Vorkurs Informatik

Dirk Frettlöh

dfrettloeh@techfak.de

20. September 2023

Das erste Programm

Funktionen

- das Auslagern als Funktion
- Funktionen mit Parametern
- Funktionen mit return-Wert

Mastermind

- wichtige Fragen
- Der Ablauf
- Die Implementation
- random
- Weiter im Programm
- Benutzereingaben
- Mastermind
- Eingabe

Ende

Mittelwert berechnen

Wir wollen jetzt ein kleines Programm schreiben, welches den Mittelwert berechnen kann.

Wie gehen wir vor?

Wichtige Fragen, die man sich vorher stellen sollte

1. Welcher Mittelwert?
2. Wie sieht die Eingabe aus?
3. Wie sieht die Ausgabe aus?
4. Was ist der Ablauf?

1) Welcher Mittelwert?

1) Welcher Mittelwert?

Am Anfang nehmen wir das arithmetische Mittel:

$$\frac{1}{n}\sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

2) Wie sieht die Eingabe aus?

2) Wie sieht die Eingabe aus?

Um es einfach zu gestalten, nehmen wir als Eingabe einfach eine Liste an.

```
1 numbers = [1,2,3,4,5,6,7,8,9,10,33]
```

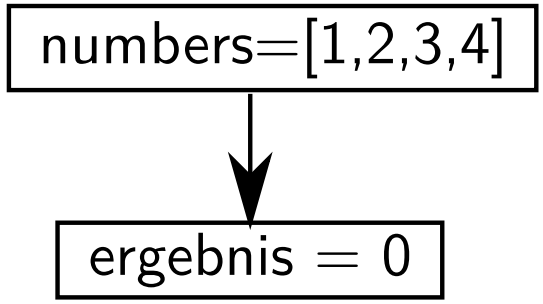
3) Wie sieht die Ausgabe aus?

3) Wie sieht die Ausgabe aus?

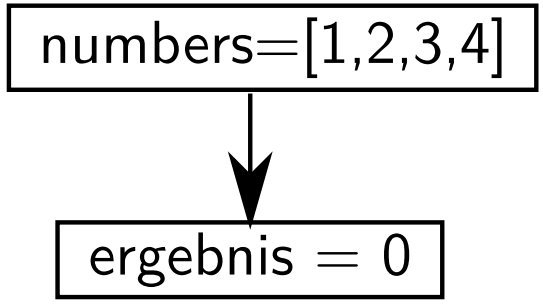
Bei der Ausgabe reicht uns ein einfaches `print`

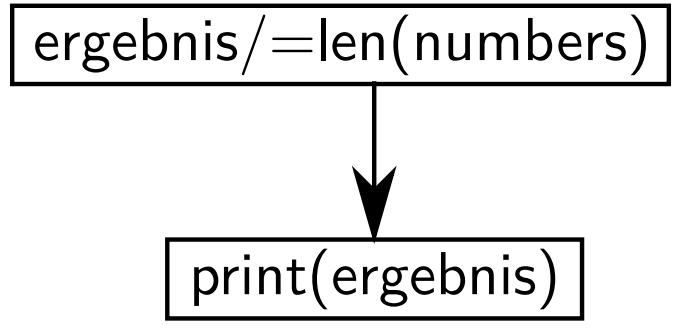
```
1 print("Das␣Arithmetische␣Mittel␣ist:", ergebnis)
```

Was wir schon wissen:

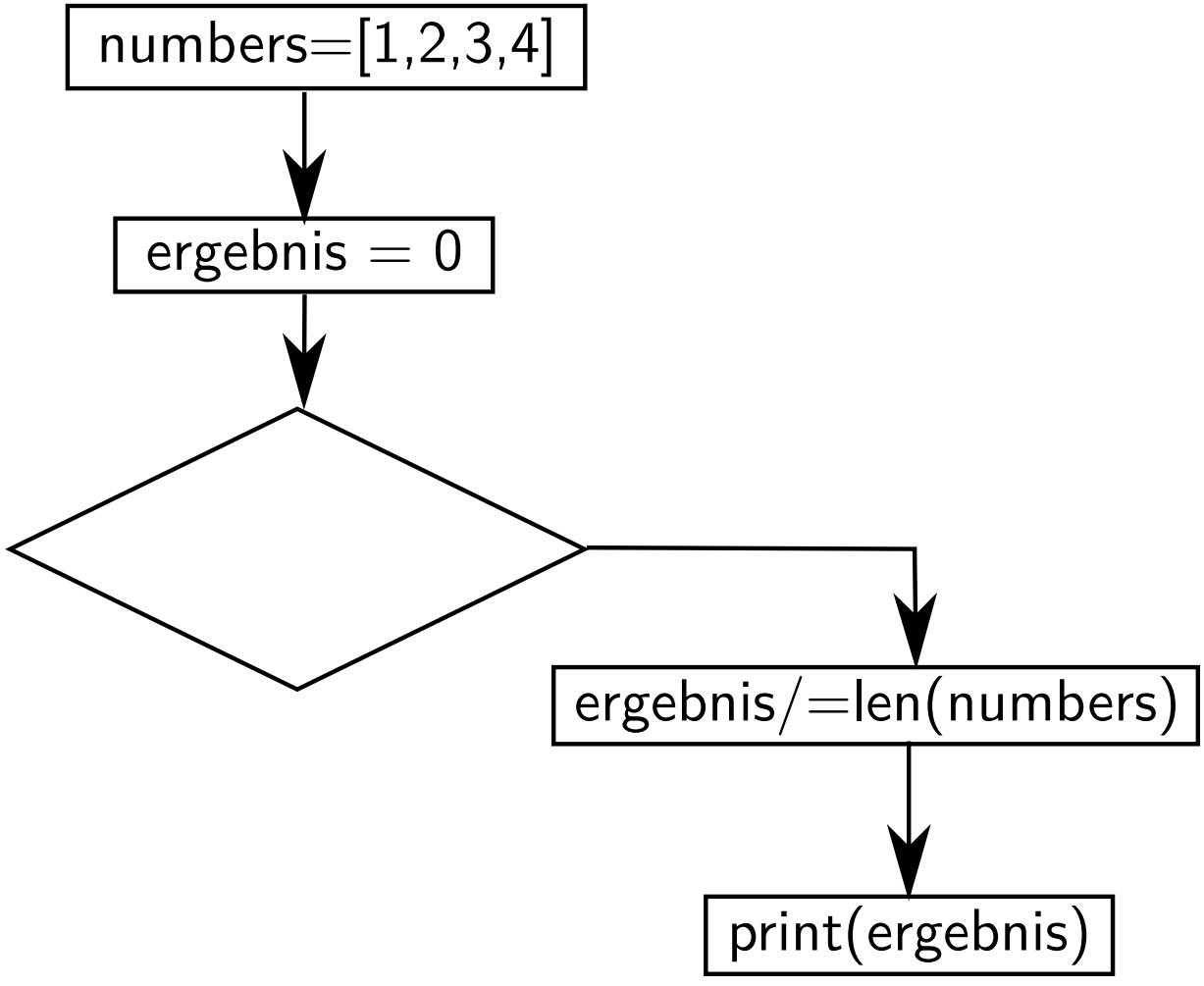


Aus der Formel ergibt sich:

