variablen und Datentypen</b> .....	<b>35</b>
3.1 Prelude: Die Standardimporte .....	35
3.2 Variablen .....	36
3.2.1 Die Deklaration einer Variable .....	37
3.2.2 Eine Variable initialisieren .....	38
3.2.3 Eine Variable mit dem Schlüsselwort »mut« veränderlich machen ...	39
3.2.4 Die Typinferenz bei Variablen .....	43
3.2.5 Statische Variablen .....	44
3.2.6 Shadowing: Wenn sich Variablen mit gleichem Namen überdecken	50
3.2.7 Praxisbeispiel: Das Alter einlesen und verarbeiten .....	52

- 3.3 Konstanten** ..... 56
  - 3.3.1 Was sind Konstanten? ..... 56
  - 3.3.2 Konstante Kontexte ..... 59
- 3.4 Skalare Datentypen** ..... 60
  - 3.4.1 bool ..... 61
  - 3.4.2 Integer ..... 62
  - 3.4.3 Fließkommazahlen ..... 72
  - 3.4.4 Character ..... 75
  - 3.4.5 Wie Sie mit »as« den Typ wechseln ..... 78
- 3.5 Wie Rust mit »Option<T>« auf null verzichtet** ..... 81
- 3.6 Zusammenfassung** ..... 84

**4 Speichernutzung und Referenzen** ..... 87

---

- 4.1 Wichtige Speicherbereiche** ..... 87
  - 4.1.1 Der Stack ..... 88
  - 4.1.2 Der Heap ..... 88
  - 4.1.3 Statischer Speicher ..... 89
- 4.2 Eigentumsverhältnisse im Speicher** ..... 89
  - 4.2.1 Ein Wert, ein Eigentümer ..... 90
  - 4.2.2 Gültigkeitsbereiche und Blöcke beeinflussen Bindungen ..... 91
  - 4.2.3 Die Lebenszeit des Werts einer Variable ..... 94
  - 4.2.4 Bitweise Kopien mit »Copy« erzeugen ..... 95
  - 4.2.5 Move bindet den Wert an einen neuen Eigentümer ..... 96
- 4.3 Referenzen und der leihweise Zugriff** ..... 98
  - 4.3.1 Adressen, Zeiger und Referenzen: ein Überblick ..... 98
  - 4.3.2 Zugriff auf Werte über geteilte Referenzen ..... 100
  - 4.3.3 Veränderliche Referenzen: Der exklusive Zugriff ..... 105
- 4.4 Mit Box Objekte im Heap ablegen** ..... 111
  - 4.4.1 Klare Eigentumsverhältnisse auch für die Box ..... 111
  - 4.4.2 Auch Smart Pointer wie die Box können nicht alle Fehler verhindern ..... 113
  - 4.4.3 Nur die Box kann den Heap-Speicher freigeben ..... 116
  - 4.4.4 Ein Praxisbeispiel ..... 118
- 4.5 Zusammenfassung** ..... 121

---

## 5 Strings 123

---

<b>5.1</b>	<b>Der String-Slice</b> .....	123
5.1.1	String-Slices können nur hinter Referenzen auftreten .....	124
5.1.2	Von wem Sie einen String-Slice erhalten .....	126
5.1.3	Byte-String-Slices .....	128
5.1.4	Raw String-Slices .....	132
5.1.5	String-Slices zusammenfügen .....	132
<b>5.2</b>	<b>Der String</b> .....	134
5.2.1	Wie Sie einen »String« anfordern .....	134
5.2.2	Einen String in andere Datentypen parsen .....	136
5.2.3	Nützliche Bearbeitungswerkzeuge und wie man in einen eigenen Datentyp parst .....	138
<b>5.3</b>	<b>Wie Sie Strings formatieren</b> .....	147
5.3.1	Das neue Makro und Parameter .....	147
5.3.2	Die Ausgabe transformieren .....	150
5.3.3	Ausgabe-Traits .....	152
5.3.4	Makros, mit denen Sie Strings formatieren können .....	152
<b>5.4</b>	<b>Zusammenfassung</b> .....	154

