

## L1\_1.1 Die Sequenz

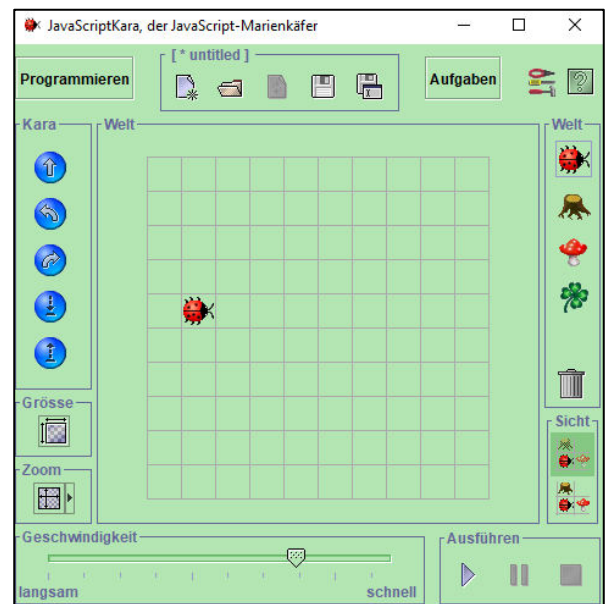
Programme bestehen aus Code, der in Abschnitte unterteilt werden kann. Diese werden als Sequenz bezeichnet.

### (I) Problemstellung

Starten Sie die Programmierumgebung PythonKara.

Vorgegeben ist eine 9 x 9 Felder große Welt, auf der verschiedene Akteure (Käfer, Blatt, Pilz und Baum) platziert werden können.

Mit Hilfe von Drag & Drop lässt sich ein neuer Käfer (*kara*) in die Welt setzen.



Hinweis: Beachten Sie zur Bearbeitung der folgenden Aufgabenstellungen die Informationsmaterialien

*L1\_1\_1 Information PythonKara Programmumgebung.docx,*

*L1\_1\_2 Information Struktogrammer.docx,*

*L1\_1\_3 Information PythonKara Sequenz.docx.*

1.1 Erweitern Sie die Welt auf 10 x 10 Felder.

1.2 Platzieren Sie den Käfer *kara* in der Welt (siehe Abbildung) und kodieren Sie ein Programm, das *kara* einen Schritt vorwärts bewegt.

Speichern Sie Ihre Ergebnisse im Ordner *meineErgebnisse* unter den Namen

*L1\_1\_1\_A1\_Sequenz.world* und *L1\_1\_1\_A1\_Sequenz.py*.

2 Das Programm soll so erweitert werden, dass der Käfer zwei Schritte vorwärts geht, anschließend nach links dreht und dann nochmals drei Schritte vorwärts geht.

Erstellen Sie ein Struktogramm zur Lösung des beschriebenen Problems und kodieren Sie die Lösung.

Speichern Sie Ihre Ergebnisse im Ordner *meineErgebnisse* unter den Namen

*L1\_1\_1\_A2\_Sequenz.stg* (Struktogramm)

*L1\_1\_1\_A2\_Sequenz.py* (Programm).

## L1\_1.1 Die Programmumgebung PythonKara

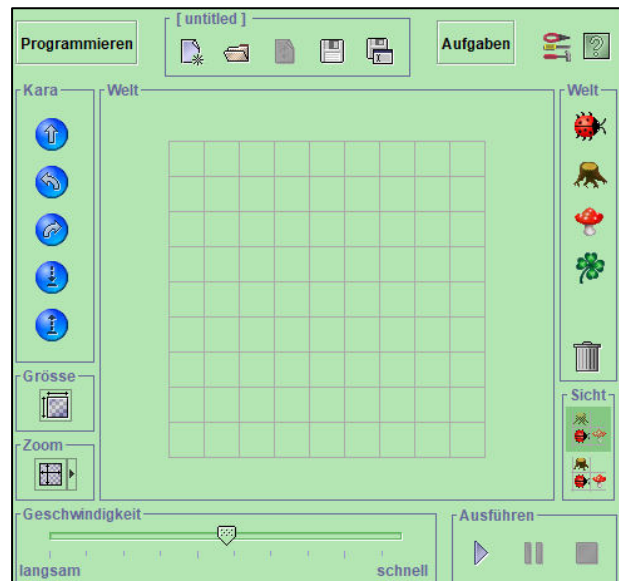
Kara ist eine interaktive Entwicklungsumgebung zur spielerischen Einführung in die Programmierung, die Anfang der 2000er Jahre an der Technischen Hochschule Zürich entwickelt wurde.

Im Laufe der Zeit wurden mehrere Versionen von Kara für die Programmierung mit unterschiedlichen Programmiersprachen zur Verfügung gestellt – u.a. auch für Python.

Sobald PythonKara gestartet wurde, erscheint die abgebildete Bildschirmoberfläche.

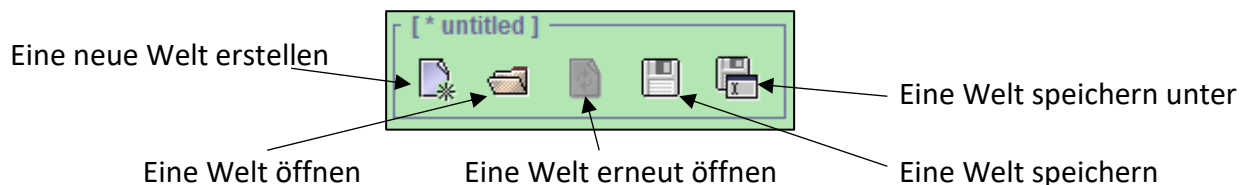
Die Vorlage enthält eine 9 x 9 Felder große Spielfläche – die Welt – und stellt verschiedene Akteure zur Verfügung.