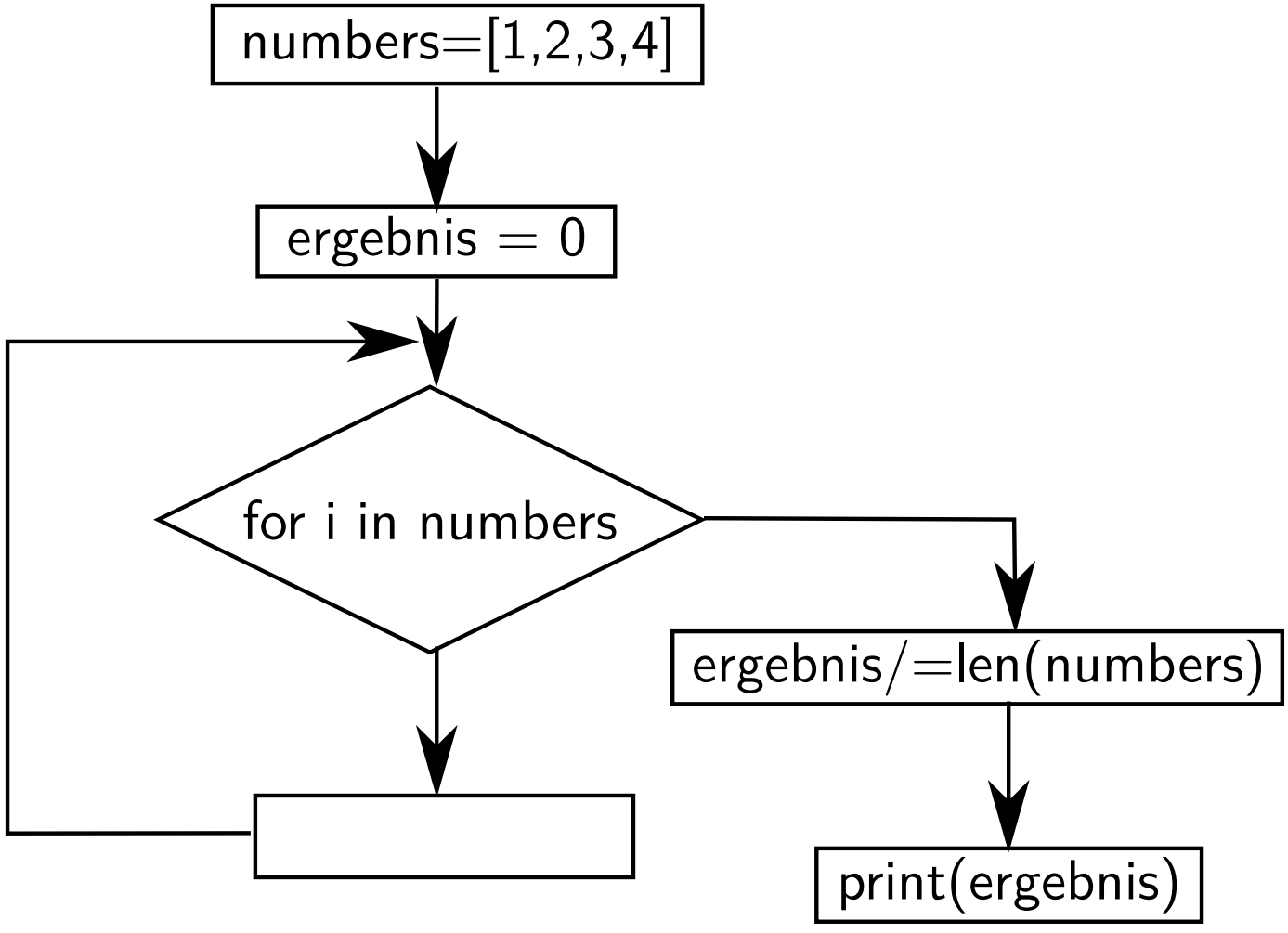




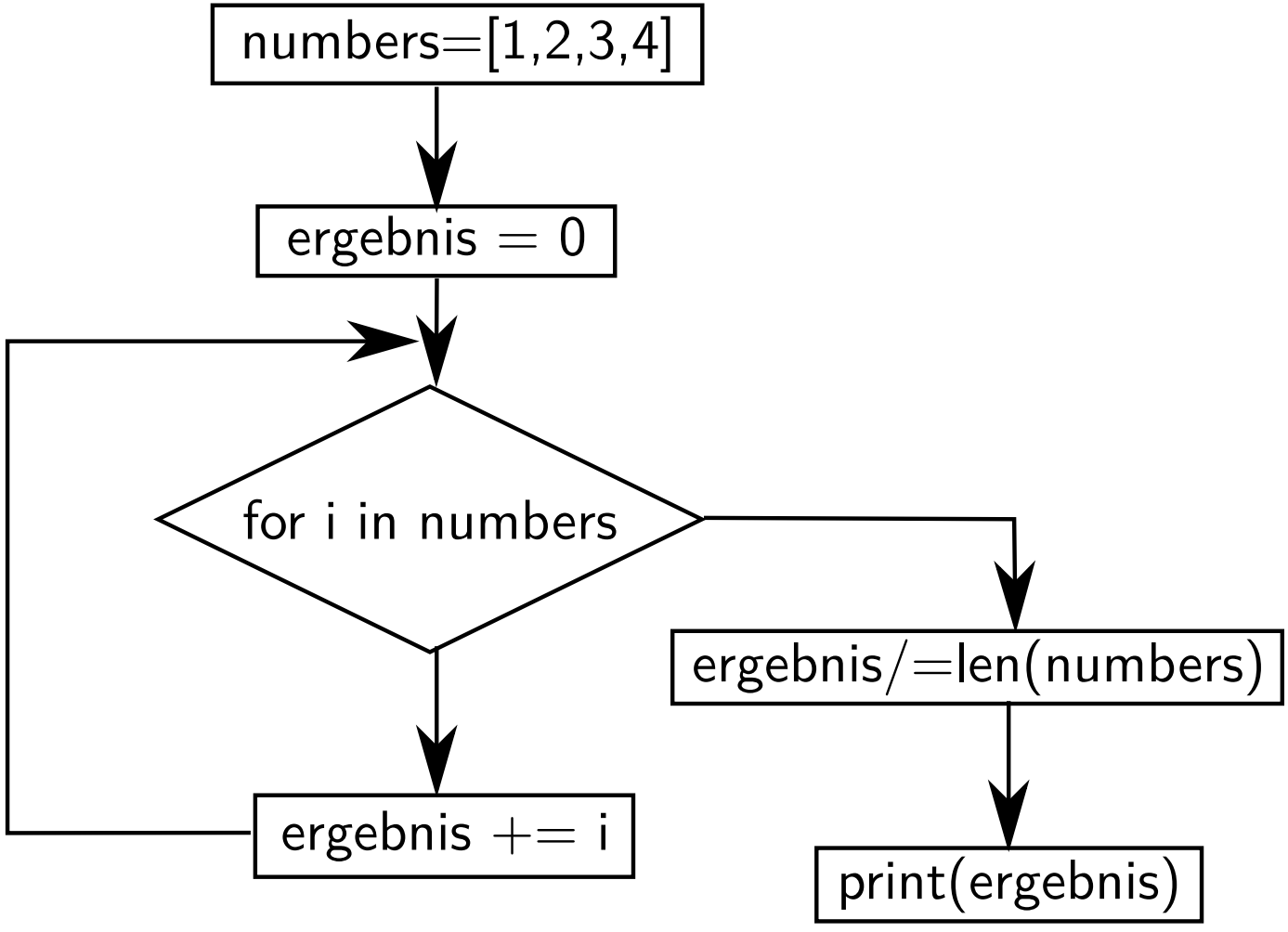
Da wir über die Liste iterieren müssen:



Dafür ist for da:



Eine Addition und wir sind fertig:



Und das Ganze jetzt in Python.

