"30 "31 "32 "33 "3B "3C "3D "01 "02 "10 "03 "04 "11 "3E "3F "1D "18 "23 "16 "72 "73 "74 "75 "76

Informatik mit dem Tablet unterrichten

Programmiersprache swift mit dem iPad lernen

Michael Dienert

9. Juni 2024

Inhaltsverzeichnis

1 Swift Playgrounds **1**
1.1 Swift . . . 1
1.2 Playgrounds . . . 1

2 Swift Syntax **1**
2.1 Funktionen . . . 1
2.2 Wiederholungen: Zählschleife . . . 2
2.3 Ergänzungen zu Zählschleifen . . . 3
2.4 Ergänzungen zu Funktionen . . . 3
2.5 Bedingte Ausführung von Anweisungen . . . 3
2.6 Boole'sche Ausdrücke . . . 4
2.7 Verknüpfung Boole'scher Ausdrücke . . . 4
2.8 Verknüpfung Boole'scher Ausdrücke . . . 5
2.9 Verknüpfung Boole'scher Ausdrücke . . . 5
2.10 while-Schleifen . . . 5
2.11 Ein Labyrith-Algorithmus . . . 6
2.12 Ein Labyrith-Algorithmus: Lösung . . . 6

3 Typen sind Objekte **6**
3.1 Objekte . . . 6
3.2 Deklaration von Variablen . . . 8
3.3 Deklaration von Variablen . . . 8
3.4 Deklaration von Listen . . . 9

4 swift on Linux **9**
4.1 swift ohne Apple-HW . . . 9
4.2 Quellcode PrimSieb . . . 10
4.3 Quellcode PrimSieb, Fortsetzung . . . 10

1 Swift Playgrounds

1.1 Swift

- Swift ist eine seit 2014 von Apple entwickelte Programiersprache für allgemeine Anwendungen.
- das Wichtigste (sonst wäre ich nicht hier): Swift ist *quelloffen* (Open Source): der Quellcode ist *frei zugänglich* und kann den Bestimmungen der Lizenz entsprechend (Apache License 2.0) verwendet werden.
- Swift wurde als Nachfolgesprache von C / Objective-C / C++ entworfen. Urspüngliche Anwendung war die Programmierung von Anwendungen für Apple-Geräte.
- Swift ist objektorientiert und funktional (Funktionen können wie Variablen verwendet werden)
- Da Swift quelloffen ist, existieren Compiler für verschiedene Plattformen (Apple, Windows, Linux)

1.2 Playgrounds

- *Swift Playgrounds* ist eine Anwendung (aka *App*) für iPad und Mac-Rechner.
- Zielgruppe von Swift Playgrounds sind **Kinder**! Aber: *Es ist nie zu spät für eine glückliche Kindheit*.
- *Swift Playgrounds* installieren und starten
- Mit *Programmieren lernen 1* starten. Wer es eilig hat, kann auch *Los geht's mit Code* ausführen, die Inhalte decken sich. *Programmieren lernen* ist etwas ausführlicher.

2 Swift Syntax

2.1 Funktionen

- Um die Spielfigur zu bewegen und die Aufgaben des ersten Kapitels zu lösen, muss man die Funktionen
  - moveForward()
  - turnLeft()
  - collectGem()
  - toggleSwitch()

  in bestimmter Reihenfolge aufrufen.

- Eine Funktion erkennt man immer daran, dass ein *Bezeichner* direkt von einer sich öffnenden *runden Klammer* gefolgt wird. Bei Funktionen die keine Parameter übergeben bekommen, wird die Klammer direkt wieder geschlossen.

- Es soll die Funktione `turnRight()` selbst erstellt werden.

- Allgemeine Syntax für das Erstellen einer Funktion ohne Parameter und ohne Rückgabewert:

```
func machWas(){
  anweisung1
  anweisung2
  usw.
}
```

  - **`func`** ist ein Schlüsselwort, muss zwingend da stehen.
  - **`machWas()`** ist der Name unserer Funktion, kann frei gewählt werden. Die **`()`**-Klammern sind zwingend.
  - die **`{`**-Klammer startet die Gruppe von Anweisungen, die ausgeführt werden sollen.
  - die **`}`**-Klammer schliesst die Anweisungsgruppe und beendet den Code der Funktion.

2.2 Wiederholungen: Zählschleife

- In den Übungen im zweiten Kapitel geht es um Funktionen.