In der Programmierumgebung PythonKara interagiert der Käfer *Kara* mit Akteuren (*Baum*, *Pilz* und *Blatt*) in einer Welt aus Quadraten.



### Erläuterungen zum Weltfenster von PythonKara

#### Welt erstellen, speichern und öffnen:



#### Akteure (Elemente) der Welt:



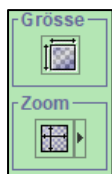
#### Elemente in die Welt einfügen:

Element anklicken und auf einem Rechteck der Welt platzieren (Drag & Drop)

#### Elemente aus der Welt entfernen:

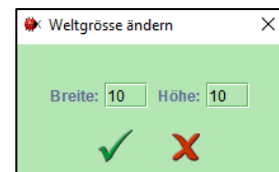
Drag auf dem zu löschenden Element starten (Klick) und über dem Abfalleimer loslassen.

### Größe und Ansicht der Welt:



Festlegen der Weltgröße. Die Standardgröße von 9 x 9 Feldern kann bis zu 1000 x 1000 Felder vergrößert werden.

Die Ansicht der Welt (Fenstergröße) kann verkleinert bzw. vergrößert werden.



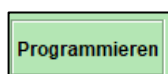
### Direkte Steuerung des Käfers :



### (zugehörige Python-Befehle)

- |                                       |                                   |
|---------------------------------------|-----------------------------------|
| Der Käfer geht einen Schritt vorwärts | → <code>kara.move();</code>       |
| Der Käfer dreht sich nach links       | → <code>kara.turnLeft();</code>   |
| Der Käfer dreht sich nach rechts      | → <code>kara.turnRight();</code>  |
| Der Käfer legt ein Blatt ab           | → <code>kara.putLeaf();</code>    |
| Der Käfer hebt ein Blatt auf          | → <code>kara.removeLeaf();</code> |

### Programmierung des Käfers



Die Schaltfläche startet den Programmeditor. Hier werden die einzelnen Aktionen festgelegt, die während des Programmablaufs „abgearbeitet“ werden sollen. Man spricht hierbei von der Kodierung des Programms

Das Editor-Fenster zeigt die Befehle (Aktionen) und Sensoren, die zur Steuerung des Käfers zur Verfügung stehen (Zeilen 1 – 8).

Der Hashtag zu Beginn der Zeilen 1 bis 12 bedeutet, dass es sich in diesen Zeilen um keine Python-Anweisungen handelt, sondern um Kommentare.

Einzeiliger Kommentar → #

Mehrzeiliger Kommentar → """  
(Drei Anführungszeichen zu Beginn eines mehrzeiligen Kommentars und drei Anführungszeichen danach.)

```
[untitled]
1 # BEFEHLE: kara.
2 # move() turnRight() turnLeft()
3 # putLeaf() removeLeaf()
4 #
5 # SENSOREN: kara.
6 # treeFront() treeLeft() treeRight()
7 # mushroomFront() onLeaf()
8 #
9
10 # hier können Sie eigene Methoden definieren
11
12 # hier kommt das Hauptprogramm hin, zB:
13 while not kara.treeFront():
14     kara.move()
15
```

Außerdem enthält das Editor-Fenster bereits einen Programmcode, der den Käfer so lange einen Schritt gehen lässt, bis er vor einem Baum steht.

Bei der Entwicklung und Implementierung eigener Programme können die Zeilen 10 – 12 (Kommentare) gelöscht werden. Die Zeilen 13 und 14 (Programmcode) müssen gelöscht werden.

### Sensoren zur Steuerung des Käfers



`kara.treeFront()`

Kara prüft, ob er vor einem Baum steht und liefert einen Wert vom Typ boolean zurück (true / false).



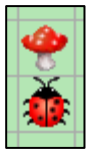
`kara.treeLeft()`

Kara prüft, ob links von ihm ein Baum steht und liefert einen Wert vom Typ boolean zurück (true / false).



`kara.treeRight()`

Kara prüft, ob rechts von ihm ein Baum steht und liefert einen Wert vom Typ boolean zurück (true / false).



`kara.mushroomFront()`

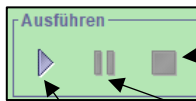
Kara prüft, ob er vor einem Pils steht und liefert einen Wert vom Typ boolean zurück (true / false).



`kara.onLeaf()`

Kara prüft, ob er auf einem Blatt steht und liefert einen Wert vom Typ boolean zurück (true / false).