Der erste Teil in Python:

```
1 numbers= [1,2,3,4,5,6,7,8,9,10,33]
2 ergebnis = 0
3
4
5
6 print("Das␣Arithmetische␣Mittel␣ist:", ergebnis)
```

Das Teilen kommt dazu:

```
1 numbers= [1,2,3,4,5,6,7,8,9,10,33]
2 ergebnis = 0
3 ____
4 ____
5 ergebnis /= len(numbers)
6 print("Das Arithmetische Mittel ist:", ergebnis)
```

Das for wird hinzugefügt:

```
1 numbers= [1,2,3,4,5,6,7,8,9,10,33]
2 ergebnis = 0
3 for i in numbers:
4 ____
5 ergebnis /= len(numbers)
6 print("Das␣Arithmetische␣Mittel␣ist:",ergebnis)
```

Jetzt ist es fertig:

```
1 numbers= [1,2,3,4,5,6,7,8,9,10,33]
2 ergebnis = 0
3 for i in numbers:
4     ergebnis += i
5 ergebnis /= len(numbers)
6 print("Das Arithmetische Mittel ist:",ergebnis)
```

Kleine Verbesserung

Dieses kleine Programm funktioniert zwar, jedoch würde es sehr unübersichtlich werden, wenn wir weitere Mittelwerte berechnen wollen.

Kleine Verbesserung

Dieses kleine Programm funktioniert zwar, jedoch würde es sehr unübersichtlich werden, wenn wir weitere Mittelwerte berechnen wollen. Auch wäre es umständlich wenn wir eine andere Mittelwertsberechnung verwenden möchten.

Kleine Verbesserung

Dieses kleine Programm funktioniert zwar, jedoch würde es sehr unübersichtlich werden, wenn wir weitere Mittelwerte berechnen wollen. Auch wäre es umständlich wenn wir eine andere Mittelwertsberechnung verwenden möchten.

Was passiert, wenn wir bei einem größeren Projekt an mehreren Stellen den arithmetischen Mittelwert benötigen?

Kleine Verbesserung

Dieses kleine Programm funktioniert zwar, jedoch würde es sehr unübersichtlich werden, wenn wir weitere Mittelwerte berechnen wollen. Auch wäre es umständlich wenn wir eine andere Mittelwertsberechnung verwenden möchten.

Was passiert, wenn wir bei einem größeren Projekt an mehreren Stellen den arithmetischen Mittelwert benötigen?

Um mehrfaches Auftauchen von gleichen Codeschnipseln zu verhindern, gibt es Funktionen.

Schreiben wir unsere Berechung als eine Funktion:

```
1 def arithmetic_mean():
2     numbers= [1,2,3,4,5,6,7,8,9,10,33]
3     ergebnis = 0
4     for i in numbers:
5         ergebnis += i
6     ergebnis /= len(numbers)
7     print("Das␣Arithmetisches␣Mittel␣ist:",ergebniss)
```

Was ist neu?

Es ist nur eine Zeile dazu gekommen.

```
1 def arithmetic_mean():
```

`def` = es handelt sich um eine Funktion

`arithmetic_mean` = Funktionsname (dieser ist frei wählbar)

`():` = Parameter und Ende

Bevor wir uns mit den Funktionen weiter beschäftigen, gucken wir uns an, wie wir sie aufrufen können.

```
1  def arithmetic_mean():
2      numbers= [1,2,3,4,5,6,7,8,9,10,33]
3      ergebnis = 0
4      for i in numbers:
5          ergebnis += i
6      ergebnis /= len(numbers)
7      print("Das Arithmetische Mittel ist:",ergebnis)
8
9
10 arithmetic_mean()
```

Ein einfaches `arithmetic_mean()` reicht aus um unsere Funktion aufzurufen.

So etwas kennen wir doch schon?!

```
1 print("Hallo")
```

Das `print` ist auch eine Funktion. Diese Funktion wurde von den Python-Entwicklern für uns geschrieben.

So etwas kennen wir doch schon?!

```
1 print("Hallo")
```

Das `print` ist auch eine Funktion. Diese Funktion wurde von den Python-Entwicklern für uns geschrieben.

Doch bei dieser Funktion schreiben wir etwas in die runden Klammern. (Das sind die *Parameter*).

Einbauen von Parametern:

Bis jetzt haben wir:

```
1 def arithmetic_mean():
2     numbers= [1,2,3,4,5,6,7,8,9,10,33]
3     ergebnis = 0
4     for i in numbers:
5         ergebnis += i
6     ergebnis /= len(numbers)
7     print("Das␣Arithmetisches␣Mittel␣ist:",ergebnis)
```

Jedoch ergibt es ja wenig Sinn, immer für die gleichen Zahlen den Mittelwert zu bestimmen.

Besser wäre es, wenn wir die Liste als Parameter bekommen würden.

```
1 def arithmetic_mean(numbers):
2     ergebnis = 0
3     for i in numbers:
4         ergebnis += i
5     ergebnis /= len(numbers)
6     print("Das␣Arithmetische␣Mittel␣ist:",ergebnis)
```

Jetzt sieht der Aufruf unserer Funktion etwas anders aus:

```
1 zahlen  = [1,2,3,4,45,5,6,8]
2 arithmetic_mean(zahlen)
```

Zudem ist es jetzt möglich, von mehreren Listen den Mittelwert zu berechnen.

```
1 liste1 = [1,23,4,5,6,2,42,6,2,43,6,87,21,76,3]
2 liste2 = [42,433,2,1,3,45,65,632,232,564,43,5,223,65]
3 arithmetic_mean(liste1)
4 arithmetic_mean(liste2)
```

Infos zu Parametern

Eine Funktion kann auch mehrere Parameter bekommen.

```
1 def add(a,b,c,d,e):
2     ergebnis = a + b + c + d + e
3     print(ergebnis)
```

Der Aufruf der Funktion sieht dann entsprechend aus:

```
1 add(1,2,3,4,5)
```

Infos zu Parametern

Eine Funktion kann auch mehrere Parameter bekommen.

```
1 def add(a,b,c,d,e):
2     ergebnis = a + b + c + d + e
3     print(ergebnis)
```

Der Aufruf der Funktion sieht dann entsprechend aus:

```
1 add(1,2,3,4,5)
```

Wichtig ist, dass die Reihenfolge stimmt. In diesem Beispiel wäre: a = 1, b = 2, c = 3, d = 4 und e = 5

Was ist wenn wir mit dem Ergebnis noch weiter rechnen möchten?

Was ist wenn wir mit dem Ergebnis noch weiter rechnen möchten?

Auch dafür gibt es natürlich eine Lösung:

```
1 def arithmetic_mean(numbers):
2     ergebnis = 0
3     for i in numbers:
4         ergebnis += i
5     ergebnis /= len(numbers)
6     return ergebnis
```

Was ist wenn wir mit dem Ergebnis noch weiter rechnen möchten?

Auch dafür gibt es natürlich eine Lösung:

```
1 def arithmetic_mean(numbers):
2     ergebnis = 0
3     for i in numbers:
4         ergebnis += i
5     ergebnis /= len(numbers)
6     return ergebnis
```

Mit dem `return` können wir Python mitteilen, dass die Funktion hier zu Ende ist und dass der Inhalt vom Ergebnis übergeben werden soll.