- Wer schon etwas Programmiererfahrung hat, kann sich die Tipparbeit durch den Einsatz von Schleifen sehr erleichtern.

- Wer in der Reihenfolge bleiben möchte: Schleifen werden erst im dritten Kapitel eingefuehrt.

- Schleifensyntax:

```
for i in 0...3{
  anweisung1
  anweisung2
  usw.
}
```

2.3 Ergänzungen zu Zählschleifen

- Wie oft wird die Zählschleife durchlaufen?
- Von der unteren bis einschliesslich der oberen Grenze:

```
for i in 1...3{
    print(i)
}
1
2
3
```

- Von der unteren bis ausschliesslich der oberen Grenze:

```
for i in 1..<3{
    print(i)
}
1
2
```

2.4 Ergänzungen zu Funktionen

- Man kann Funktionen auch mit Parametern aufrufen.
- In der Spielewelt muss man manchmal eine bestimmte Anzahl Schritte gehen.
- Das kann man mit der Übergabe der Schrittanzahl an eine Funktion lösen:

```
// definition der funktion
func schritte(anzahl:Int){
  for i in 1 ... anzahl{
    moveForward()
  }
}

// aufruf im code:
schritte(anzahl:3)
```

- Das Schlüsselwort `Int` steht für den Datentyp. `Int` bedeutet *ganze Zahl* (integer).

2.5 Bedingte Ausführung von Anweisungen

- Mit dem Schlüsselwort `if` lässt sich erreichen, dass ein Block von Anweisungen nur ausgeführt wird, wenn eine vorgegebene Bedingung **wahr** ist.
- Syntax:

```
if bedingung {
  anweisung1
  anweisung2
  usw.
}
```

- `if-else`-Bedingung:

```
if bedingung {
  anweisung
   ...
} else {
  anweisung
   ...
 }
```

2.6 Boole'sche Ausdrücke

- George Boole, englischer Mathematiker, 1815-1864.
- Ein Boole'scher-Ausdruck ist ein Ausdruck, der entweder den Wert `true` oder `false` annimmt.
- Der Vergleichsoperator für die Identität ist ==
- Beispiele:

```
let a=1
if a==1 {
  //code wird ausgefuehrt a==1 liefert true
}
if a>5{
  //code wird nicht ausgefuehrt a>5 liefert false
}
```

2.7 Verknüpfung Boole'scher Ausdrücke

- Boole'sche Ausdrücke lassen sich mit *logischen Operatoren* verknüpfen.
- NICHT-Operator: der NICHT-Operator (Negation, engl. `not`) ändert den Wert des Boole'schen Ausdruck in sein Gegenteil. Er ist ein dem Boole'schen Ausdruck *vorangestelltes* Ausrufezeichen.

```
if !bedingung {
  anweisung
```

```
  ...
}
```

2.8 Verknüpfung Boole'scher Ausdrücke

- Die logische ODER-Verknüpfung ergibt nur dann einen wahren Wert wenn `mindestens einer` der Teilausdrücke wahr ist. Der logisch-ODER-Operator sind die beiden Zeichen: `||`

```
if bedingung1 || bedingung2 {
  anweisung
   ...
}
```

2.9 Verknüpfung Boole'scher Ausdrücke

- Die logische UND-Verknüpfung ergibt nur dann einen wahren Wert wenn `beide` Teilausdrücke wahr sind. Der logisch-UND-Operator sind die beiden Zeichen: `&&`

```
if bedingung1 && bedingung2 {
  anweisung
   ...
}
```

2.10 while-Schleifen

- In einer `while`-Schleife wird ein Anweisungsblock so lange wiederholt, solange eine Bedingung *wahr* ist.
- Z.B. solange, bis alle 7 Edelsteine eingesammtelt sind:

```
var zaehler=0
while zaehler<7 {
  if isOnGem{
    collectGem()
    zaehler=zaehler+1
  }

  //block wird so oft wiederholt, bis
  //alle 7 Edelsteine gesammelt wurden
  anweisungen
  ...

}
```

2.11 Ein Labyrith-Algorithmus

- Die Aufgaben in der Spielewelt sind oft ein bisschen mühsam.
- Viel mehr Spass würde es machen, wenn die Spielefigur automatisch alles einsammelt.
- Mit folgender, einfacher Regel kommt man durch jedes Labyrinth (nicht unbedingt schnell): immer mit der rechten Hand am Hindernis zur Rechten bleiben, wenn geradeaus blockiert ist, nach links abbiegen.
- Voraussetzung ist, dass man links von einer Wand stehend beginnt.

