Grundkurs Programmieren

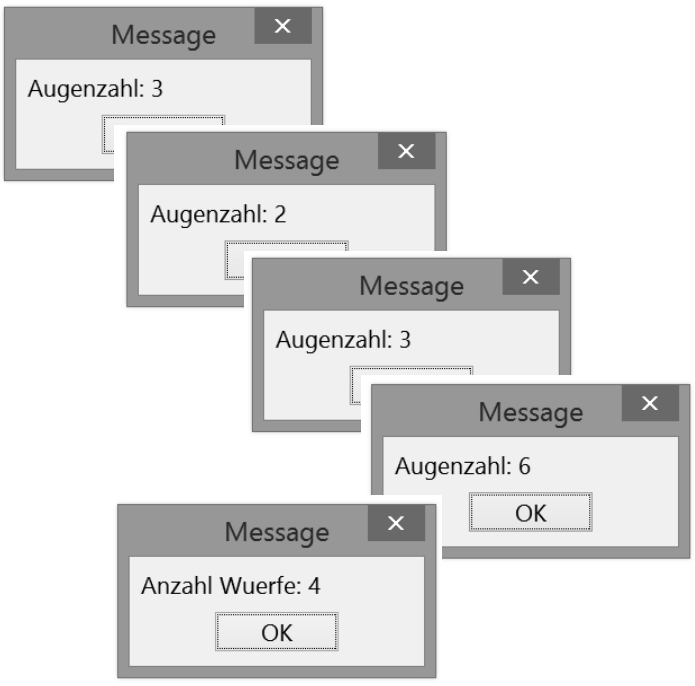
Für den Einsatz im Mathematikunterricht unter Verwendung von Python mit dem Fokus auf Zufallsvorgänge

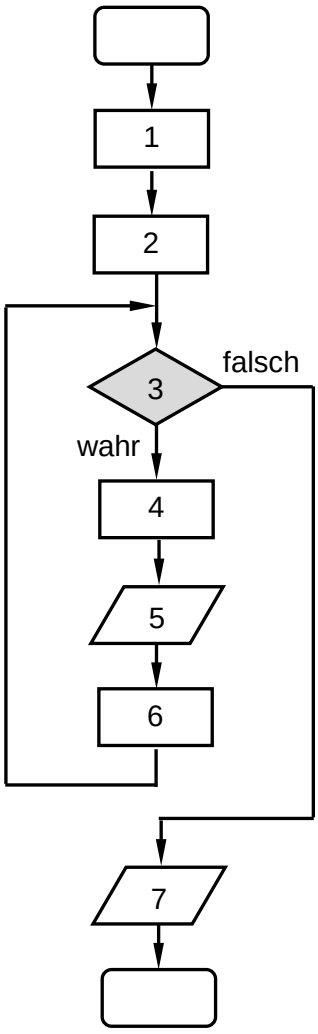
Moritz Adelmeyer / Dezember 2016

Einleitung und Kapitel 3

Bezug der kompletten Unterlage als PDF auf Anfrage per Mail

```
    from random import *
1   a = 0
2   w = 0
3   while a != 6 :
4       a = randint(1,6)
5       msgDlg("Augenzahl:", a)
6       w = w + 1
7   msgDlg("Anzahl Wuerfe:", w)
```





Zielesetzung und Zielgruppe

Dieser Kurs hat zwei Ziele: Einerseits will er das **algorithmische Denken** fördern, andererseits bietet er eine Einführung in die **strukturierte Programmierung** anhand der Sprache Python.

Der Kurs wurde für den Einsatz im **Grundlagenfach Mathematik** an Schweizer Gymnasien entwickelt. Er will **allen Gymnasiastinnen und Gymnasiasten**, insbesondere auch denen, die kein mathematisch-naturwissenschaftliches Profil gewählt haben, ein **Grundrüstzeug im Programmieren** mitgeben. Ein solches gehört heute sowohl zur **Allgemeinbildung** und als auch zur **Hochschulvorbereitung**.

Kapiteleinteilung

Konzeption

Das Motto dieses Kurses lautet: **Mit möglichst wenig möglichst viel erreichen**!

Dementsprechend werden in den grundlegenden Kapiteln 2 bis 4 nur **wenige ausgewählte Programmierelemente** eingeführt: **Anweisungen** zur Verarbeitung von **Zahlen** sowie je eine Art von **Verzweigung** und **Schleife**. Dafür wird umso mehr darauf geachtet, diese Elemente vielfältig zu verwenden, etwa indem mehrere Verzweigungen und Schleifen **aufeinander folgend** oder **ineinander geschachtelt** angeordnet werden.

Im weiterführenden Kapitel 6 werden zusätzlich **Listen** eingeführt, wobei sich der Kurs auch hier auf einen Aspekt beschränkt: den Zugriff auf die einzelnen Listenelemente über den Index.

Kapitelaufbau

Die einzelnen Kapitel bestehen im Wesentlichen aus **Beispielen** und **Aufgaben**. Anhand der Beispiele werden neuen Elemente eingeführt, erläutert und illustriert. Die Aufgaben knüpfen an den Beispielen an und greifen dann weitere zum Kapitel passende Problemstellungen auf. Am Ende jedes Kapitels sind **Lösungen** zu allen Aufgaben abgedruckt.

Die einzelnen Kapitel sind als **A4-Broschüren** konzipiert. Öfters gehören die linke und rechte Seite inhaltlich zusammen.

Zufallsvorgänge

Ein Grossteil der Beispiele und Aufgaben in diesem Kurs handelt von **Zufallsvorgängen**. Das hat mehrere Gründe.

Zum ersten ist die Simulation von Zufallsprozessen auf dem Computer in vielen Gebieten von Bedeutung. Das gilt für die Ingenieur- und Naturwissenschaften ebenso wie die Finanz-, Wirtschafts- und Sozialwissenschaften.

Zum zweiten wird mit den Zufallsvorgängen eine Brücke zur **Wahrscheinlichkeitsrechnung** geschlagen. Die Wahrscheinlichkeitsrechnung mit ihrem **theoretischen Zugang** und der Programmierkurs mit seinem **experimentellen Zugang** ergänzen sich ideal.

Zum dritten sind Zufallsvorgänge in der Regel einfach beschreibbar und intuitiv erfassbar. Dementsprechend können die Schülerinnen und Schüler ihre Aufmerksamkeit ganz auf die Programmierung richten.

Durch den Fokus auf Zufallsvorgänge erhält der Programmierkurs eine **Verortung im Mathematikunterricht**. Es ist dabei einerlei, ob die Wahrscheinlichkeitsrechnung vor oder nach dem Programmieren behandelt wird.