---

## 6 Collections 157

---

<b>6.1</b>	<b>Tupel</b> .....	157
6.1.1	Tupel und Muster .....	158
6.1.2	Ein Tupel ist nicht gleich ein Tupel .....	162
6.1.3	Eigenschaften von Tupeln .....	163
6.1.4	Ein ganz besonderes Tupel: Der Unit-Typ .....	165
<b>6.2</b>	<b>Arrays</b> .....	166
6.2.1	Elemente aus einem Array auslesen .....	167
6.2.2	Vorsicht bei Datentypen, die nicht Copy sind .....	168
6.2.3	Die Größe mit konstanten Ausdrücken festlegen .....	171
6.2.4	Elemente in einem veränderlichen Array neu zuweisen .....	173
<b>6.3</b>	<b>Elementbereiche</b> .....	173
6.3.1	Elementbereiche haben einen Iterator .....	174
6.3.2	Einschränkungen des Iterators .....	175
6.3.3	Elementbereiche sind nicht Copy .....	177
6.3.4	Rust besitzt mehrere Elementbereich-Typen .....	177

6.3.5	Ein gemeinsamer Nenner .....	178
6.3.6	Elementbereiche als kleine Helferlein .....	181
<b>6.4</b>	<b>Vektoren</b> .....	182
6.4.1	Vektoren initialisieren .....	183
6.4.2	Elemente in einen Vektor einfügen .....	186
6.4.3	Einen Vektor mit einer anderen Collection zusammenlegen .....	187
6.4.4	Vektoren auslesen .....	190
6.4.5	Elemente aus Vektoren entfernen .....	194
6.4.6	Die Länge und Kapazität eines Vektors .....	199
6.4.7	Wenn Sie eine Queue benötigen oder auch effizient am Anfang einfügen müssen .....	204
6.4.8	Verkettete Listen .....	213
<b>6.5</b>	<b>Slices</b> .....	214
6.5.1	Einen Slice in Stücke aufteilen .....	214
6.5.2	Elemente zusammenführen oder wiederholen .....	218
6.5.3	Eigene Datentypen mit »concat« und »join« verarbeiten .....	220
6.5.4	Abfragen auf einem Slice ausführen .....	221
6.5.5	Elemente in einem Slice referenzieren .....	223
6.5.6	Veränderungen im Slice durchführen .....	224
6.5.7	Mit ASCII arbeiten .....	229
<b>6.6</b>	<b>HashMap und BTreeMap</b> .....	231
6.6.1	HashMap .....	231
6.6.2	BTreeMap .....	243
<b>6.7</b>	<b>Hashes</b> .....	245
<b>6.8</b>	<b>Mengen verwalten</b> .....	248
<b>6.9</b>	<b>Die Entry API</b> .....	251
6.9.1	VacantEntry und OccupiedEntry .....	252
6.9.2	Automatisch einen Eintrag einfügen, wenn er fehlt .....	254
6.9.3	Veränderungen on-the-fly vornehmen .....	256
<b>6.10</b>	<b>Elemente verschiedener Datentypen in eine Collection einfügen</b> .....	257
<b>6.11</b>	<b>Zusammenfassung</b> .....	260

---

## 7 Funktionen 263

---

<b>7.1</b>	<b>Der Aufbau einer Funktion</b> .....	264
7.1.1	Den Zugriff über die Sichtbarkeit steuern .....	264