2.12 Ein Labyrith-Algorithmus: Lösung

```
var zaehler=0
while zaehler<7 {
  if isOnGem{
    collectGem()
    zaehler=zaehler+1
  }

  if isOnClosedSwitch{
    toggleSwitch()
  }

  if !isBlockedRight{
    turnRight()
  }

  if isBlocked{
    turnLeft()
  }else{
    moveForward()
  }

}
```

3 Typen sind Objekte

3.1 Objekte

- swift ist eine *Objektorientierte Programmiersprache*.
- Objekte sind *Datenstrukturen* im Speicher des Rechners.
- Bei swift heissen die Objekte ***Instanzen***, die Vorschrift, wie so eine Instanz aussieht heisst in swift ***Typ***.
- Den Speicher stellt man sich dabei als einen sehr hohen Schubladenschrank vor.
- Die Datenstrukturen entsprechen den einzelnen Schubladen:

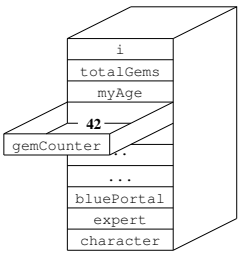

Abbildung 1: Speichermodell

- Die Schubladen können in ihrem Inneren beliebig komplex aufgebaut werden.
- Es gibt ganz einfache Schubladen, in denen z.b. nur eine einzelne, ganze Zahl liegt.
- Auf den Inhalt einer einfachen Schublade greift man direkt über ihren Namen zu:

```
while gemCounter < 42{
  //codeblock
}
```

- Bei komplexen Schubladen mit einer weiteren Unterteilung im Inneren benötigt man die **Punktnotation**: `bluePortal.isActive`

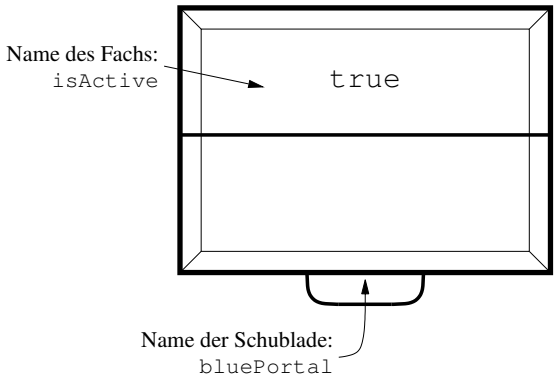

Abbildung 2: Schublade mit Punktnotation

- Komplexe Schubladen dürfen nicht nur Werte enthalten, sondern man kann auch *Taster* einbauen, die bestimmte Aktionen auslösen.
- Mit dem Drücken eines Tasters wird der Aufruf einer *Funktion* ausgelöst.
- Im Umfeld der Objektorientierung werden Funktionen auch als *Methoden* eines Objekts bezeichnet.
- Entsprechend werden die Werte in den Schubladenfächern als *Eigenschaftenx* bezeichnet.

- Auf die Taster wird auch mit der Punktnotation zugegriffen.