Vorstellungshilfen

Der Kurs verwendet **Flussdiagramme** und **Werteprotokolle**, um den Aufbau von Programmen grafisch darzustellen und deren Ablauf Schritt für Schritt tabellarisch festzuhalten. Flussdiagramme und Werteprotokolle dienen in diesem Kurs als **zentrale Vorstellungshilfen** zum Verständnisaufbau.

Zeile	a	w	$a \neq 6$
1	0		
2		0	
3			wahr
4	3		
6		1	
3			wahr
4	2		
6		2	
3			wahr
4	3		
6		3	
3			wahr
4	6		
6		4	
3			falsch

Programmiersprache und Programmierumgebung

Der Fokus dieses Kurses liegt auf dem **Programmieren**. Die verwendete Sprache ist zweitrangig. **Python** wurde hautsächlich darum gewählt, weil ihr Programmcode gegenüber vergleichbaren Sprachen wie etwa Java besonders einfach und übersichtlich gehalten ist. Ein weiterer Grund ist die Nähe von Python zur Mathematiksoftware **Sage** (vgl. letzte Seite).

Als Programmierumgebung wird **TigerJython** verwendet (www.tigerjython.ch). TigerJython wurde für die Schule entwickelt, ist dementsprechend einfach gehalten, gratis verfügbar und kann als Java-Applikation ohne Installationsaufwand plattformübergreifend benutzt werden.

Unterrichtseinsatz

Dieser Kurs ermöglicht verschiedene Arten der Unterrichtsgestaltung. Er gibt lediglich Programmierelemente, Beispiele und Aufgaben vor. Starke Schülerinnen und Schüler können die Kapitel im Selbststudium durcharbeiten, schwächere brauchen den Klassenunterricht und die Unterstützung der Lehrperson.

Für die grundlegenden Kapitel 1 bis 5 sind ca. 24 Lektionen zu veranschlagen, für die weiterführenden Kapitel 6 und 7 noch einmal ca. 10 Lektionen. Dazu kommen Hausaufgaben.

Vorbild

Der Kurs ist inspiriert durch folgendes Lehrbuch:

Robert Sedgewick, Kevin Wayne, Robert Dondero:
Introduction to Programming in Python – An Interdisciplinary Approach.
Addison-Wesley, 2015.

Das Buch hat vier Teile: Elements of Programming, Functions and Modules, Object-Oriented Programming und Algorithms and Data Structures. In gekürzter und vereinfachter Form versehen mit eigenen Beispielen und Aufgaben folgt dieser Kurs dem ersten Teil des Buchs.

Sage

Sage ist eine frei verfügbare Mathematiksoftware, die als Alternative zu kommerziellen Programmen wie Maple oder Mathematica entwickelt wurde (www.sagemath.org).

Sage ist eng mit Python verknüpft. Grosse Teile der Software von Sage sind in Python geschrieben. Die Python-Module NumPy und SymPy werden für numerische bzw. symbolische Berechnungen verwendet. Zudem kann in Sage mit der Syntax von Python programmiert werden.

Eine einfache Online-Arbeitsumgebung für Sage wird unter sagecell.sagemath.org zur Verfügung gestellt. Sie kann aus allen gängigen Webbrowsern ohne Installation benutzt werden.

Für den Mathematikunterricht an Gymnasien ist Sage eine Alternative zu grafik- und algebrafähigen Taschenrechnern bzw. eine Ergänzung zu einfachen Taschenrechnern und bildet zusammen mit Python ein leistungsstarkes Duo.

Kontakt

Moritz Adelmeyer

Kantonale Maturitätsschule für Erwachsene / Fachschaft Mathematik
Mühlebachstrasse 112 / CH-8008 Zürich

moritz.adelmeyer@kme.ch

Grundkurs Programmieren 3: Verzweigungen

M. Adelmeyer / Dezember 2016

Überblick

Die Programme im letzten Kapitel bestanden aus Anweisungen, die eine nach der anderen in der immer gleichen Reihenfolge ausgeführt wurden. Beim Zahlenratespiel High-Low aus dem vorletzten Kapitel war dies anders: Das Programm stellte nach einem Rateversuch des Spielers fest, ob er zu hoch, zu tief oder richtig lag und gab je nach Fall eine unterschiedliche Meldung aus.

Solche **Verzweigungen** sind ein wichtiges Element von Algorithmen bzw. Programmen. In diesem dritten Kapitel lernst du den grundlegendsten Typ von Verzweigung kennen und verwenden.

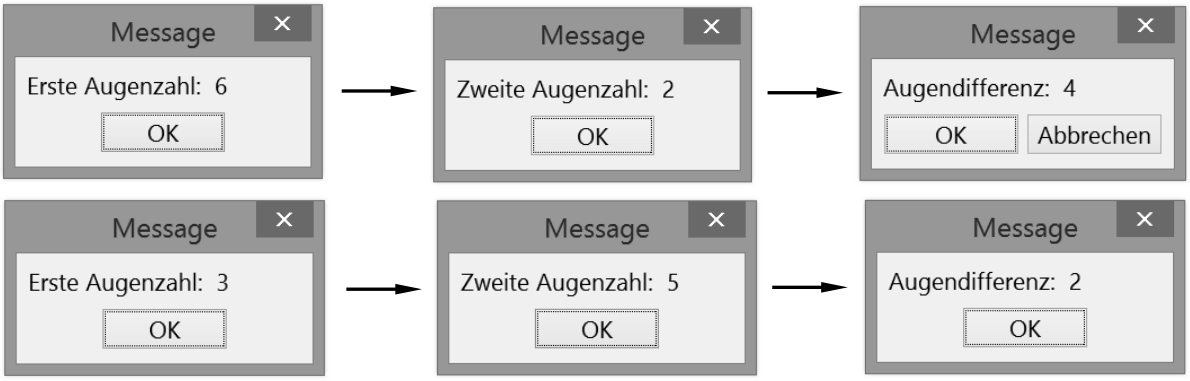
Am Anfang einer Verzweigung steht eine **Bedingung**. Sie entscheidet darüber, welche der zur Auswahl stehenden Anweisungen als nächstes ausgeführt werden. Du lernst in diesem Kapitel, wie man Bedingungen formuliert und wie Bedingungen miteinander verknüpft werden können.

Mit Verzweigungen nimmt die Komplexität der Programme deutlich zu. Im **Aufgabenteil** wirst du auf Programme treffen, die zwei oder gar drei Verzweigungen enthalten. Manche der Verzweigungen sind aufeinander folgend angeordnet, manche ineinander verschachtelt. Flussdiagramme und Werteprotokolle werden dir helfen, den Durchblick zu bewahren.