### Steuerung der Programmausführung



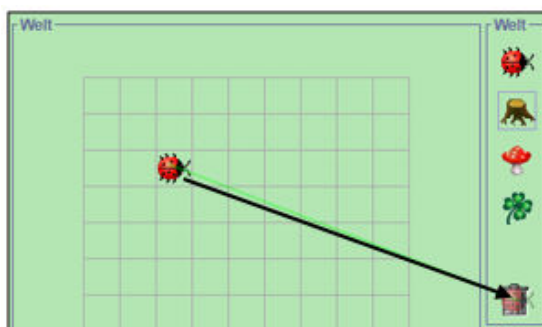
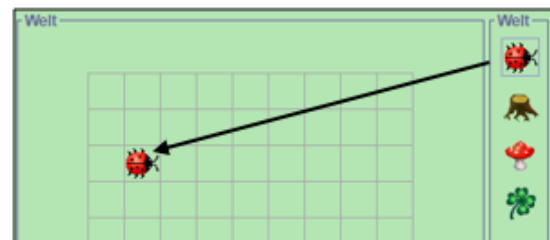
Startet das Programm  
Hält den Programmablauf an  
Stoppt das Programm



Regelt die Geschwindigkeit des Programmablaufs

### Akteure verwenden und Aktionen zuweisen

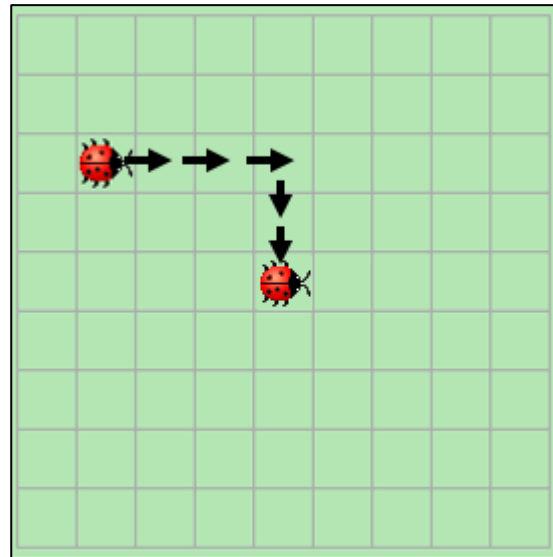
Um Akteure auf der Welt zu platzieren, wird das gewünschte Symbol in der Leiste der Element-Leiste angeklickt und per Drag & Drop auf ein beliebiges Feld der Welt gezogen.



Elemente können auch wieder von der Welt entfernt werden. Dazu wird das gewünschte Element in der Welt angeklickt und per Drag & Drop in den Abfalleimer gezogen.

Um den Käfer vom Ausgangspunkt in der dritten Zeile und zweiten Spalte zum Endpunkt in der fünften Zeile und fünften Spalte mit Blick zum rechten Rand der Welt zu bringen, muss er folgende Aktionen ausführen:

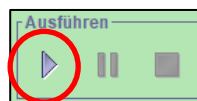
- Kara, gehe einen Schritt vorwärts
- Kara, gehe einen Schritt vorwärts
- Kara, gehe einen Schritt vorwärts
- Kara, drehe nach rechts
- Kara, gehe einen Schritt vorwärts
- Kara, gehe einen Schritt vorwärts
- Kara, drehe nach links



Um diese Anweisungen durch ein Programm ablaufen zu lassen ist folgender Python-Programmcode notwendig:

Nachdem das Programm gespeichert wurde (siehe unten), kann es mit der Schaltfläche 'Programm laufen lassen'

ausgeführt werden.



Falls das Python-Programm Fehler enthalten sollte, kann mit einem Klick auf die Fehlermeldung unterhalb des Programmeditors die Stelle ermittelt werden, an welcher der Fehler aufgetreten ist.

```

JavaScriptKara programmieren
[* test.js]
1  /* BEFEHLE: kara.
2  *   move() turnRight() turnLeft()
3  *   putLeaf() removeLeaf()
4  *
5  * SENSOREN: kara.
6  *   treeFront() treeLeft() treeRight()
7  *   mushroomFront() onLeaf()
8  */
9  kara.move()
10 kara.move()
11 kara.move()
12 kara.turnRight()
13 kara.move()
14 kara.move()
15 kara.turnLeft()

```

```

10 kara.move()
11 kara.move()
12 kara.move()
13 kara.turnRight()
14 kara.move()
15 kara.move()
16 kara.moveLeft()

```

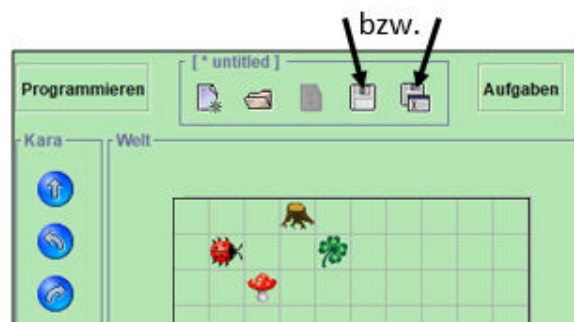
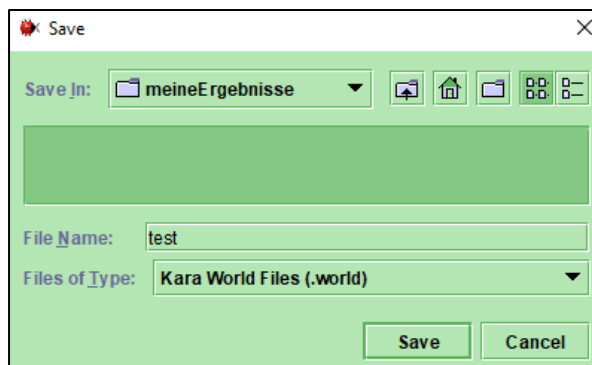
AttributeError: 'javainstance' object has no attribute 'moveLeft', in line 16

## Speichern der Welten und Programme

Beim Speichern eines mit PythonKara entwickelten Projekts muss unterschieden werden, ob nur der eingefügte Programmcode oder auch die in der Kara-Welt platzierten Akteure gespeichert werden sollen.

## Speichern der Kara-Welt

Eine Welt, in der Akteure (Käfer, Bäume etc.) platziert sind, kann gespeichert werden. Sie steht bei erneutem Öffnen in entsprechender Form wieder zur Verfügung.