---

7.1.2	Bezeichner und die Parameterliste .....	265
7.1.3	Der Rückgabewert .....	267
<b>7.2</b>	<b>Funktionszeiger</b> .....	268
<b>7.3</b>	<b>Referenzen und Lebenszeiten in Funktionen</b> .....	271
7.3.1	Warum und wann Sie Lebenszeiten angeben müssen .....	272
7.3.2	Wie Sie generische Lebenszeiten notieren .....	274
7.3.3	Varianz in Lebenszeiten .....	276
<b>7.4</b>	<b>Konstante Funktionen</b> .....	280
7.4.1	Die Ausführung ist zur Kompilierzeit und zur Laufzeit möglich .....	281
7.4.2	Seiteneffekte sind nicht erlaubt .....	282
7.4.3	Lebenszeiten in konstanten Funktionen deklarieren .....	283
7.4.4	Konditionale Ausdrücke und Schleifen .....	284
<b>7.5</b>	<b>Anonyme Funktionen und Closures</b> .....	285
7.5.1	Anonyme Funktionen .....	286
7.5.2	Was der Compiler aus einer anonymen Funktion macht .....	288
7.5.3	Anonyme Funktionen hinter Funktionszeigern .....	290
7.5.4	Parameter .....	291
7.5.5	Closures und die Umgebung einer anonymen Funktion .....	292
7.5.6	Wie die Speicherverwaltung Closures beeinflusst .....	300
<b>7.6</b>	<b>Funktions-Traits</b> .....	302
7.6.1	Die Grenzen des Funktionszeigers .....	303
7.6.2	Warum es drei verschiedene Funktions-Traits gibt .....	304
7.6.3	Die Call-Funktionen .....	310
<b>7.7</b>	<b>Zusammenfassung</b> .....	311

---

## **8 Anweisungen, Ausdrücke und Muster** 313

---

<b>8.1</b>	<b>Von der Anweisung zum Ausdruck und Muster</b> .....	313
8.1.1	Die Item-Anweisung .....	314
8.1.2	Die Ausdruck-Anweisung .....	314
<b>8.2</b>	<b>Die Zuweisung im Detail</b> .....	316
8.2.1	Wertausdrücke: Die rechte Seite der let-Anweisung .....	316
8.2.2	Beispiel match: Ein komplexer Wertausdruck .....	317
8.2.3	Seiteneffekte, aber kein Rückgabewert .....	318
<b>8.3</b>	<b>Speicherausdrücke</b> .....	319
8.3.1	Der Speicherausdruck auf der linken Seite der Zuweisung .....	320

8.3.2	Der Speicherausdruck im Wertausdruck .....	321
8.3.3	Der Wertausdruck im Speicherausdruck .....	323
<b>8.4</b>	<b>Operatoren</b> .....	325
8.4.1	Arithmetische Operatoren und Vergleichsoperatoren .....	325
8.4.2	Mit dem Gruppen-Ausdruck in die Auswertungsreihenfolge eingreifen .....	326
8.4.3	Der Zuweisungsoperator .....	327
8.4.4	Operatoren auf die eigenen Datentypen anwenden .....	328
8.4.5	Übersicht zum Vorrang aller Ausdrücke und Operatoren in Rust .....	329
<b>8.5</b>	<b>Konditionale Ausdrücke</b> .....	330
8.5.1	Der if-Ausdruck .....	330
8.5.2	Der if let-Ausdruck .....	332
8.5.3	Der match-Ausdruck .....	333
<b>8.6</b>	<b>Schleifen</b> .....	342
8.6.1	Der Label-Block .....	342
8.6.2	Bis zur Unendlichkeit mit loop .....	343
8.6.3	while und while let .....	346
8.6.4	Die for in-Schleife .....	348
<b>8.7</b>	<b>Muster</b> .....	350
8.7.1	Muster in der let-Anweisung .....	350
8.7.2	Einfache Muster .....	352
8.7.3	Der Bindungsoperator .....	359
8.7.4	Die Widerlegbarkeit von Mustern .....	360
<b>8.8</b>	<b>Zusammenfassung</b> .....	364