Beispiel 31

Problemstellung (Augendifferenz bei zweimaligem Würfeln)

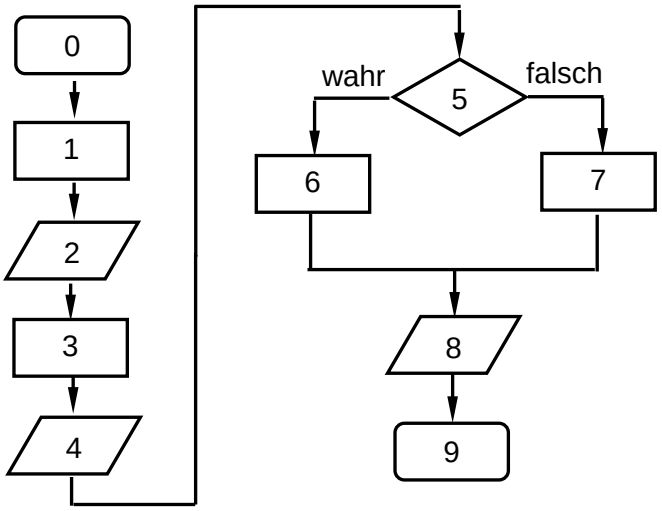
Der Computer würfelt zweimal. Nach jedem Wurf zeigt er die geworfene Augenzahl an. Zum Schluss bildet der Computer die Differenz zwischen der grösseren und der kleineren geworfenen Augenzahl und zeigt diese an.



Das Neue

Es gibt **zwei unterschiedliche Fälle**: Wenn die erste Augenzahl grösser als die zweite ist, muss die erste Augenzahl minus der zweite gerechnet werden. Ansonsten muss umgekehrt die zweite Augenzahl minus die erste gerechnet werden.

Flussdiagramm



0 Beginn

1 Zum ersten Mal würfeln
2 Erste geworfene Augenzahl anzeigen

3 Zum zweiten Mal würfeln
4 Zweite geworfene Augenzahl anzeigen

5 Prüfen, ob die erste Augenzahl grösser oder gleich der zweiten Augenzahl ist
6 Differenz zwischen erster und zweiter Augenzahl bilden
7 Differenz zwischen zweiter und erster Augenzahl bilden

8 Augendifferenz anzeigen

9 Ende

Programmcode

```
    from random import *
1   a1 = randint(1,6)
2   msgDlg("Erste Augenzahl:", a1)
3   a2 = randint(1,6)
4   msgDlg("Zweite Augenzahl:", a2)
5   if a1 >= a2 :
6       d = a1 – a2
    else :
7       d = a2 – a1
8   msgDlg("Augendifferenz:", d)
```

Erläuterungen

- In Zeile 5 tritt die **Bedingung** `a1 >= a2` auf. Es wird dabei geprüft, ob der Wert der Variablen `a1` grösser oder gleich dem Wert der Variablen `a2` ist. Das Ergebnis der Überprüfung lautet `True` oder `False` , also **wahr** oder **falsch**.
- Im Flussdiagramm werden Bedingungen durch **Rauten** dargestellt ().
- Für Zahlen gibt es folgende Vergleichsbedingungen:

`x == y`	prüft, ob die Werte von x und y **gleich** sind.
`x != y`	prüft, ob die Werte von x und y **ungleich** sind.
`x > y`	prüft, ob der Wert von x **grösser** als der Wert von y ist.
`x < y`	prüft, ob der Wert von x **kleiner** als der Wert von y ist.
`x >= y`	prüft, ob der Wert von x **grösser oder gleich** dem Wert von y ist.
`x <= y`	prüft, ob der Wert von x **kleiner oder gleich** dem Wert von y ist.

- Die Zeilen 5 bis 7 sind ein Beispiel für eine **Verzweigung**:

– In Zeile 5 wird eine Bedingung geprüft.
– Wenn die Bedingung wahr ist, wird Zeile 6 ausgeführt.
– Ansonsten, das heisst wenn die Bedingung falsch ist, wird Zeile 7 ausgeführt.

- Eine Verzweigung ist wie folgt aufgebaut:

```
if bedingung :
    anweisung
    anweisung
    .........
else :
    anweisung
    anweisung
    .........
```

`if` – Block wird ausgeführt, wenn `bedingung` **wahr** ist.

`else` – Block wird ausgeführt, wenn `bedingung` **falsch** ist.

Beachte:

– Nach **`if`** `bedingung` und nach **`else`** muss jeweils ein **Doppelpunkt `:`** stehen.
– Die Anweisungen müssen eine gleichmässige **Einrückung** aufweisen. Typischerweise sind es vier Leerzeichen.

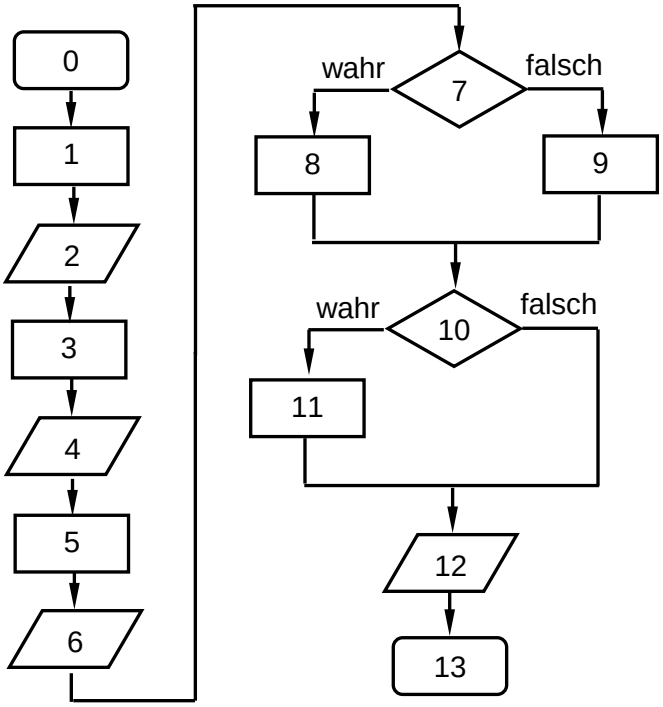
- Durch die gleichmässige Einrückung wird in der Sprache Python festgelegt, welche Anweisungen einen gemeinsamen **Block** bilden. Bei Verzweigungen gibt es den **`if`** – Block und den **`else`** – Block

Beispiel 32

Problemstellung (Grösste Augenzahl beim dreimaligen Würfeln)

Der Computer würfelt dreimal. Nach jedem Wurf zeigt er die geworfene Augenzahl an. Zum Schluss ermittelt der Computer die grösste geworfene Augenzahl und zeigt diese an.