Was müssen wir jetzt am Aufruf verändern?

Bis jetzt mussten wir folgendes Schreiben:

```
1 zahlen  = [1,2,3,4,45,5,6,8]
2 arithmetic_mean(zahlen)
```

Eigentlich nichts. Jedoch sehen wir keine Ausgabe mehr.

```
1 zahlen  = [1,2,3,4,45,5,6,8]
2 erg = arithmetic_mean(zahlen)
3 print("Das ergebnis ist", erg)
```

Infos zu return:

Mit dem `return` endet die Funktion sofort.

```
1 def test():
2     print("erster Text")
3     return 1
4     print("zweiter Text")
```

Bei dieser `test`-Funktion würde das zweite `print` nicht ausgeführt werden.

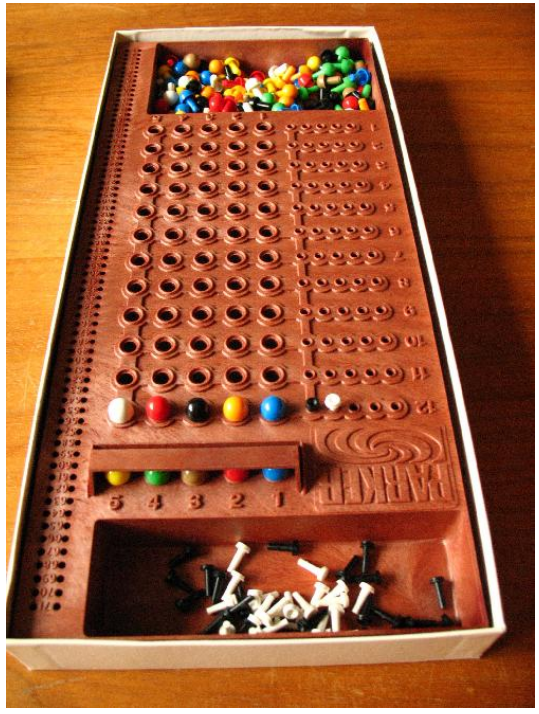
Das zweite Programm:

Das erste Programm war sehr klein und (zugegeben) auch nicht wirklich interessant. Deswegen werden wir jetzt ein kleines Spiel programmieren.

Das zweite Programm:

Das erste Programm war sehr klein und (zugegeben) auch nicht wirklich interessant. Deswegen werden wir jetzt ein kleines Spiel programmieren.

Mastermind



Wir erinnern uns an die Fragen beim ersten Programm.

Wir erinnern uns an die Fragen beim ersten Programm.

Wichtige Fragen, die man sich vorher stellen sollte:

1. Welche Regeln hat das Spiel?
2. Wie sieht die Eingabe aus?
3. Wie sieht die Ausgabe aus?
4. Was ist der Ablauf?

1) Welche Regeln hat das Spiel?

1) Welche Regeln hat das Spiel?

- Es werden 5 verschiedene (!) Farben ausgewählt und hintereinandergeschrieben.

1) Welche Regeln hat das Spiel?

- Es werden 5 verschiedene (!) Farben ausgewählt und hintereinandergeschrieben.
- Zur Auswahl stehen 8 Farben.

1) Welche Regeln hat das Spiel?

- Es werden 5 verschiedene (!) Farben ausgewählt und hintereinandergeschrieben.
- Zur Auswahl stehen 8 Farben.
- Der Spieler hat 12 Versuche, um die richtige Farbenfolge zu erraten.

1) Welche Regeln hat das Spiel?

- Es werden 5 verschiedene (!) Farben ausgewählt und hintereinandergeschrieben.
- Zur Auswahl stehen 8 Farben.
- Der Spieler hat 12 Versuche, um die richtige Farbenfolge zu erraten.
- Nach jedem Versuch bekommt er Rückmeldung, wie häufig die richtige Farbe am richtigen Platz ist (P), und wie häufig die richtige Farbe am falschen Platz ist (F).

Zum letzten Punkt: jede richtige Farbe wird nur einmal gezählt.

2) Wie sieht die Eingabe aus?

2) Wie sieht die Eingabe aus?

Wir benötigen eine Eingabe für jede Runde (z.B. eine Liste).

3) Wie sieht die Ausgabe aus?

3) Wie sieht die Ausgabe aus?

Die Ausgabe können wir mit `print` realisieren, allerdings brauchen wir dafür verschiedene Ausgaben:

1. Nach jedem inkorrekten Versuch des Spielers eine Ausgabe mit den Informationen: Runden-Nummer, Infos zur Eingabe, oder:

3) Wie sieht die Ausgabe aus?

Die Ausgabe können wir mit `print` realisieren, allerdings brauchen wir dafür verschiedene Ausgaben:

1. Nach jedem inkorrekten Versuch des Spielers eine Ausgabe mit den Informationen: Runden-Nummer, Infos zur Eingabe, oder:
2. Wenn der Spieler richtig geraten hat: “Gewonnen!”, oder:

3) Wie sieht die Ausgabe aus?

Die Ausgabe können wir mit `print` realisieren, allerdings brauchen wir dafür verschiedene Ausgaben:

1. Nach jedem inkorrekten Versuch des Spielers eine Ausgabe mit den Informationen: Runden-Nummer, Infos zur Eingabe, oder:
2. Wenn der Spieler richtig geraten hat: "Gewonnen!", oder:
3. Wenn der Spieler es nicht geschafft hat, in der maximalen Anzahl Runden richtig zu raten: "Verloren.".

3) Wie sieht die Ausgabe aus?

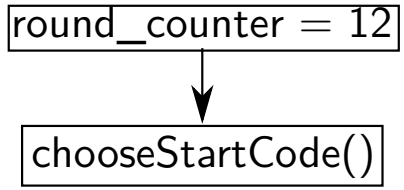
Die Ausgabe können wir mit `print` realisieren, allerdings brauchen wir dafür verschiedene Ausgaben:

1. Nach jedem inkorrekten Versuch des Spielers eine Ausgabe mit den Informationen: Runden-Nummer, Infos zur Eingabe, oder:
2. Wenn der Spieler richtig geraten hat: "Gewonnen!", oder:
3. Wenn der Spieler es nicht geschafft hat, in der maximalen Anzahl Runden richtig zu raten: "Verloren.".

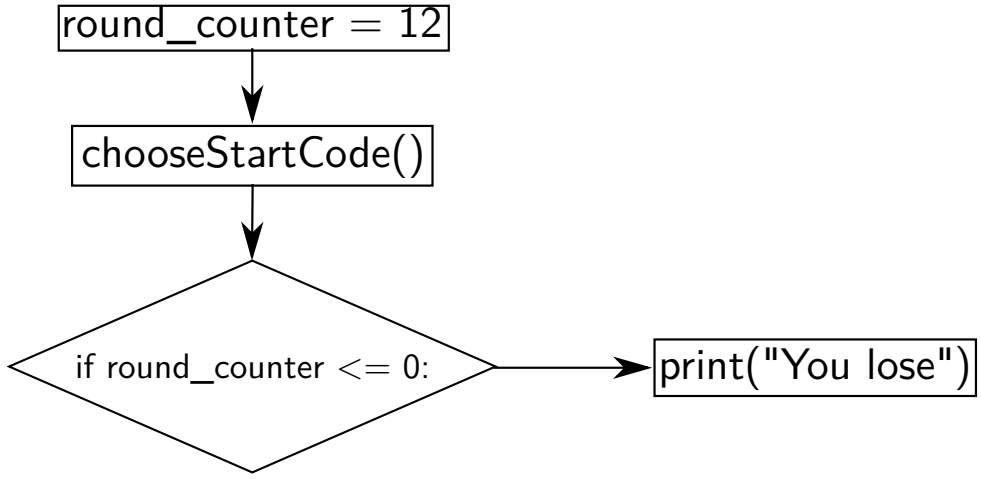
4) Und wie der Ablauf?