## 9 Fehlerbehandlung 367

---

<b>9.1</b>	<b>Fehler, von denen sich das Programm nicht erholen kann</b> .....	367
9.1.1	Eine Panic tritt pro Thread auf .....	368
9.1.2	Wenn Sie Panics für möglich halten, isolieren Sie sie vom Hauptthread .....	371
9.1.3	Mit »abort« vermeiden, dass der Stack abgewickelt wird .....	372
9.1.4	Panics abfangen und behandeln .....	373
<b>9.2</b>	<b>Erwartbare Fehler behandeln</b> .....	381
9.2.1	Result: Ein Typ, zwei Wege .....	382
9.2.2	Die Werte in Result verarbeiten und verwenden .....	386
9.2.3	Der Fragezeichen-Operator .....	397

9.2.4	Das Trait Error .....	399
9.2.5	Option<T>: Behälter und Fehlertyp .....	407
<b>9.3</b>	<b>Zusammenfassung .....</b>	<b>418</b>

## **10 Strukturen** 421

---

<b>10.1</b>	<b>Daten zusammenhängend ablegen .....</b>	<b>422</b>
<b>10.2</b>	<b>Records: Der Struktur-Urtyp .....</b>	<b>423</b>
<b>10.3</b>	<b>Strukturen und Instanzen .....</b>	<b>426</b>
10.3.1	Keine Konstruktoren in Rust .....	427
10.3.2	Kurzschreibweise und Update-Syntax .....	428
10.3.3	Der Zugriff auf Felder und die Veränderlichkeit .....	430
10.3.4	Die automatische Dereferenzierung in der Struktur .....	433
10.3.5	Datenkapselung in Rust .....	434
<b>10.4</b>	<b>Lebenszeiten: Wenn Felder Referenzen enthalten .....</b>	<b>441</b>
10.4.1	Statische und nicht statische Referenzen .....	441
10.4.2	Generische Lebenszeiten in der Struktur .....	443
10.4.3	Vermeiden Sie zu viele generische Lebenszeiten in Strukturen .....	446
10.4.4	Generische Lebenszeiten, die nicht begrenzt werden .....	448
<b>10.5</b>	<b>Wie Sie dem Compiler mit PhantomData wichtige Typinformationen übergeben .....</b>	<b>449</b>
10.5.1	Die Speichersicherheit mit »PhantomData« erweitern .....	454
10.5.2	Einen generischen Datentyp mit PhantomData einsetzen .....	456
<b>10.6</b>	<b>Eine Datenstruktur ohne feste Größe .....</b>	<b>460</b>
<b>10.7</b>	<b>Die drei Strukturen .....</b>	<b>462</b>
10.7.1	Die Tupel-Struktur .....	462
10.7.2	Das Newtype-Muster .....	463
10.7.3	Unit-Typ-ähnliche Strukturen .....	465
<b>10.8</b>	<b>Muster .....</b>	<b>466</b>
<b>10.9</b>	<b>Daten und Verhalten sind getrennt .....</b>	<b>468</b>
10.9.1	Der inhärente Implementierungsblock .....	468
10.9.2	Generische Lebenszeiten im Implementierungsblock .....	469
10.9.3	Eine generische Lebenszeit im Implementierungsblock einführen ....	472
10.9.4	Wenn der implementierende Typ eine generische Lebenszeit aufweist .....	474
10.9.5	Anonyme Lebenszeiten .....	475

<b>10.10 Strukturen in der Praxis: Das Bestellsystem überarbeiten</b>	475
10.10.1 Die Anforderungen und der Entwurf des Systems	475
10.10.2 Projektaufbau und erste Datenstrukturen	477
<b>10.11 Assoziierte Funktionen und die new-Konvention</b>	480
10.11.1 Öffnungszeiten im Restaurant	481
10.11.2 Die Sichtbarkeit erweitert sich nicht	482
10.11.3 Strukturen mit »new« initialisieren	484
<b>10.12 Methoden</b>	486
10.12.1 Die Methode und der self-Parameter	486
10.12.2 self ist eine Kurzform	488
10.12.3 Mehrere Parameter	490
10.12.4 »&mut self« und andere Seiteneffekte in Methoden	494
10.12.5 »self« und »mut self«: Die eigene Instanz verbrauchen	498
<b>10.13 Referenzen in assoziierten Funktionen und Methoden</b>	501
<b>10.14 Praxisbeispiel: Simulationsfähigkeiten im Restaurant-System</b>	503
10.14.1 Hunger und Durst	503
10.14.2 Die Wahl-Funktion in »BestellungBuilder« anschließen	505
<b>10.15 Rekursion in Strukturen</b>	507
<b>10.16 Typ-Aliasie</b>	510
<b>10.17 Zusammenfassung</b>	512