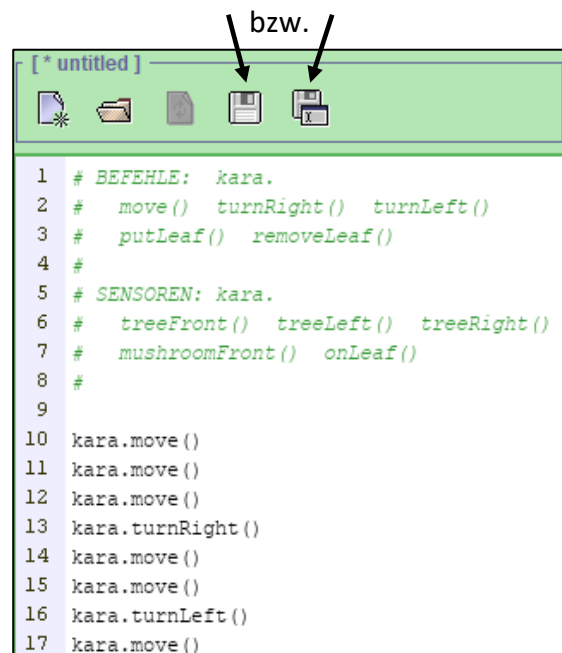
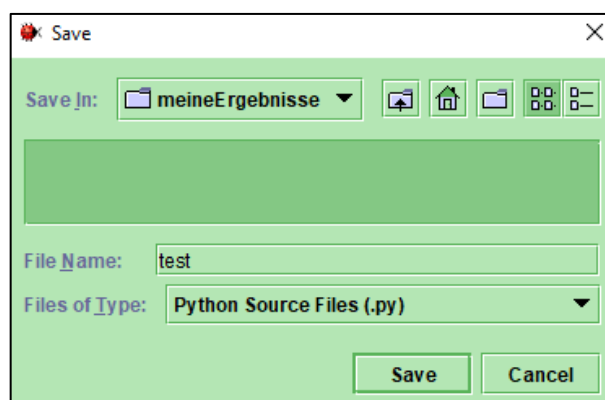


Die Datei erhält den Dateinamenszusatz 'world'.

In Ihrem Ergebnisordner befindet sich anschließend die Datei *test.world*.

## Speichern des Programmcodes

Der Programmcode zu einem Python-Projekt wird in einer separaten Datei gespeichert.



Die Datei erhält den Dateinamenszusatz 'py'. In Ihrem Ergebnisordner befindet sich danach die Datei *test.py*.

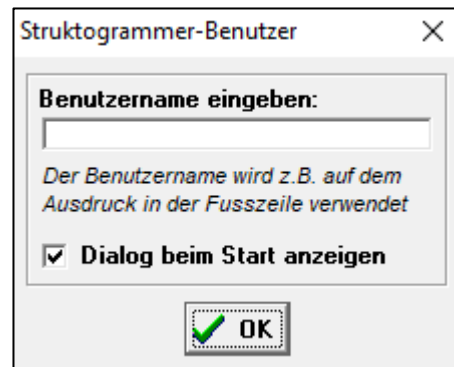
Um ein vorhandenes Python-Projekt zu bearbeiten, müssen somit immer zwei Dateien geöffnet werden, die World-Datei und die Python-Programmdatei.

## L1\_1.2 Struktogrammer

### 1 Der hus – Struktogrammer (Version: 97.05)

Bei dem hus - Struktogrammer (Version 97.05) handelt es sich um ein Software-Tool, das eine digitale Erstellung von Struktogrammen ermöglicht.

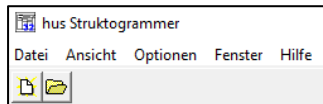
Nach dem Start der Software kann ein Benutzername eingegeben werden. Das Eingabefenster kann aber auch mit 'OK' übersprungen werden.



Ein neues Struktogramm wird mit der Befehlsfolge  
Datei

Neu

oder mit Hilfe der Schaltfläche  erzeugt.



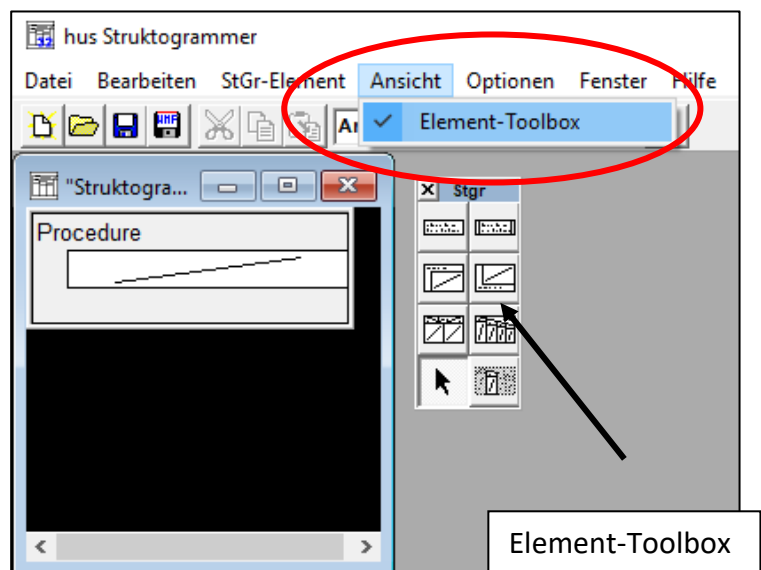
Das Struktogrammer-Fenster enthält nun die Zeichenfläche für die Struktogramm-Elemente sowie eine Element-Toolbox.