$\boxed{\text{round\_counter} = 12}$

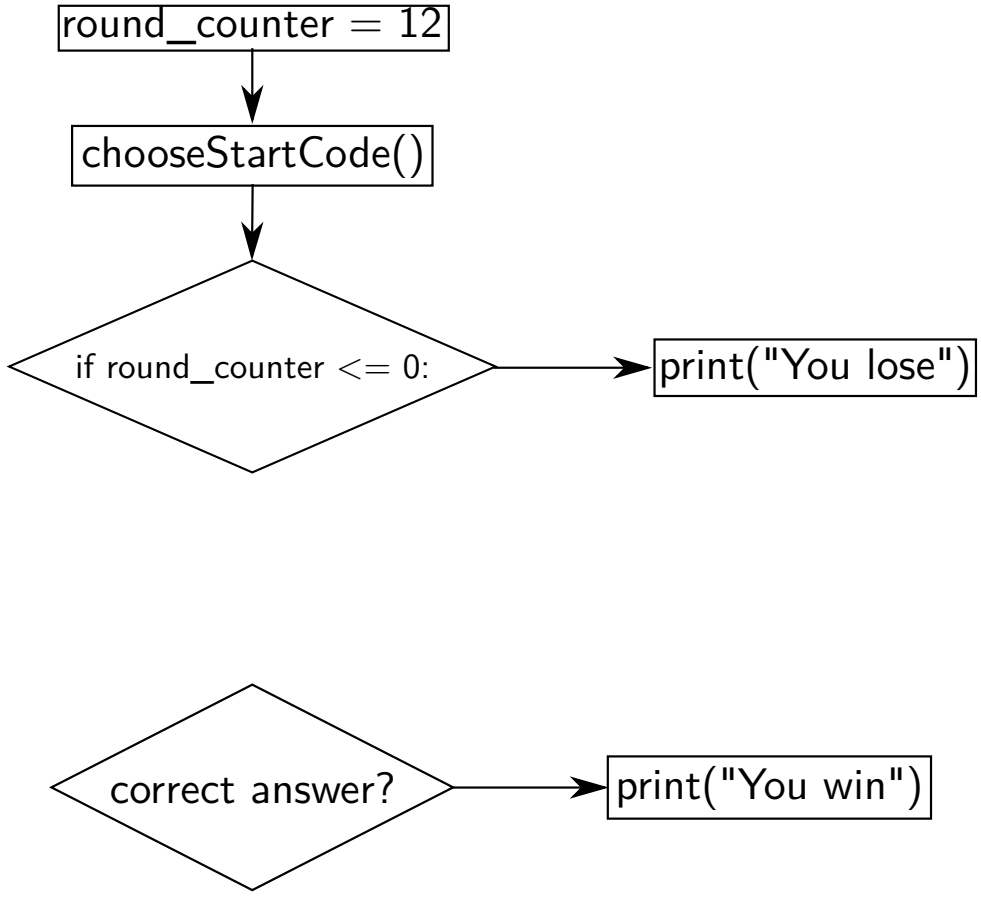
Als Erstes brauchen wir etwas, um die verbleibende Anzahl an Runden (Versuche) zu zählen.



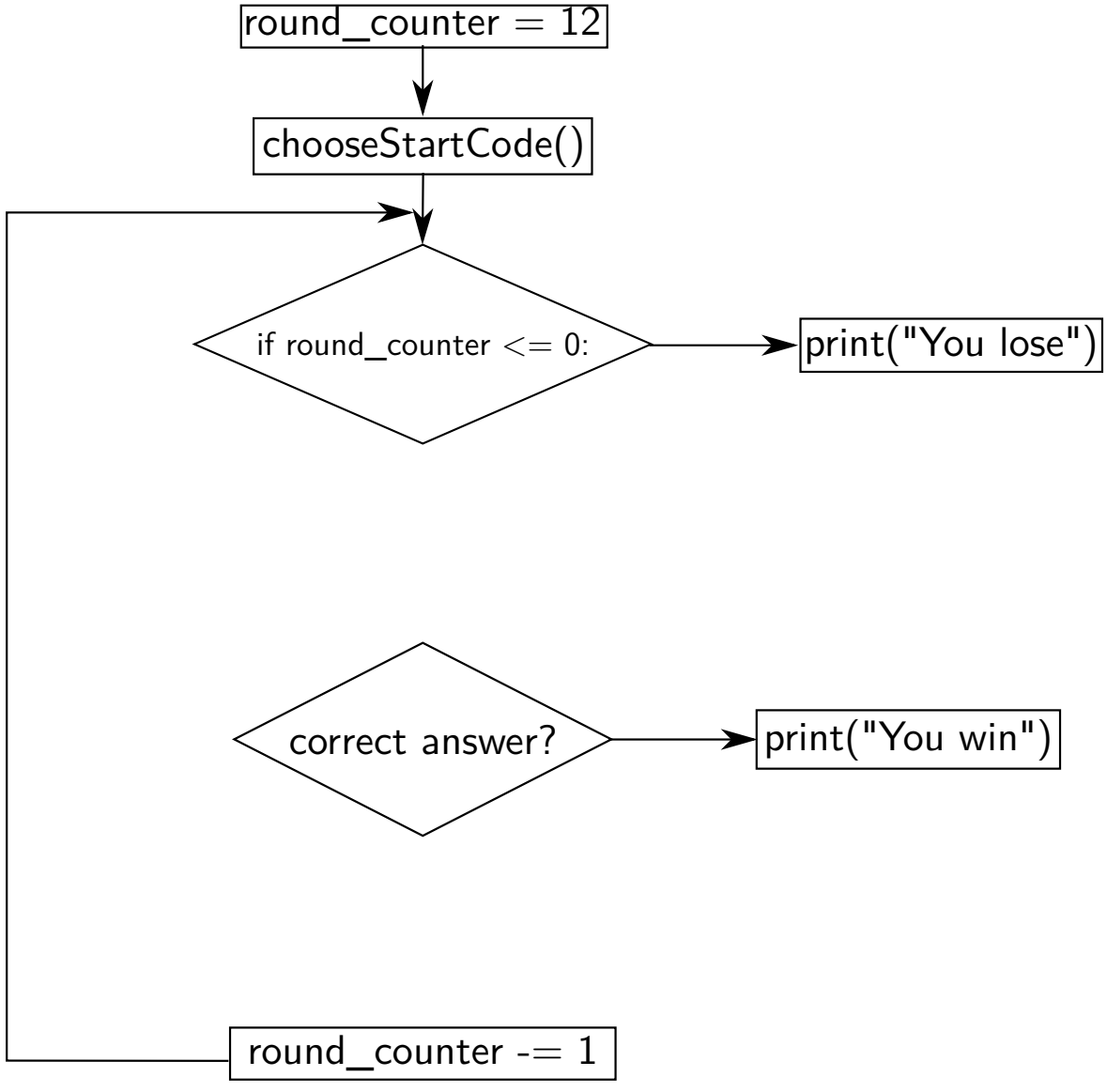
Bevor der Spieler raten kann, muss erst eine Farbkombination ausgewählt werden.