---

## 11 Traits 515

---

<b>11.1 Marker-Traits</b>	516
<b>11.2 Trait-Implementierungsblöcke</b>	517
11.2.1 Die Orphan-Regel	518
11.2.2 Elemente, die Sie mit Traits assoziieren können	521
11.2.3 Sichtbarkeiten	538
<b>11.3 Sie können ein Trait jeweils für T und &amp;T implementieren</b>	541
11.3.1 Self kann T oder &T sein	542
11.3.2 Lebenszeiten von Referenzen in der Trait-Implementierung	543
11.3.3 Mehrere Lebenszeiten im Trait-Implementierungsblock	545
<b>11.4 Super-Traits</b>	546
<b>11.5 Trait-Objekte</b>	549
11.5.1 Wie ein Trait-Objekt entsteht	552
11.5.2 Wie eine vTable aussieht	552



---

11.5.3	Ein Trait-Objekt ist eine Einbahnstraße .....	555
11.5.4	Der Sized-Marker .....	556
11.5.5	Nicht jedes Trait kann zu einem Trait-Objekt werden .....	559
11.5.6	Objektsicherheit: Was ist das? .....	559
11.5.7	Ein Trait für ein Trait-Objekt implementieren .....	561
11.5.8	Der inhärente Implementierungsblock eines Trait-Objekts .....	563
<b>11.6</b>	<b>Beispielprojekt: Trait-Objekte von »Form« .....</b>	<b>564</b>
11.6.1	Die Komponenten .....	565
11.6.2	Das Koordinatenfeld zeichnen .....	566
11.6.3	Die Implementierungen von »Form« für Punkt und Linie .....	569
11.6.4	Das Rechteck: Mehrere Linien berechnen .....	571
11.6.5	Der Kreis .....	572
11.6.6	Trait-Objekte an das Koordinatenfeld übergeben .....	573
<b>11.7</b>	<b>Undurchsichtige Datentypen zurückgeben .....</b>	<b>574</b>
11.7.1	Abstrakte Rückgabetypen .....	575
11.7.2	Anonyme Typparameter .....	576
<b>11.8</b>	<b>Traits in der Praxis .....</b>	<b>578</b>
11.8.1	Copy und Clone .....	579
11.8.2	Any und Typeld .....	585
11.8.3	»drop« – der Rust-Destruktor .....	595
11.8.4	Default: ein Standardkonstruktor à la Trait .....	602
11.8.5	Borrow<T> und BorrowMut<T> .....	605
11.8.6	AsRef<T> und AsMut<T> .....	613
11.8.7	Typkonvertierungen mit From<T> .....	615
11.8.8	Into<T> ist das Gegenstück zu From<T> .....	622
11.8.9	From<T> oder Into<T>: Wann Sie welches Trait implementieren sollten .....	623
11.8.10	TryFrom<T> und TryInto<T> .....	625
<b>11.9</b>	<b>Zusammenfassung .....</b>	<b>627</b>

---

## 12 Enumerationen 631

---

<b>12.1</b>	<b>Die Eigenschaften einer Enumeration .....</b>	<b>632</b>
12.1.1	Eine Enumeration ohne Varianten .....	633
12.1.2	Implizite und explizite Diskriminanten .....	635
12.1.3	Explizite Diskriminanten sind konstante Werte .....	638
12.1.4	Instanzen einer Enumeration .....	639