Flussdiagramm



0 Beginn

1 Zum ersten Mal würfeln
2 Erste geworfene Augenzahl anzeigen
3 Zum zweiten Mal würfeln
4 Zweite geworfene Augenzahl anzeigen
5 Zum dritten Mal würfeln
6 Dritte geworfene Augenzahl anzeigen

7 Prüfen, ob die erste Augenzahl grösser als die zweite Augenzahl ist
8 Erste Augenzahl als bisher grösste Augenzahl speichern
9 Zweite Augenzahl als bisher grösste Augenzahl speichern

10 Prüfen, ob die dritte Augenzahl grösser als die bisher grösste Augenzahl ist
11 Dritte Augenzahl als neue grösste Augenzahl speichern

12 Grösste Augenzahl anzeigen

13 Ende

Programmcode

```
    from random import *

1   a1 = randint(1,6)
2   msgDlg("Erste Augenzahl:", a1)
3   a2 = randint(1,6)
4   msgDlg("Zweite Augenzahl:", a2)
5   a3 = randint(1,6)
6   msgDlg("Dritte Augenzahl:", a3)

7   if a1 > a2 :
8       g = a1
    else :
9       g = a2

10  if a3 > g :
11      g = a3

12  msgDlg("Groesste Augenzahl:", g)
```

Werteprotokolle

Zeile	a1	a2	a3	g	a1 > a2	a3 > g
1	2					
3		3				
5			5			
7					falsch	
9				3		
10						wahr
11				5		

Zeile	a1	a2	a3	g	a1 > a2	a3 > g
1	4					
3		1				
5			4			
7					wahr	
8				4		
10						falsch

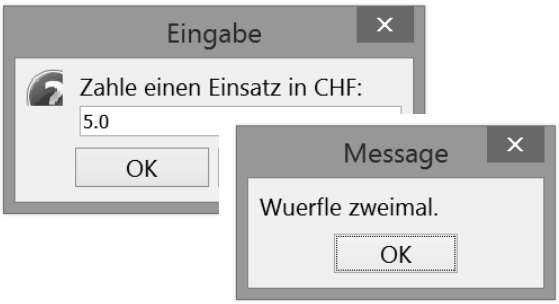
Erläuterungen

- Das Programm enthält zwei **aufeinander folgende** Verzweigungen auf.
- Bei der zweiten Verzweigung enthält der **`else`** – Block keine Anweisungen. Er kann dann im Programmcode weggelassen werden.

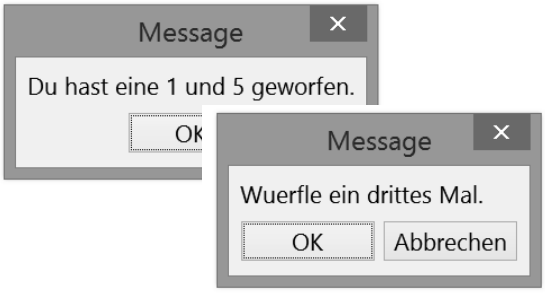
Beispiel 33

Problemstellung (Ein Würfelspiel)

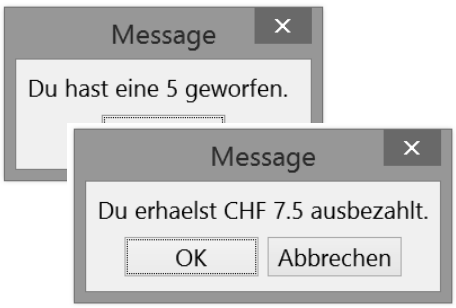
Der Spieler zahlt zunächst einen Einsatz. Dann würfelt er zweimal.

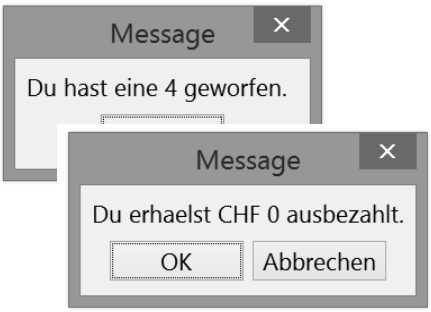


Wenn die beiden Augenzahl gleich sind, erhält er den dreifachen Einsatz ausbezahlt. Andernfalls würfelt der Spieler ein drittes Mal.



Wenn die dritte Augenzahl gleich der ersten oder zweiten ist, erhält der Spieler den eineinhalbfachen Einsatz ausbezahlt. Andernfalls erhält er keine Auszahlung.