Sollte die Toolbox nicht sichtbar sein, kann sie über die Befehlsfolge  
Ansicht

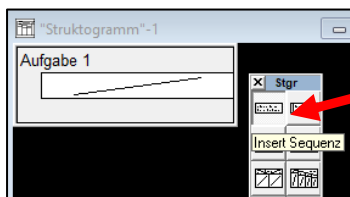
Element-Toolbox

angezeigt werden.

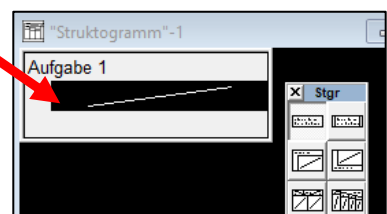
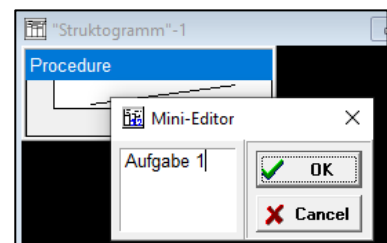
Die Größe des Struktogramms und der Zeichenfläche kann über die jeweiligen Seitenränder mit Hilfe der Maus beliebig vergrößert und verkleinert werden.

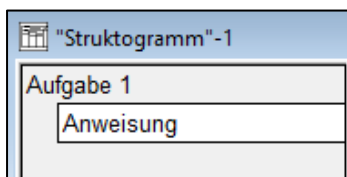


Mit einem Doppelklick auf den Struktogrammtitel ("Procedure") kann in einem Mini-Editor ein neuer Titel vergeben werden.



Die im Struktogramm benötigten Strukturelemente werden in der Element-Toolbox aktiviert und mit der veränderten Mausanzeige an die gewünschte Stelle im Struktogramm eingefügt (Drag & Drop).





Eine eingefügte Sequenz enthält den Text 'Anweisung'. Der Mini-Editor (Doppelklick) ermöglicht die Eingabe einer konkreten Anweisung (z.B.: kara, gehe einen Schritt vorwärts).

## Elemente eines Struktogramms

In den einführenden Beispielen zu den Grundlagen der Programmierung werden die Strukturelemente

Sequenz,


Wiederholung

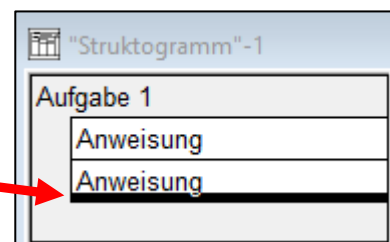
Alternative

thematisiert. Die entsprechenden Symbole stehen in der Element-Toolbox zur Verfügung.



### Sequenz

Um Sequenzen in ein Struktogramm einzufügen, muss das Sequenz-Symbol aktiviert werden (Klick auf ). Danach ändert sich die Mausanzeige in einen geschwungenen Pfeil mit Sequenz-Symbol. Anschließend muss mit der Maus auf die Stelle gezeigt werden, an der das Element eingefügt werden soll. Elemente können auch zwischen bereits vorhandenen Elementen eingefügt werden.



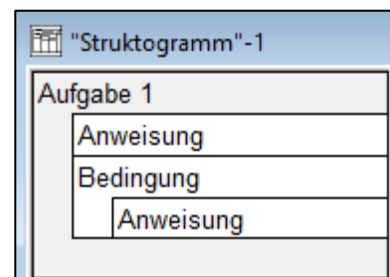
Um ein überflüssiges bzw. falsch eingefügtes Element aus dem Struktogramm zu entfernen, muss dieses markiert und mit der Entfernen-Taste gelöscht werden.

### Wiederholung (Schleife)

Das Strukturelement 'Wiederholung' zeigt eine Zeile für den Schleifenkopf (Bedingung).

Der Schleifenkörper wird eingerückt dargestellt. Im Schleifenkörper können weitere beliebige Strukturelemente eingefügt werden.

Die Aufgaben zur Wiederholungsstruktur finden Sie in L1\_2\_1 bis L1\_3\_2



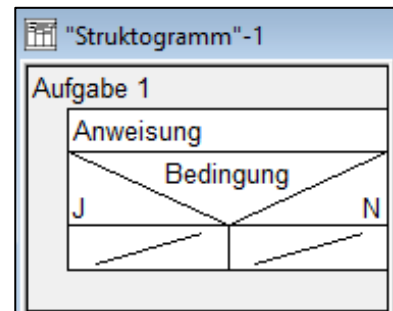


### Alternative (Verzweigung)

Das Strukturelement 'Alternative' zeigt eine Zeile für die Verzweigungsbedingung (Bedingung).

Darunter können in zwei Spalten die Strukturelemente eingefügt werden, die im Ja-Fall (linke Spalte) bzw. Nein-Fall (rechte Spalte) zu beachten sind.