<b>12.2</b>	<b>Verschiedene Variant-Typen</b>	644
12.2.1	Tupel-Varianten	644
12.2.2	Struktur-Varianten	647
12.2.3	Varianten als Datentypen	648
12.2.4	Speicherbedarf	653
<b>12.3</b>	<b>Enumerationen und Muster</b>	656
<b>12.4</b>	<b>Implementierungsblöcke und Verhalten</b>	660
12.4.1	»Gewicht« und der inhärente Implementierungsblock	660
12.4.2	Das Trait »Add« für »Gewicht«	662
12.4.3	Das Trait »Display« für »Gewicht«	665
<b>12.5</b>	<b>Zusammenfassung</b>	667

## **13 Module, Pfade und Crates** 669

---

<b>13.1</b>	<b>Das Modul</b>	669
13.1.1	Das implizite Modul	670
13.1.2	Ein explizites Modul definieren	671
13.1.3	Namensraum und Sichtbarkeit im Modul	672
13.1.4	Der Modulbaum	678
13.1.5	Denken Sie in Modulen, nicht in Dateien oder Verzeichnissen	687
<b>13.2</b>	<b>Pfade</b>	697
13.2.1	Die Anatomie eines Pfads	697
13.2.2	Welche Elemente bekommen einen Pfad?	699
13.2.3	Wohin ein Pfad führen kann	701
13.2.4	Wie Sie einen Pfad »umbiegen«	710
13.2.5	Die »use«-Deklaration	712
<b>13.3</b>	<b>Vom Crate zum Paket, vom Paket zum Workspace</b>	721
13.3.1	Am Anfang war das Crate	721
13.3.2	Das kleinste Crate ist eine Rust-Datei	722
13.3.3	Das Paket	723
13.3.4	Paketabhängigkeiten hinzufügen: Ein Paket kommt selten allein	731
13.3.5	Paketabhängigkeiten konfigurieren	734
13.3.6	»dev-dependencies«: Abhängigkeiten nur für die Entwicklung	735
13.3.7	»Build-Skripte« und die »build-dependencies«	736
13.3.8	Paketversionen anfordern	737
13.3.9	Attribute	742
13.3.10	Konditionale Kompilierung	753

13.3.11 Features .....	762
13.3.12 Workspaces .....	772
<b>13.4 Zusammenfassung .....</b>	<b>777</b>

---

## **14 Generische Programmierung** 781

---

<b>14.1 Von der Vorlage zur Konkretisierung: Monomorphisierung .....</b>	<b>781</b>
<b>14.2 Typparameter, generische Konstanten und Lebenszeiten .....</b>	<b>783</b>
<b>14.3 Syntaktische Elemente, die generisch sein können .....</b>	<b>785</b>
14.3.1 Implementierungsblöcke reichen Typparameter weiter .....	785
14.3.2 Der assoziierte Datentyp eines Datentyps ist generisch .....	786
14.3.3 Generische und assoziierte Datentypen verbinden .....	787
14.3.4 Assoziierte Datentypen und Trait-Grenzen .....	788
<b>14.4 Mehr zu Trait-Grenzen .....</b>	<b>789</b>
14.4.1 Trait-Grenzen kombinieren .....	789
14.4.2 Anonyme Typparameter: »impl Trait« .....	791
14.4.3 Trait Grenzen mit »where« .....	791
14.4.4 Blanket-Implementierungen .....	792
<b>14.5 Zusammenfassung .....</b>	<b>794</b>

---

## **15 Iteratoren** 797

---

<b>15.1 Wie Sie einen Iterator beziehen .....</b>	<b>798</b>
15.1.1 Das Trait »Intoliterator« .....	799
15.1.2 »Iterator« implementieren .....	800
<b>15.2 Iterator-Adapter .....</b>	<b>805</b>
15.2.1 Eigentum im Iterator .....	807
15.2.2 Iteratoren zusammenfügen .....	812
15.2.3 Transformationen in der Iterator-Kette .....	813
15.2.4 Weitere Iterator-Adapter .....	816
<b>15.3 Einen Iterator konsumieren .....</b>	<b>816</b>
15.3.1 Eine Zusammenstellung einfacher Konsumenten .....	817
15.3.2 Finden, falten und reduzieren .....	817
15.3.3 Sonstige Methoden, die einen Iterator konsumieren .....	820
<b>15.4 Zusammenfassung .....</b>	<b>822</b>