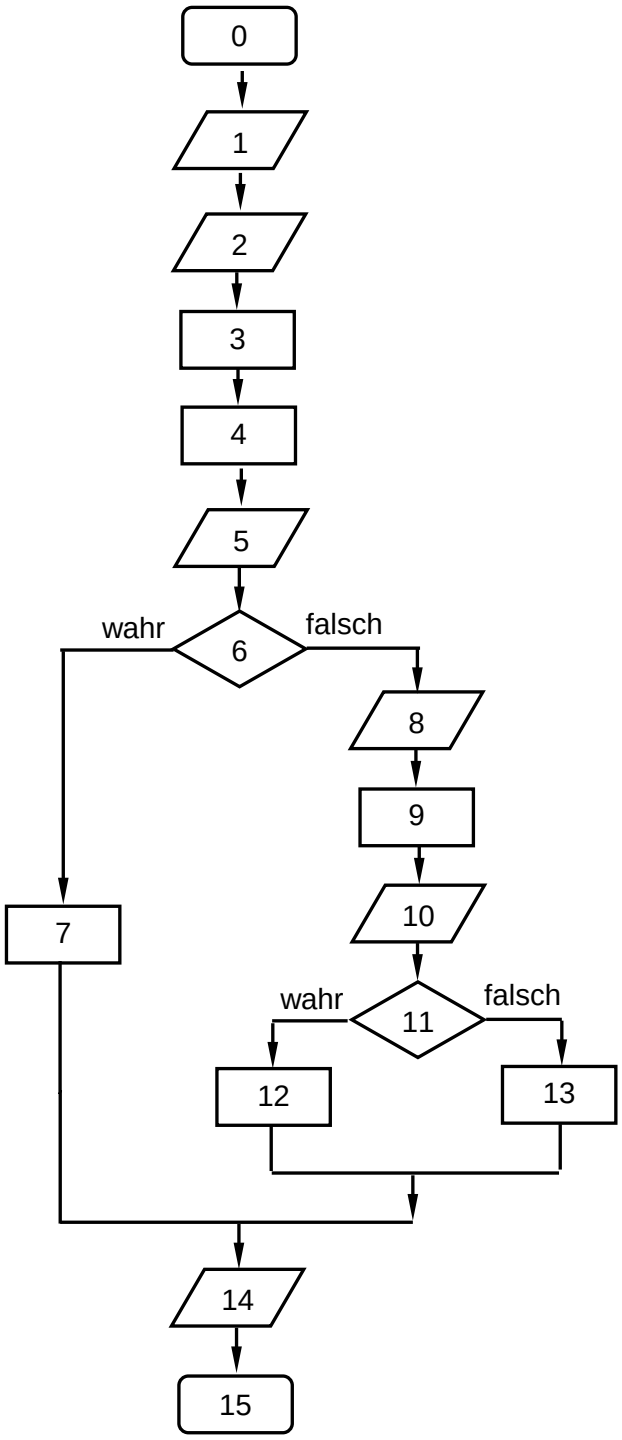




Programmcode

```
    from random import *

1   e = inputFloat("Zahle einen Einsatz in CHF:")

2   msgDlg("Wuerfle zweimal.")
3   w1 = randint(1,6)
4   w2 = randint(1,6)
5   msgDlg("Du hast eine", w1, "und", w2, "geworfen.")

6   if w1 == w2 :
7       a = 3*e
    else :
8       msgDlg("Wuerfle ein drittes Mal.")
9       w3 = randint(1,6)
10      msgDlg("Du hast eine", w3, "geworfen.")

11      if w3 == w1 or w3 == w2 :
12          a = 1.5*e
        else :
13          a = 0

14  msgDlg("Du erhaelst CHF", a, "ausbezahlt.")
```

Flussdiagramm



0 Beginn

1 Spieler zur Eingabe des Einsatzes auffordern und Einsatz speichern

2 Spieler zum zweimaligen Würfeln auffordern
3 Ein erstes Mal würfeln und Augenzahl speichern
4 Ein zweites Mal würfeln und Augenzahl speichern
5 Erste und zweite Augenzahl anzeigen

6 Prüfen, ob erste und zweite Augenzahl gleich sind
7 Auszahlung gleich dreifachem Einsatz setzen

8 Spieler zum einem dritten Wurf auffordern
9 Ein drittes Mal würfeln und Augenzahl speichern
10 Dritte Augenzahl anzeigen

11 Prüfen, ob die dritte Augenzahl gleich der ersten oder zweiten ist
12 Auszahlung gleich eineinhalbfachem Einsatz setzen
13 Auszahlung gleich null setzen

14 Auszahlung anzeigen

15 Ende

Erläuterungen

- Das Programm enthält zwei **ineinander verschachtelte** Verzweigungen:

– Die **äussere** Verzweigung hat Zeile 6 als Bedingung. Zeile 7 bildet den **`if`** – Block und Zeilen 8 bis 13 den **`else`** – Block der äusseren Verzweigung.
– Die **innere** Verzweigung hat Zeile 11 als Bedingung. Zeile 12 bildet den **`if`** – Block und Zeile 13 den **`else`** – Block der inneren Verzweigung.

  Beachte, dass entsprechend der Verschachtelung die Zeilen 12 und 13 **zweifach eingerückt** sein müssen.

- Bedingungen können mit **`and`** bzw. **`or`** , also mit **und** bzw. **oder** verknüpft werden:

– `bedingung1` **`and`** `bedingung2` **`and`** `...` ist wahr, wenn **alle** Bedingungen wahr sind.
– `bedingung1` **`or`** `bedingung2` **`or`** `...` ist wahr, wenn **mindestens eine** Bedingung wahr ist.

Aufgaben

Aufgabe 31 (Beispiel 31 / eine Verzweigung)

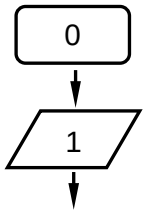
a) Tippe das Programm aus Beispiel 31 ein und führe es aus.

b) Ändere das Programm aus Beispiel 31 so ab, dass nicht die Augendifferenz berechnet und angezeigt wird, sondern dass festgestellt und angezeigt wird, ob die beiden geworfenen Augenzahlen gleich oder verschieden sind.

Aufgabe 32 (Kugel ziehen ohne Zurücklegen / eine Verzweigung)

- In eine Urne werden zunächst w weisse Kugeln und s schwarze Kugeln gelegt. Insgesamt liegen dann n = w + s Kugeln in der Urne. Die weissen Kugeln werden mit 1 bis w nummeriert, die schwarzen mit w+1 bis n.
- Dann wird eine Kugel zufällig gezogen und nicht mehr zurückgelegt. Es wird nacheinander angezeigt, welche Nummer die gezogene Kugel hat, ob diese weiss oder schwarz ist sowie wie viele weisse und schwarze Kugeln nach dem Zug noch in der Urne sind.

```
   from random import *

1  w = inputInt("Anzahl weisse Kugeln:")
2  s = inputInt("Anzahl schwarze Kugeln:")
3  n = w + s

4  k = randint(1,n)
5  msgDlg("Nummer der gezogenen Kugel:", k)

6  if  k <= w :
7      msgDlg("Es wurde eine weisse Kugel gezogen.")
8      w = w – 1
   else :
9      msgDlg("Es wurde eine schwarze Kugel gezogen.")
10     s = s – 1

11 n = n – 1

12 msgDlg("Es sind noch", n, "Kugeln in der Urne,", w, "weisse und",
   s, "schwarze.")
```



usw.

a) Zeichne das zum Programm gehörige Flussdiagramm.

b) Das linke Werteprotokoll ist vollständig ausgefüllt, das rechte erste teilweise. Fülle es ganz aus.

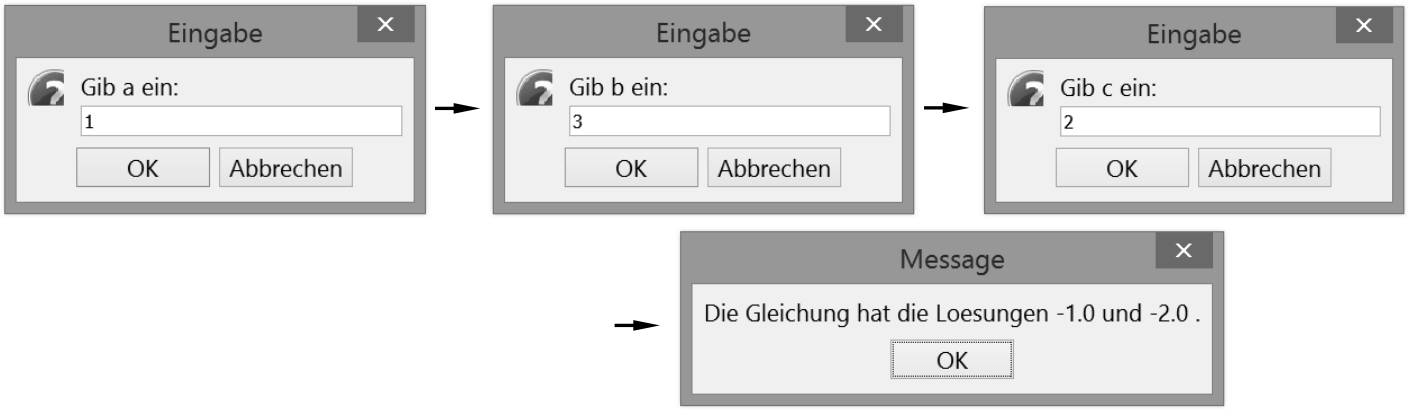
Zeile	w	s	n	k	k ≤ w
1	4				
2		6			
3			10		
4				2	
6					wahr
8	3				
11			9		