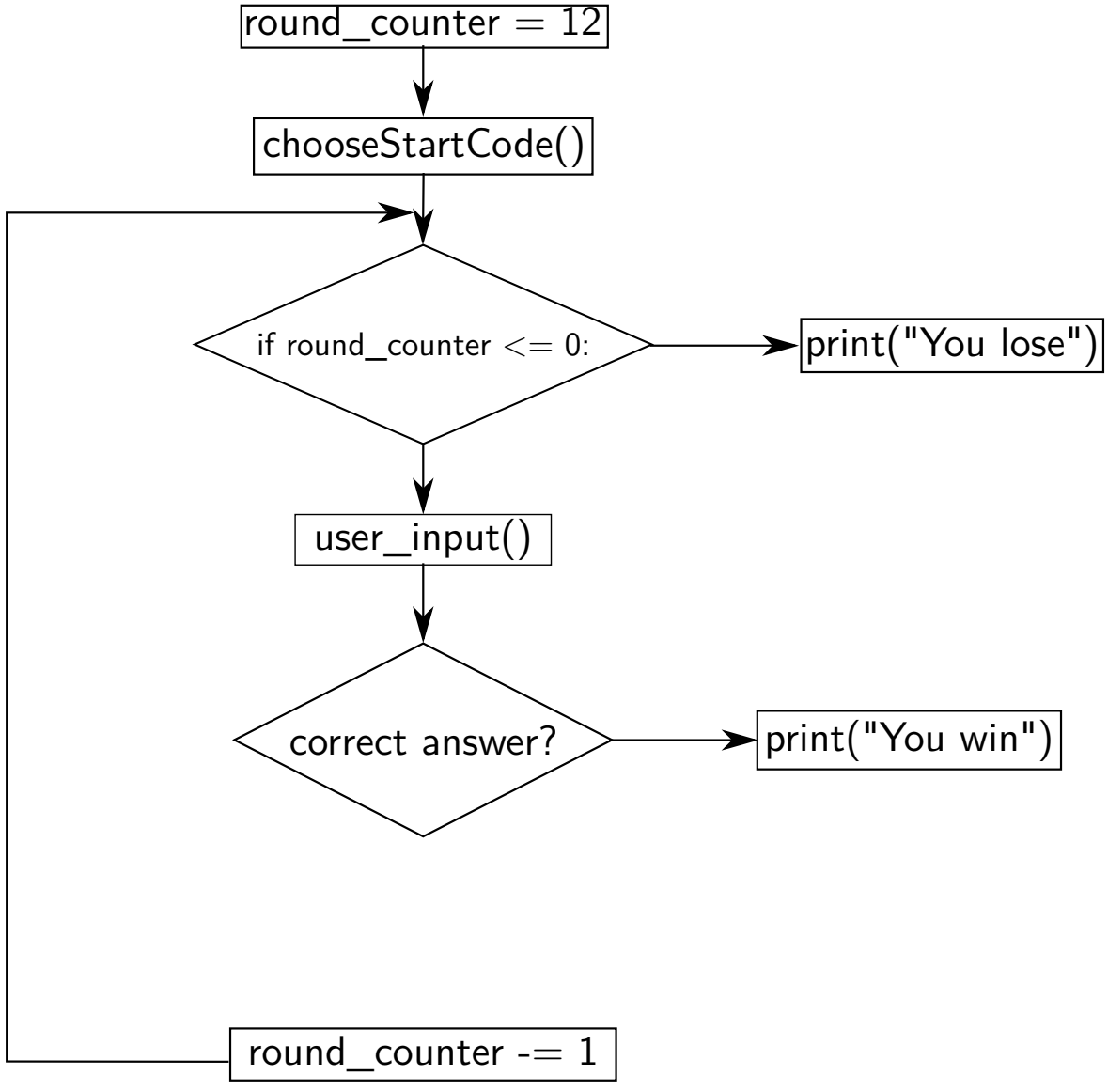
Wenn der Runden-Zähler auf 0 sinkt, hat der Spieler verloren.



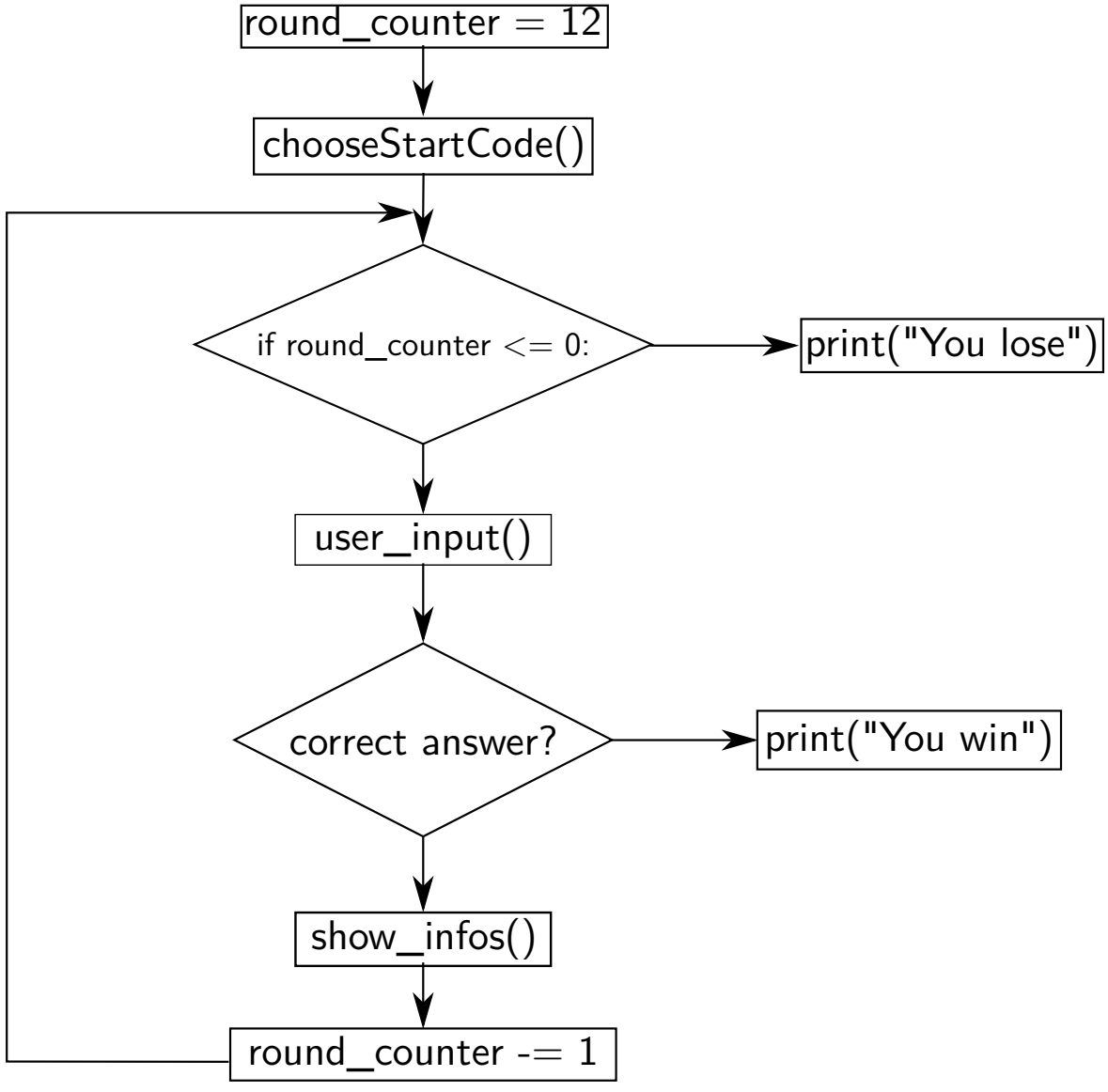
Wenn die richtige Antwort gegeben wurde, hat er gewonnen.