Die Aufgaben zu den Alternativen finden Sie in L1\_4\_1 und L1\_4\_2

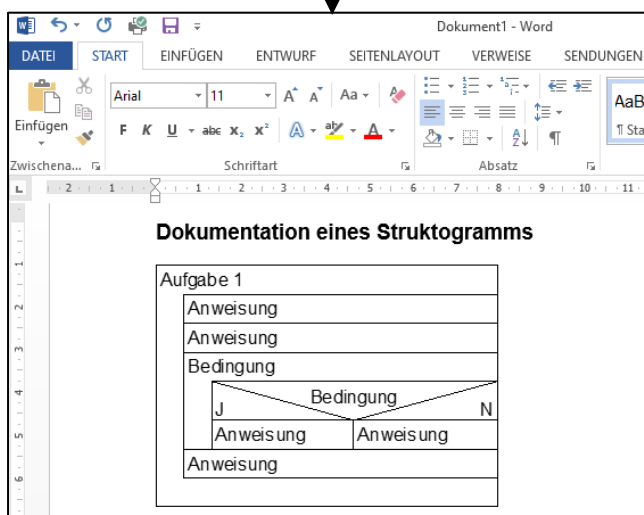
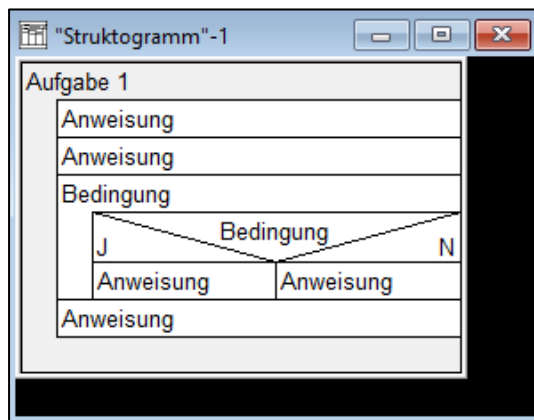
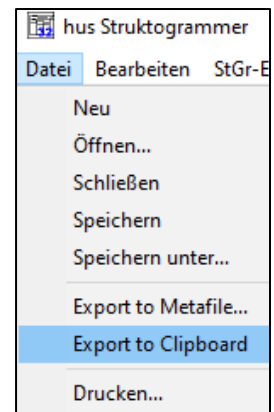


### Dokumentation eines Struktogramms

Das erstellte Struktogramm kann mit der Befehlsfolge  
Datei



Export to Clipboard

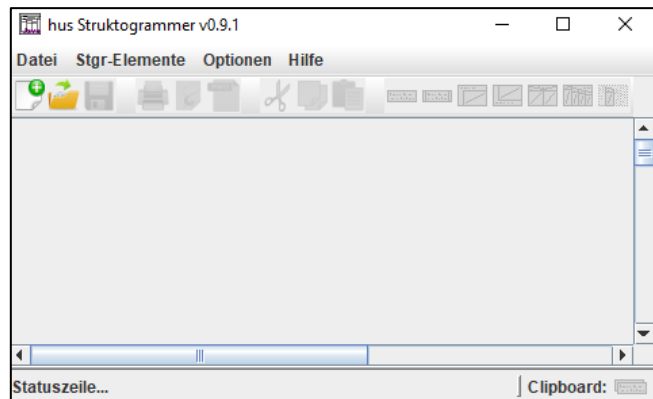
in die Zwischenablage kopiert und anschließend in einem Textverarbeitungsdocument eingefügt werden.



## 2 Der hus – Struktogrammer (Version: v0.9.1 - Java)

Bei dem hus-Struktogrammer (Version v0.9.1) handelt es sich um ein Software-Tool, das eine digitale Erstellung von Struktogrammen ermöglicht.

Nach dem Start der Software erscheint die Programmoberfläche mit vier Befehlsmenüs und zwei aktiven Befehlschaltflächen ('Neues Struktogramm'  und 'Struktogramm laden' ) sowie einer leeren Zeichenfläche.

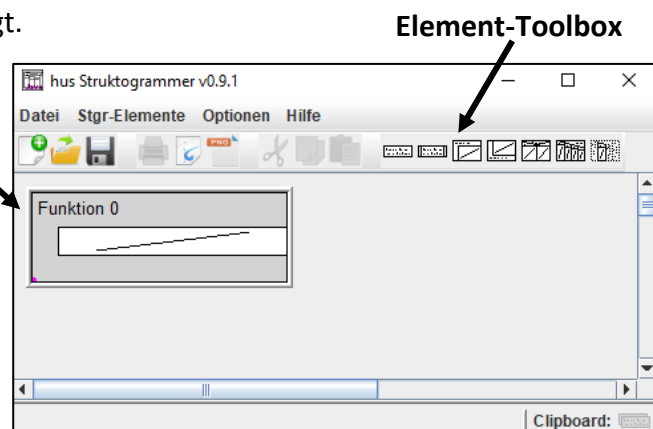


Ein neues Struktogramm wird mit der Befehlsfolge  
Datei

Neu  erzeugt.

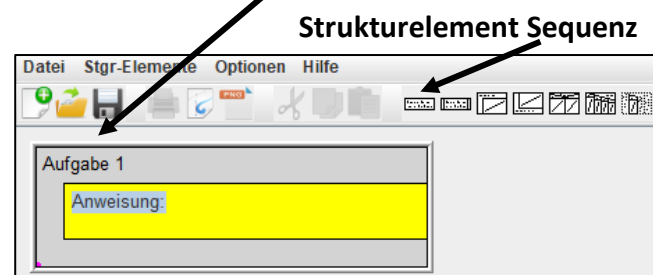
Das Struktogrammer-Fenster enthält nun eine Element-Toolbox und eine Zeichenfläche für die Struktogramm-Elemente.

Die Größe der Zeichenfläche kann über die jeweiligen Seitenränder mit Hilfe der Maus beliebig vergrößert und verkleinert werden.



Mit einem Klick in die Zeile des Struktogrammtitels ("Funktion 0") kann ein neuer Titel vergeben werden.