**16 Nebenläufige und asynchrone Programmierung** 823

---

<b>16.1 Nebenläufige Programmierung</b> .....	824
16.1.1 Threads .....	826
16.1.2 »Send« und »Sync«: sicherer nebenläufiger Code .....	838
16.1.3 Channels: Die Kommunikationsinfrastruktur zwischen Threads .....	851
16.1.4 Synchronisierung in Konkurrenz: Der wechselseitige Ausschluss .....	858
16.1.5 Einen oder mehrere Threads per Signal aufwecken mit »Condvar« ...	864
16.1.6 RwLock: Mehrere Leser oder ein exklusiver Zugriff .....	868
16.1.7 Atomare Datentypen und Operationen .....	869
<b>16.2 Smart Pointer</b> .....	876
16.2.1 »Cow«: Referenz und Klonanleitung zugleich .....	877
16.2.2 Cells und die Interior Mutability .....	880
16.2.3 Mit Referenzzählern zum geteilten Eigentum .....	888
<b>16.3 Asynchrone Programmierung</b> .....	893
16.3.1 Der Unterschied zwischen Thread und Task .....	894
16.3.2 »async« und »await« .....	895
16.3.3 Die asynchrone Laufzeitumgebung .....	897
16.3.4 Asynchrone Funktionen, Parallelität und Join .....	899
16.3.5 Die Future ist ein Zustandsautomat .....	902
<b>16.4 Zusammenfassung</b> .....	915

**17 Makros** 917

---

<b>17.1 Deklarative Makros</b> .....	917
17.1.1 Warum Makros? .....	918
17.1.2 Ein Beispiel-Makro .....	920
17.1.3 Die Meta-Syntax .....	921
17.1.4 Der Gültigkeitsbereich .....	938
17.1.5 Makro-Hygiene .....	939
<b>17.2 Prozedurale Makros</b> .....	939
17.2.1 Für prozedurale Makros müssen Sie eine Bibliothek anlegen .....	941
17.2.2 Die drei prozeduralen Makro-Arten .....	942
<b>17.3 Zusammenfassung</b> .....	950

---

## 18 Automatische Tests und Dokumentation 953

---

<b>18.1 Tests</b>	954
18.1.1 Das Untermodul »tests«	954
18.1.2 Testfunktionen, Result<T, E> und der Fragezeichen-Operator	956
18.1.3 Die Attribute »ignore« und »should_panic«	957
18.1.4 Teststrukturierung	960
18.1.5 Dynamische und statische »asserts«	963
18.1.6 Integrationstests	965
<b>18.2 Rust-Projekte dokumentieren</b>	966
18.2.1 Die Dokumentation erzeugen	967
18.2.2 Wie Sie Ihr Projekt dokumentieren	970
18.2.3 »Doc-Tests«: Codebeispiele in Kommentaren	975
<b>18.3 Zusammenfassung</b>	979

---

## 19 Unsafe Rust und das Foreign Function Interface 981

---

<b>19.1 Unsafe Rust</b>	981
19.1.1 »unsafe« in Blöcken und Funktionen	982
19.1.2 Unsichere Traits und Trait-Implementierungen	985
19.1.3 Statisch-globale Variablen verändern	986
<b>19.2 Primitive Zeiger</b>	987
19.2.1 Wie Sie gültige Zeiger erhalten	987
19.2.2 Null-Zeiger	989
19.2.3 Operationen	990
19.2.4 Der Fallstrick Move	996
<b>19.3 Union</b>	998
<b>19.4 Foreign Function Interface</b>	1001
19.4.1 Von Rust zu C oder C++	1002
19.4.2 Von C oder C++ zu Rust	1003
<b>19.5 Zusammenfassung</b>	1005
<b>Index</b>	1007