Der Runden-Zähler muss natürlich nach jeder Runde verringert werden.



Natürlich muss noch die Spielereingabe hinzukommen.



Zum Schluss fehlt noch die Ausgabe, in der die Eingabe bewertet wird.

Überführen wir unsere Grafik in Quellcode

```
1  def game_start():
2      round_counter = 12
3      code = chooseStartCode()
4
5      while 1:
6          if (round_counter <= 0):
7              print("Du␣hast␣verloren␣die␣richtige␣Antwort␣waere:", code)
8              return 1
9          inp = user_input()
10         ret = check_input(inp, code)
11         if (ret == 2):
12             print("Du␣hast␣die␣richtige␣Kombination␣erraten␣und␣noch",
13                     round_counter, "Runden␣uebrig.")
14             return 1
15         if (ret == 1):
16             round_counter -= 1
17         show_infos()
```

Das Erstellen von der zu ratenden Kombination:

Da die Farbeingabe etwas schwieriger ist, lassen wir den Spieler keine Farben, sondern Zahlen raten. Wir benötigen also 5 verschiedene (!) Zahlen, die zwischen 1 und 8 liegen.

```
1 def chooseStartCode():
2     code = []
3     while len(code)<5:
4         # Zufallszahl z waehlen
5         if not z in code:
6             code.append(z)
7     print("␣" * 12 + "?????")
8     return code
```

Dafür benötigen wir Zufallszahlen.

random

Um Zufallszahlen zu erstellen, gibt es die Funktion `randrange`. Diese hat als Parameter die Grenzen des Zufallszahlenbereichs (wie `range`).

```
1 randrange(1,9)
```

Jedoch kennt Python selbst diesen Befehl nicht. Wir müssen ihn erst einbinden.
Das passiert über:

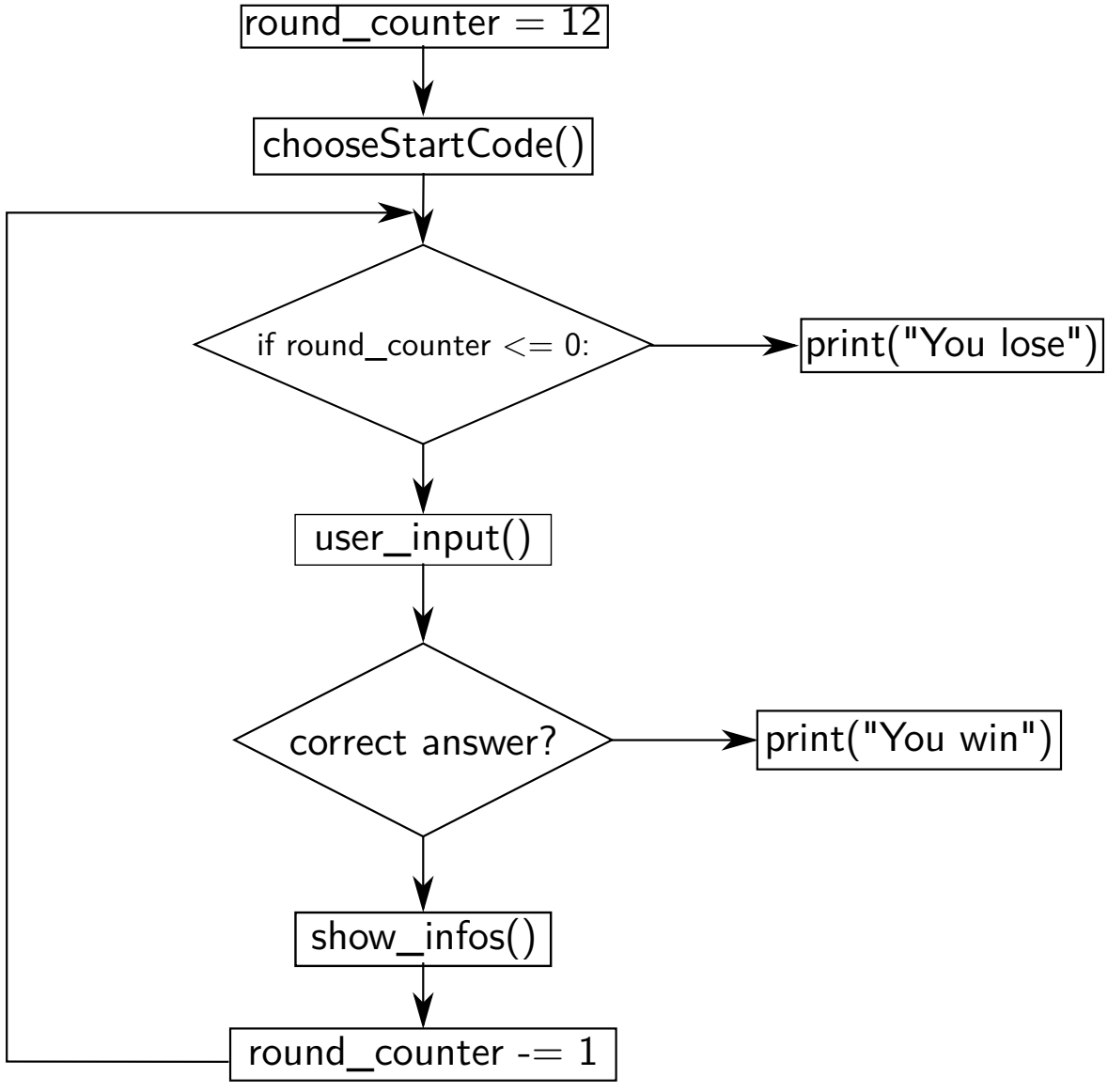
```
1 from random import randrange
```

Diese Zeile kommt einfach weit nach oben ins Programm.

Mit dieser Zeile sieht unsere Funktion folgendermaßen aus:

```
1 def chooseStartCode():
2     code = []
3     while len(code)<5:
4         z=randrange(1,8)
5         if not z in code:
6             code.append(z)
7     print("␣" * 12 + "?????")
8     return code
```

Da code eine Liste ist, können wir mit dem Befehl append einfach eine weitere Zahl hinzufügen.



Was uns jetzt noch fehlt, sind die Funktionen `user_input()` und `show_infos()`.

Wie kann der Spieler seine Zahlen-Kombination eingeben?

Für eine einfache Benutzereingabe gibt es die Funktion `input()`

```
1 eingabe = input("Was␣haben␣Sie␣zu␣sagen?")
2 print(eingabe)
```

Als Parameter kann die Funktion einen Text bekommen, der vor der Eingabe ausgegeben werden soll.
Als Rückgabe bekommt man die Eingabe des Benutzers.