Die im Struktogramm benötigten Strukturelemente werden in der Element-Toolbox aktiviert und mit der veränderten Mausanzeige an die gewünschte Stelle im Struktogramm eingefügt (Drag & Drop).



Eine eingefügte Sequenz enthält den Text 'Anweisung'. Ein Klick in diese Zeile ermöglicht die Eingabe einer konkreten Anweisung (z.B.: kara, gehe einen Schritt vorwärts).

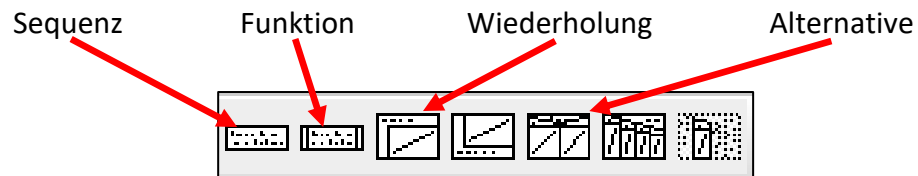
Entsprechend können weitere Strukturelemente in das Struktogramm eingefügt werden.




## Elemente eines Struktogramms

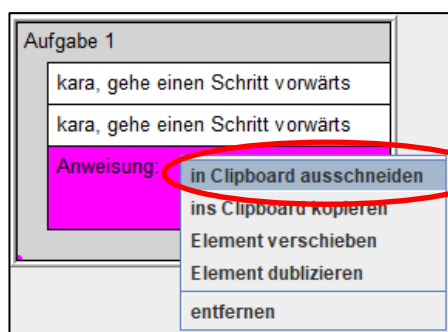
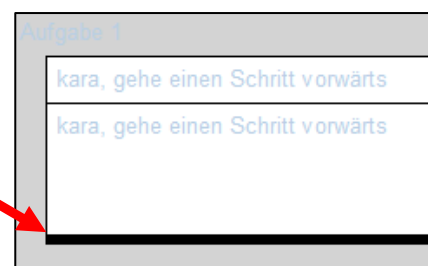
In den einführenden Beispielen zu den Grundlagen der Programmierung werden die Strukturelemente

thematisiert.



### Sequenz

Um Sequenzen in ein Struktogramm einzufügen, muss das Sequenz-Symbol aktiviert werden (Klick auf ). Danach ändert sich die Mausanzeige in einen geschwungenen Pfeil mit Sequenz-Symbol. Anschließend muss mit der Maus auf die Stelle gezeigt werden, an der das Element eingefügt werden soll. Elemente können auch zwischen bereits vorhandenen Elementen eingefügt werden.

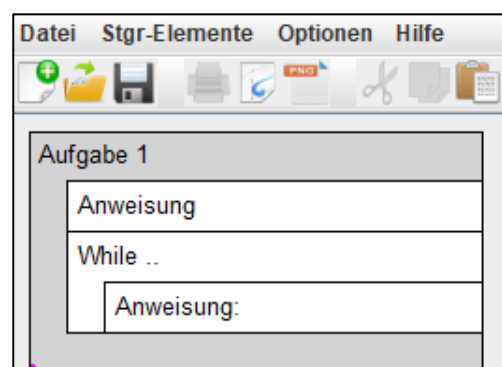


Um ein überflüssiges bzw. falsch eingefügtes Element aus dem Struktogramm zu entfernen, muss im Kontextmenü (rechte Maustaste) zu dem zu löschenden Elements die Option 'in Clipboard ausschneiden' gewählt werden.

### Wiederholung (Schleife)

Das Strukturelement 'Wiederholung' zeigt eine Zeile für den Schleifenkopf (*While ...*), in der die Schleifenbedingung eingetragen wird.

Der Schleifenkörper wird eingerückt dargestellt. Im Schleifenkörper können weitere beliebige Strukturelemente eingefügt werden. Die konkreten Anweisungen werden wie oben beschrieben eingetragen. Die Aufgaben zur Wiederholungsstruktur finden Sie in den Arbeitsaufträgen L1\_2\_1 bis L1\_3\_2.

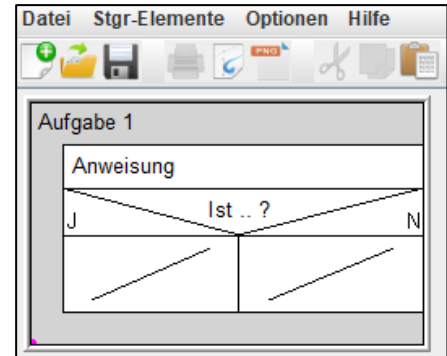


### Alternative (Verzweigung)

Das Strukturelement 'Alternative' zeigt eine Zeile für die Verzweigungsbedingung (*Ist ..?*). Hier wird die konkrete Bedingung für die Verzweigung eingetragen.

Darunter können in zwei Spalten die Strukturelemente eingefügt werden, die im Ja-Fall (linke Spalte) bzw. Nein-Fall (rechte Spalte) zu beachten sind.

Die Aufgaben zu den Alternativen finden Sie in den Arbeitsaufträgen L1\_4\_1 und L1\_4\_2.



### Dokumentation eines Struktogramms

Das erstellte Struktogramm kann für einen späteren Zugriff gespeichert werden. Die gespeicherte Datei erhält den Dateinamenszusatz *.stgr* und kann nur mit der Struktogrammer-Software geöffnet werden.