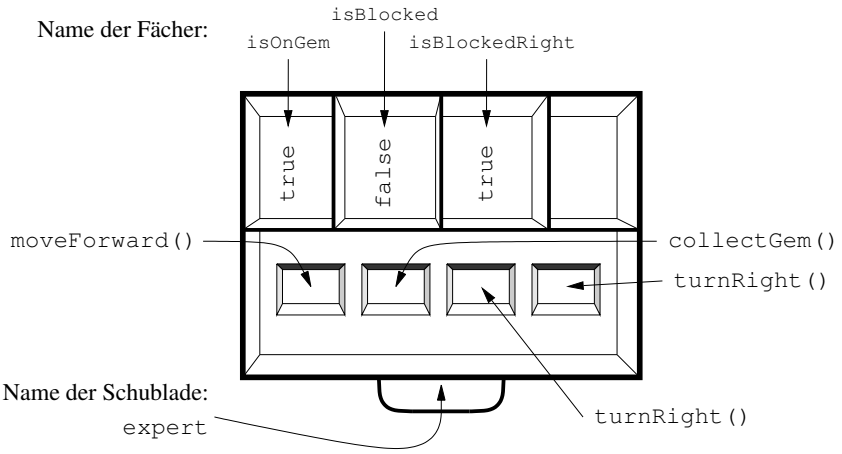

Abbildung 3: Modell eines Objekts mit Eigenschaften und Methoden

3.2 Deklaration von Variablen

- Die Schubladen = Variablen im Speicher brauchen Platz.
- Damit der Compiler weiss, wieviel Platz reserviert werden muss, müssen Variablen vor der Verwendung deklariert werden.
- Bisher haben wir das *implizit deklariert*:

```
var i = 0
var x = 3.1415
```

- Der swift-Compiler bestimmt den Typ der Variablen *implizit* aus dem übergebenen Zahlenwert.
- Explizit sieht das so aus:

```
var i: Int
var x: Double
i = 1
x = 3.1415
var j:Int = 10;  //einzeilig explizit
```

3.3 Deklaration von Variablen

- Komplexe Schubladen = Objekte werden so deklariert:

```
var expert1: Expert      //explizite deklaration
expert1 = Expert()       //''bauen'' der schublade

var expert2 = Expert()   //implizite deklaration und
                         //erzeugen der datenstruktur

if expert2.isBlocked {   //auf eigenschaft zugreifen
   expert2.jump()        //methode aufrufen
}
```

3.4 Deklaration von Listen

- Listen werden in der Datenstruktur *Array* gespeichert.
- Ein Array kann man sich wie eine Schublade vorstellen, die intern gleichmaessig in Fächer ***gleichen Typs*** eingeteilt ist.
- Ein Array hat immer einige vordefinierte Methoden und Eigenschaften

```
var listig1 = [2,3,1024,117] //implizit deklariert
var listig2: [Int] = [4,2,1] //explizit deklariert
var wert = listig2[0]        //zugriff auf 0-tes element

listig2.append(0)            //wert hinten anhaengen
listig2.insert(3, at: 1)     //einfuegen: 4,3,2,1,0
listig2.remove(at: 0)        //loeschen:    3,2,1,0
for element in listig2{
  print(element)
}
```

4 swift on Linux

4.1 swift ohne Apple-HW

- swift lässt sich unter Linux und anderen OS problemlos verwenden.
- auf XCode muss man leider verzichten.
- Installation unter *Debian11/Ubuntu20.04* (Links aktukell im Mai 2022):

```
cd /usr/local
sudo wget https://download.swift.org/swift-5.6.1-release/ubuntu2004/\
     swift-5.6.1-RELEASE/swift-5.6.1-RELEASE-ubuntu20.04.tar.gz
sudo tar -xzf swift-5.6.1-RELEASE-ubuntu20.04.tar.gz
sudo ln -s swift-5.6.1-RELEASE-ubuntu20.04
export PATH=/usr/local/swift/usr/bin
```

- Projekt anlegen:

```
mkdir primSieb
cd primSieb
swift package init --type executable
swift build  #quellen uebersetzen
swift run    #binaries ausfuehren
.build/debug/primSieb 12 #binary direkt mit parameter (hier: 12) starten
```

4.2 Quellcode PrimSieb

```
//leeres feld anlegen
var zahlen: [Bool] = []

func sieb(kante: Int){

    let max = kante*kante

// hier nicht ueber i sondern _ iterieren, da die zaehlvariable nie verwendet wird
    for _ in 1 ... max{
        zahlen.append(true)
    }

    zahlen[0] = false
    zahlen[1] = false

    for i in 2 ..< zahlen.count{
        if zahlen[i]{
            for j in stride(from: i*i, to: zahlen.count, by: i){
                zahlen[j] = false
            }
        }
    }
}//func sieb
```

4.3 Quellcode PrimSieb, Fortsetzung

```
func ausgeben(kante: Int){
    for (index,value) in zahlen.enumerated(){
        if value {
            print(index, terminator:"\t")
        }else{
            print("*", terminator:"\t")
        }
        if index % kante == 0{
            print()
        }
    }
    print()
}

let arguments = CommandLine.arguments

print(arguments[1])
if arguments.count != 2 {
    print("Aufruf: primSieb <seitenlaenge des feldes>")
}else{
    let kante = Int(arguments[1]) ?? 0
    sieb(kante: kante)
    ausgeben(kante: kante)
}
```