Zeile	w	s	n	k	k ≤ w
1	6				
2		10			
3					
4				12	

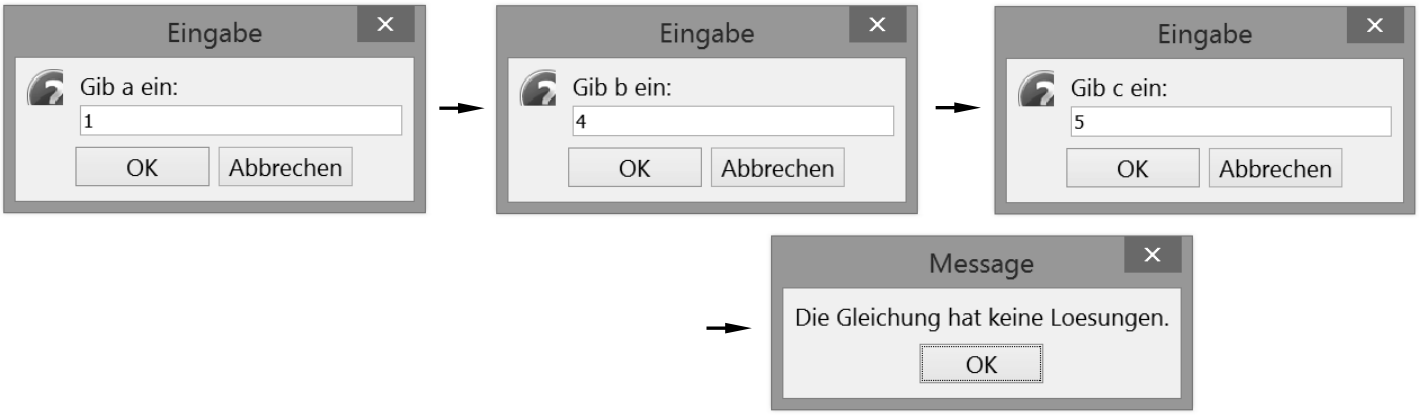
c) Tippe das Programm ein und führe es aus.

Aufgabe 33 (Quadratische Gleichungen / eine Verzweigung)

In einer Aufgabe von Kapitel 2 hast du ein Programm geschrieben, dass eine quadratische Gleichung der Form $a \cdot x^2 + b \cdot x + c = 0$ löst. Der Benutzer gibt a, b und c ein, der Computer berechnet die beiden Lösungen und zeigt sie an:



Erweitere das Programm aus Kapitel 2 so, dass es prüft, ob die quadratische Gleichung Lösungen besitzt und wenn nicht dies angibt:



Aufgabe 34 (Beispiel 32 / zwei bzw. drei aufeinanderfolgende Verzweigungen)

a) Tippe das Programm aus Beispiel 32 ein und führe es aus.

b) Ändere das Programm aus Beispiel 32 so ab, dass nicht dreimal sondern viermal eine Augenzahl geworfen wird und dass nicht die grösste sondern die kleinste geworfene Augenzahl ermittelt und angezeigt wird.

Aufgabe 35 (Notenbilanz / drei aufeinander folgende Verzweigungen)

```
1   n1 = inputFloat("Gib erste Note ein:")
2   n2 = inputFloat("Gib zweite Note ein:")
3   d1 = n1 - 4
4   d2 = n2 – 4
5   if  d1 < 0 :
6       d1 = 2*d1
7   if  d2 < 0 :
8       d2 = 2*d2
9   s = d1 + d2
10  if  s >= 0 :
11      msgDlg("Bestanden mit", s, "Pluspunkten.")
    else :
12      s = -s
13      msgDlg("Nicht bestanden mit", s, "Minuspunkten.")
```

a) Zeichne das zum Programm gehörige Flussdiagramm.

b) Wenn die Noten 5.5 und 3.5 eingegeben werden, so sieht das Werteprotokoll wie folgt aus:

Zeile	n1	n2	d1	d2	s	d1 < 0	d2 < 0	s ≥ 0
1	5.5							
2		3.5						
3			1.5					
4				–0.5				
5						falsch		
7							wahr	
8				–1				
9					0.5			
10								wahr

Erstelle das Werteprotokoll, wenn die Noten 3.25 und 5 eingegeben werden:

Zeile	n1	n2	d1	d2	s	d1 < 0	d2 < 0	s ≥ 0

c) Tippe das Programm ein und führe es aus.

Aufgabe 36 (Beispiel 33 / zwei ineinander verschachtelte Verzweigungen)

Tippe das Programm aus Beispiel 33 ein und führe es aus.

Aufgabe 37 (Münzen werfen / zwei ineinander verschachtelte Verzweigungen)

Mit `randint(0,1)` wird zufällig entweder 0 oder 1 erzeugt. Dies kann man als Ergebnis eines **Münzwurfs** interpretieren, wobei **0** gleich **Kopf** und **1** gleich **Zahl** entspricht.

Im folgenden Programm werden zwei Münzen geworfen und es wird festgestellt, ob beide geworfenen Münzen Kopf, beide Münzen Zahl oder je eine Münze Kopf und Zahl zeigen.

```
   from random import *
1  msgDlg("Werfe zwei Muenzen.")
2  m1 = randint(0,1)
3  m2 = randint(0,1)
4  if  m1 != m2 :
5      msgDlg("Du hast einmal Kopf und einmal Zahl geworfen.")
   else :
6      if m1 == 0 :
7          msgDlg("Du hast zweimal Kopf geworfen.")
       else :
8          msgDlg("Du hast zweimal Zahl geworfen.")
```

a) Zeichne das zum Programm gehörige Flussdiagramm.

b) Tippe das Programm ein und führe es aus.