Unsere Input-Funktion:

Unsere Funktion ist sehr sehr klein. Es ist also nicht nötig sie auszulagern.

```
1 def user_input():
2     inp= input("Eingabe:")
3     return inp
```

Wie sieht jetzt unsere game Funktion aus?

```
1  def game_start():
2      round_counter = 12
3      code = chooseStartCode()
4
5      while 1:
6          if (round_counter <= 0):
7              print("Du␣hast␣verloren␣die␣richtige␣Antwort␣waere:", code)
8              return 1
9          inp = input(str(round_counter) + ")" + ("␣" * 8) + ":")
10         ret = check_input(inp, code)
11         if (ret == 2):
12             print("Du␣hast␣die␣richtige␣Kombination␣erraten␣und␣noch",
                       round_counter, "runden␣uebrig." )
13             return 1
14         if (ret == 1):
15             round_counter -= 1
16         show_infos()
```

Wir überprüfen, ob die Eingabe richtig ist:

```
1 def check_input(inp, code):
2     pass
```

Was müssen wir testen?

Was müssen wir testen?

- Ist die Eingabe des Spielers fünf Zahlen lang?

Was müssen wir testen?

- Ist die Eingabe des Spielers fünf Zahlen lang?
- Hat der Spieler nur Zahlen eingegeben?
- Stimmt die Eingabe mit der Lösung überein?

Was müssen wir testen?

- Ist die Eingabe des Spielers fünf Zahlen lang?
- Hat der Spieler nur Zahlen eingegeben?
- Stimmt die Eingabe mit der Lösung überein?
- Stimmt Position und Zahl überein?

Was müssen wir testen?

- Ist die Eingabe des Spielers fünf Zahlen lang?
- Hat der Spieler nur Zahlen eingegeben?
- Stimmt die Eingabe mit der Lösung überein?
- Stimmt Position und Zahl überein?
- Kommt die Zahl in der Kombination vor?

Eingabe Test:

Am einfachsten ist zu überprüfen, ob die passende Anzahl an Zeichen eingegeben wurde.

```
1 def check_input(inp , code):
2     if (len(inp) != 5):
3         print("Keine gueltige Eingabe")
4         return 0
```

Eingabe Test:

Es muss noch getestet werden, ob der Spieler nur Zahlen eingeben hat. Dafür gibt es vorgefertigte Funktionen, mit denen getestet werden kann, ob ein String nur aus Zahlen besteht.

```
1 def check_input(inp, code):
2     if (len(inp) != 5):
3         print("Keine␣gueltige␣Eingabe")
4         return 0
5     if (inp.isdigit() != True):
6         print("Keine␣gueltige␣Eingabe")
7         return 0
```

Eingabe Test:

Auch muss überprüft werden, dass keine 0 oder 9 benutzt wurde.

```
1  def check_input(inp, code):
2      if (len(inp) != 5):
3          print("Keine gueltige Eingabe")
4          return 0
5      if (inp.isdigit() != True):
6          print("Keine gueltige Eingabe")
7          return 0
8      for i in inp:
9          if (i == "0" or i == "9"):
10             print("Keine gueltige Eingabe")
11             return 0
```

Eingabe Test:

Da wir für jede Zahl testen müssen, ob sie in unserer Kombination vorkommt, brauchen wir eine for Schleife.

```
1  def check_input(inp, code):
2      if (len(inp) != 5):
3          print("Keine␣gueltige␣Eingabe")
4          return 0
5      if (inp.isdigit() != True):
6          print("Keine␣gueltige␣Eingabe")
7          return 0
8      for i in inp:
9          if (i == "0" or i == "9"):
10             print("Keine␣gueltige␣Eingabe")
11             return 0
12     answer = []
13     for i in range(0, 5):
14         pass
```

Eingabe Test:

Als Erstes testen wir, ob die Antwort richtig ist. Wenn sie richtig ist, speichern wir ein "P" ab. Da unsere Lösung aus Zahlen besteht, unsere Eingabe aber aus Strings, müssen wir die Lösung in Strings (`str`) umwandeln.

```
1  def check_input(inp, code):
2      if (len(inp) != 5):
3          print("Keine gueltige Eingabe")
4          return 0
5      if (inp.isdigit() != True):
6          print("Keine gueltige Eingabe")
7          return 0
8      for i in inp:
9          if (i == "0" or i == "9"):
10             print("Keine gueltige Eingabe")
11             return 0
12     answer = []
13     for i in range(0, 5):
14         if (inp[i] == str(code[i])):
15             answer.append("P")
```

Eingabe Test:

Wenn die Zahl nicht richtig ist, müssen wir noch überprüfen, ob die Zahl an einer anderen Stelle vorkommt. Um das möglichst einfach zu realisieren, können wir unsere Liste fragen, wie häufig ein Element darin vorkommt.
Da unsere Lösung aus Zahlen (`int`) besteht, müssen wir die Eingabe in `int` umwandeln.

```
1 def check_input(inp, code):
2     [...]
3     answer = []
4     for i in range(0, 5):
5         if (inp[i] == str(code[i])):
6             answer.append("P")
7         elif (int(inp[i]) in (code[i])):
8             answer.append("F")
```

Eingabe-Test:

Um zu wissen, ob die Eingabe komplett richtig ist, können wir einfach zählen, wie viele "P" in unserer Auswertung vorkommen. Zudem macht es Sinn, dass wir den Tipp für den Spieler hier ausgeben.

```
1  def check_input(inp, code):
2      [...]
3      answer = []
4      for i in range(0,5):
5          if (inp[i] == str(code[i])):
6              answer.append("P")
7          elif (inp.count(str(code[i])) > 0):
8              answer.append("F")
9      if (answer.count("P") == 5 ):
10         return 2
11     print("␣" * 16, end="")
12     print(answer)
13
14     return 1
```

Noch mal zur Start Funktion.

Da wir die Ausgabe der Infos in `check input` erledigt haben, entfällt die `show_infos()` Funktion.

```
1  def game_start():
2      round_counter = 12
3      code = chooseStartCode()
4  ____
5      while 1:
6          if (round_counter <= 0):
7              print("Du␣hast␣verloren␣die␣richtige␣Antwort␣waere:", code)
8              return 1
9          inp = input(str(round_counter) + ")" + ("␣" * 8) + ":")
10         ret = check_input(inp, code)
11         if (ret == 2):
12             print("Du␣hast␣die␣richtige␣Kombination␣erraten␣und␣noch",
                   round_counter, "runden␣uebrig." )
13             return 1
14         if (ret == 1):
15             round_counter -= 1
```

Jetzt fehlt nur noch eine Sache ...

... und zwar das Ganze ausprobieren.

Jetzt fehlt nur noch eine Sache ...

... und zwar das Ganze ~~ausprobieren~~ debuggen.

(D.h., Fehler ['bugs'] ausmerzen)

```
1  def game_start():
2      round_counter = 12
3      code = chooseStartCode()
4  ____
5      while 1:
6          if (round_counter <= 0):
7              print("Du␣hast␣verloren␣die␣richtige␣Antwort␣waere:", code)
8              return 1
9          inp = input(str(round_counter) + ")" + ("␣" * 8) + ":")
10         ret = check_input(inp, code)
11         if (ret == 2):
12             print("Du␣hast␣die␣richtige␣Kombination␣erraten␣und␣noch",
                       round_counter, "runden␣uebrig." )
13             return 1
14         if (ret == 1):
15             round_counter -= 1
16
17 game_start()
```

Herzlichen Glückwunsch.
Wir haben jetzt ein Python-Spiel programmiert.

Fragen?

Geschafft

Nun habt ihr einen etwas tieferen Einstieg in Python.

Viel Spaß im Tutorium!