Aufgabe 38 (Würfelspiel / drei ineinander verschachtelte Verzweigungen)

0 Beginn

1 Spieler zur Zahlung eines Einsatzes von CHF 2 auffordern

2 Spieler zum einmaligen Würfeln auffordern
3 Ein erstes Mal würfeln und Augenzahl speichern
4 Erste Augenzahl anzeigen

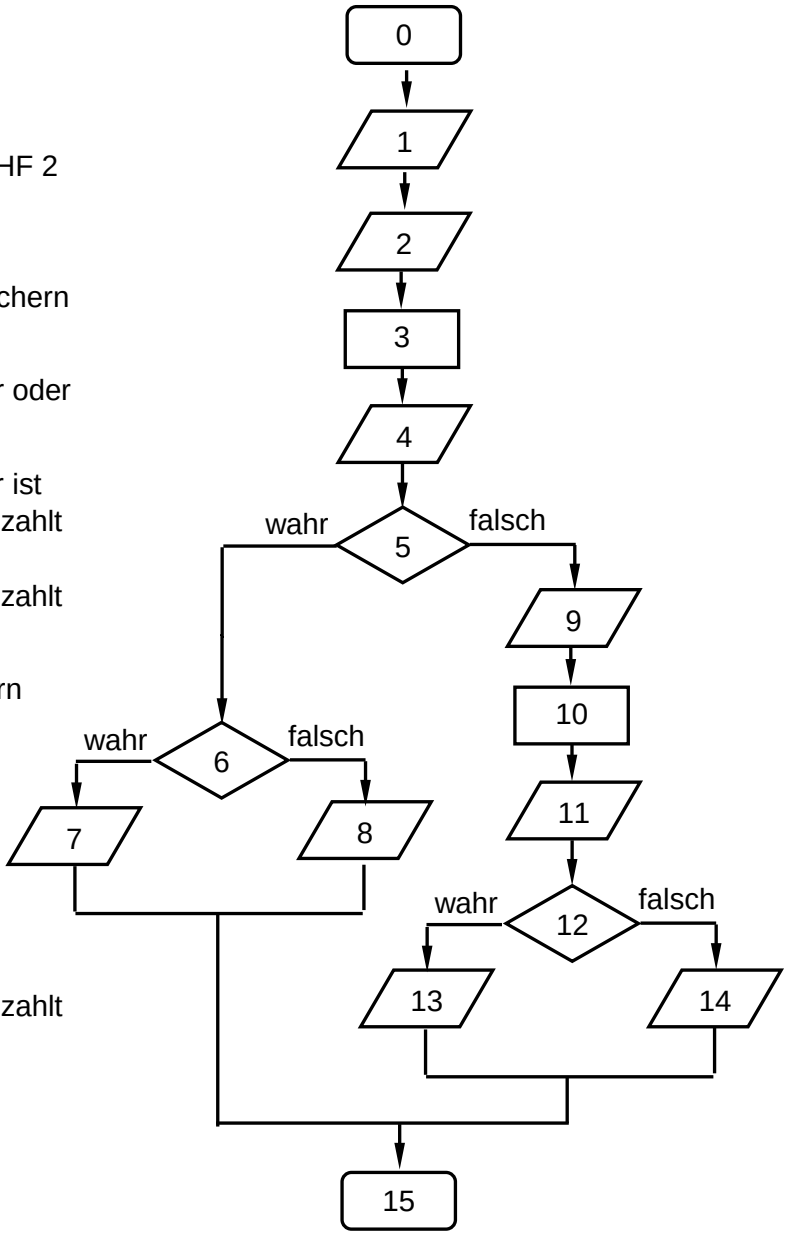
5 Prüfen, ob erste Augenzahl grösser als vier oder kleiner als drei ist

6 Prüfen, ob erste Augenzahl grösser als vier ist
7 Dem Spieler melden, dass er CHF 5 ausbezahlt bekommt.
8 Dem Spieler melden, dass er CHF 0 ausbezahlt bekommt.

9 Spieler zum nochmaligen Würfeln auffordern
10 Ein zweites Mal würfeln und Augenzahl speichern
11 Zweite Augenzahl anzeigen

12 Prüfen, ob zweite Augenzahl grösser als vier ist.
13 Dem Spieler melden, dass er CHF 3 ausbezahlt bekommt.
14 Dem Spieler melden, dass er CHF 0 ausbezahlt bekommt.

15 Ende



a) Führe das Spiel einige Male durch.
b) Schreibe das zum Flussdiagramm gehörige Programm.
c) Tippe dein Programm ein und führe es aus.

LÖSUNGEN

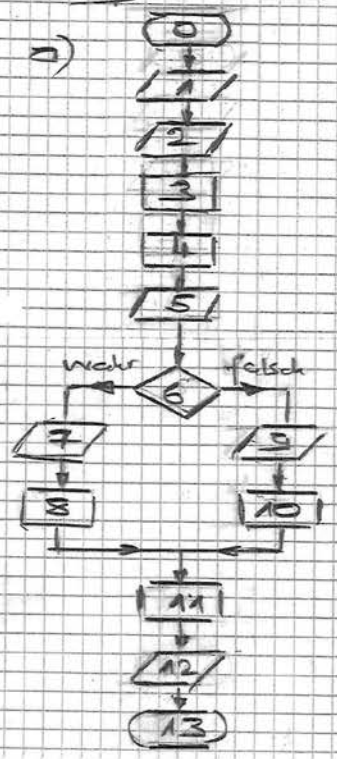
Aufgabe 31

b)

```
from random import *

a1 = randint(1,6)
msgDlg("Erste Augenzahl: ", a1)

a2 = randint(1,6)
msgDlg("Zweite Augenzahl: ", a2)

if a1 == a2 :
    msgDlg("Die Augenzahlen sind gleich.")
else :
    msgDlg("Die Augenzahlen sind verschieden.")
```

Aufgabe 32

a)



b)

Zeile	w	s	n	k	k ≤ w
1	6				
2		10			
3			16		
4				12	
6					falsch
10		5			
11			15		

Aufgabe 33

```
from math import *

msgDlg("Dieses Programm loest quadratische Gleichungen",
"vom Typ a*x^2 + b*x + c = 0.")

a = inputFloat("Gib a ein:")
b = inputFloat("Gib b ein:")
c = inputFloat("Gib c ein:")

d = b*b - 4*a*c

# d ist die sog. Determinante der quadratischen Gleichung:
# Fuer d >= 0 besitzt die Gleichung Loesungen,
# fuer d < 0 nicht.

if d >= 0 :
    x1 = (-b + sqrt(d))/(2*a)
    x2 = (-b - sqrt(d))/(2*a)
    msgDlg("Die Gleichung hat die Loesungen", x1, "und", x2,".")
else :
    msgDlg("Die Gleichung hat keine Loesungen.")
```

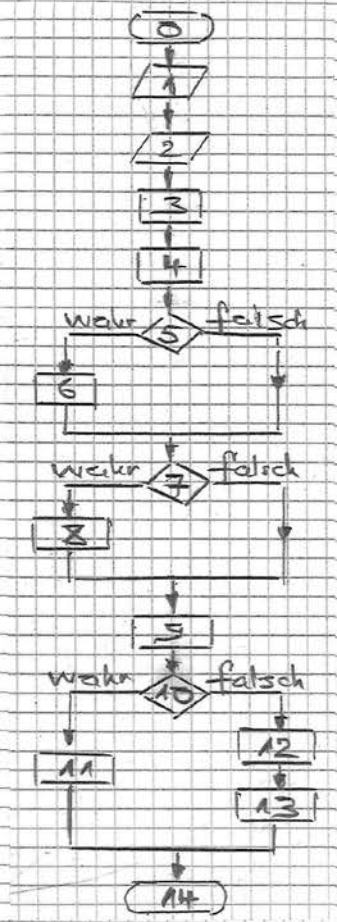
Aufgabe 34

b)

```
from random import *

a1 = randint(1,6)
msgDlg("Erste Augenzahl: ", a1)
a2 = randint(1,6)
msgDlg("Zweite Augenzahl: ", a2)
a3 = randint(1,6)
msgDlg("Dritte Augenzahl: ", a3)
a4 = randint(1,6)
msgDlg("Vierte Augenzahl: ", a4)

if a1 < a2 :
    k = a1
else :
    k = a2

if a3 < k :
    k = a3

if a4 < k :
    k = a4

msgDlg("Kleinste Augenzahl.", k)
```

Aufgabe 35

a)

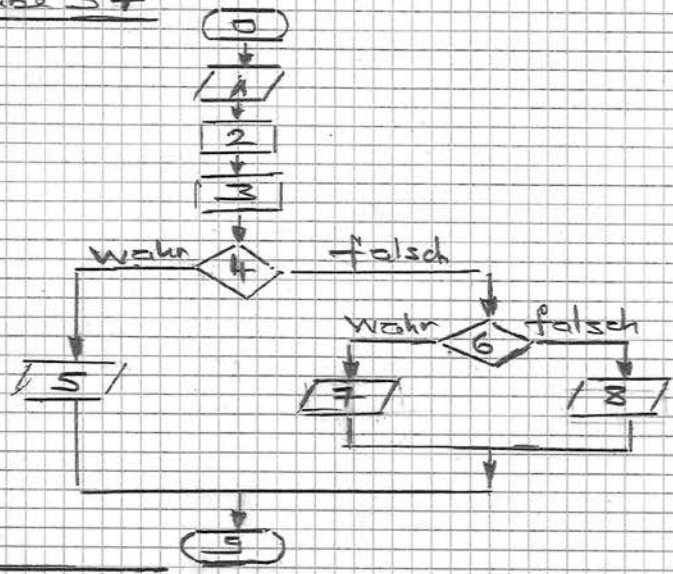


b)

Zeile	n1	n2	d1	d2	s	d1 < 0	d2 < 0	s ≥ 0
1	3.25							
2		5						
3			-0.75					
4				1				
5						wahr		
6			-1.5					
7							falsch	
9					-0.5			
10								falsch
12					0.5			

Aufgabe 37

a)

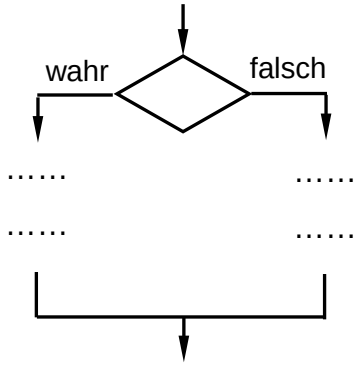


Aufgabe 38

```
from random import *
msgDlg("Zahle einen Einsatz von CHF 2.")
msgDlg("Wuerfle einmal.")
a1 = randint(1,6)
msgDlg("Du hast eine", a1, "geworfen.")
if a1 < 3 or a1 > 4 :
    if a1 > 4 :
        msgDlg("Du bekommst CHF 5 ausbezahlt.")
    else :
        msgDlg("Du bekommst CHF 0 ausbezahlt.")
else :
    msgDlg("Wuerfle noch einmal.")
    a2 = randint(1,6)
    msgDlg("Du hast eine", a2, "geworfen.")
    if a2 > 4 :
        msgDlg("Du bekommst CHF 3 ausbezahlt.")
    else :
        msgDlg("Du bekommst CHF 0 ausbezahlt.")
```

Rückblick

- Eine **Verzweigung** dient dazu, in einem Algorithmus bzw. Programm eine **Fallunterscheidung** vorzunehmen. Je nach Fall werden andere Anweisungen ausgeführt.
- Es gibt in Programmiersprachen mehrere Arten von Verzweigungen. In dieser Unterrichtseinheit verwenden wir durchgehend die **if – else – Verzweigung**. Sie unterscheidet zwei Fälle:

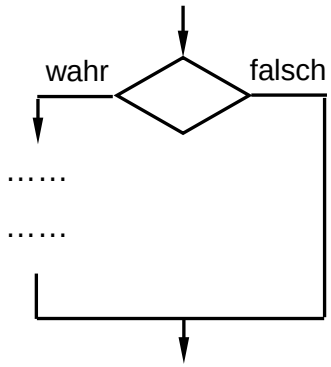



```
if bedingung :
    .....
    .....
else :
    .....
    .....
```

if – Block wird ausgeführt, wenn `bedingung` wahr ist.

else – Block wird ausgeführt, wenn `bedingung` falsch ist.

- In Python muss nach **`if`** `bedingung` und nach **`else`** ein **Doppelpunkt** stehen. Zudem müssen die Anweisungen, welche zum if – Block bzw. else – Block gehören, eine gleichmässige **Einrückung** aufweisen.
- Wenn der else – Block keine Anweisungen enthält, kann er auch weggelassen werden.




```
if bedingung :
    .....
    .....
```

- Am Anfang einer if – else – Verzweigung steht eine **Bedingung**, die entweder **wahr** oder **falsch** ist und darüber entscheidet, ob die Anweisungen im if – Block oder else – Block ausgeführt werden.
- Im Flussdiagramm werden Bedingungen durch **Rauten** dargestellt ().
- Für Bedingungen gibt es eine **Und-Verknüpfung** und eine **Oder-Verknüpfung**:

`bedingung1` **`and`** `bedingung2` **`and`** `...`	Und-Verknüpfung ergibt wahr, wenn **alle** Bedingungen wahr sind.
`bedingung1` **`or`** `bedingung2` **`or`** `...`	Oder-Verknüpfung ergibt wahr, wenn **mindestens eine** Bedingung wahr ist.