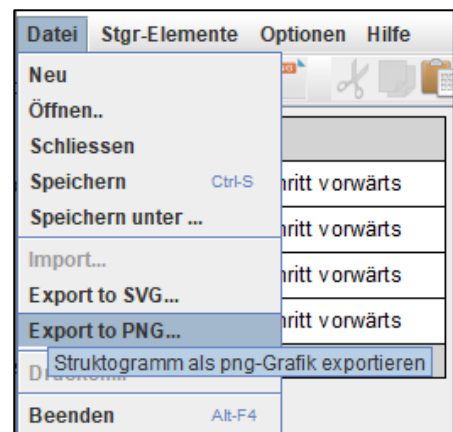
Zur Dokumentation des Arbeitsergebnisses bietet sich an, das Struktogramm als Grafikdatei zu exportieren.

Befehlsfolge:

- Datei
- Export to PNG ...

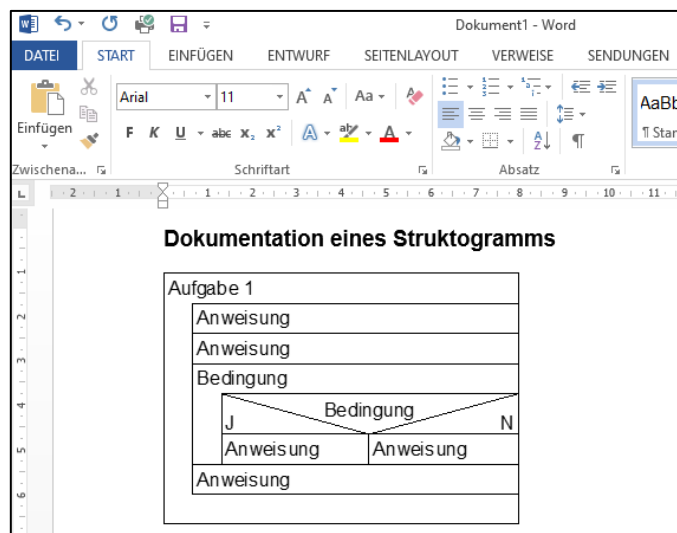
(Die gespeicherte Graphikdatei erhält den Dateinamenszusatz *.png*.)

Anschließend kann das Struktogramm als Bild in ein Textverarbeitungsprogramm (z.B. Microsoft Word) eingefügt werden.



Befehlsfolge:

- Einfügen
- Bilder
- Aus Datei



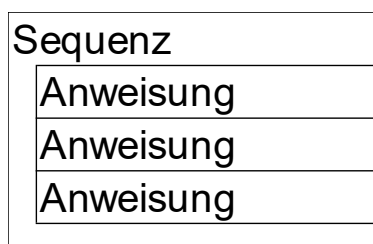
## L1\_1.3 Die Sequenz (Lineare Ablauf)

Die Entwicklung von Problemlösungen mit Hilfe einer Software erfolgt in der Regel in zwei Schritten:

- ▶ Strukturierung der einzelnen Arbeitsschritte (Struktogramm),
- ▶ Übersetzung der Arbeitsschritte in eine Programmiersprache (Programmcode).

### Struktogramm

---



Jede einzelne Anweisung wird in einen rechteckigen Strukturblock geschrieben. Die Strukturblocke werden nacheinander von oben nach unten durchlaufen. Leere Strukturblocke sind nur in Verzweigungen zulässig.

### Programmcode

---

Eine Anweisungssequenz ist eine Folge von Anweisungen, die der Reihe nach ausgeführt werden.

PythonKara-Programme bestehen in der Regel aus vielen Anweisungen.

```
# BEFEHLE: kara.  
# move() turnRight() turnLeft()  
# putLeaf() removeLeaf()  
#  
# SENSOREN: kara.  
# treeFront() treeLeft() treeRight()  
# mushroomFront() onLeaf()  
#
```

```
kara.move()  
kara.move()  
kara.move()  
kara.turnRight()  
kara.move()  
kara.move()  
kara.turnLeft()  
kara.move()
```

## L1\_1.1 Die Sequenz

1

```

1  /* BEFEHLE: kara.
2  *   move() turnRight() turnLeft()
3  *   putLeaf() removeLeaf()
4  *
5  * SENSOREN: kara.
6  *   treeFront() treeLeft() treeRight()
7  *   mushroomFront() onLeaf()
8  */
9
10 kara.move()

```

(L1\_1\_1\_A1\_Sequenz.py)

2

### Tafelanschrieb / Struktogramm

Aufgabe 2
kara, gehe einen Schritt vorwärts
kara, gehe einen Schritt vorwärts
kara, drehe nach links
kara, gehe einen Schritt vorwärts
kara, gehe einen Schritt vorwärts
kara, gehe einen Schritt vorwärts

(L1\_1\_1\_A2\_Sequenz.stg)

### Programmcode

```

1  /* BEFEHLE: kara.
2  *   move() turnRight() turnLeft()
3  *   putLeaf() removeLeaf()
4  *
5  * SENSOREN: kara.
6  *   treeFront() treeLeft() treeRight()
7  *   mushroomFront() onLeaf()
8  */
9
10 kara.move()
11 kara.move()
12 kara.turnLeft()
13 kara.move()
14 kara.move()
15 kara.move()

```

(L1\_1\_1\_A2\_Sequenz.py)

## L1\_1.2 Übungsaufgaben zur Sequenz

### Aufgabe 1

---



Welche Aktionen brauchen Sie, damit *kara* den Baum anschaut?  
Erstellen Sie ein entsprechendes Struktogramm und kodieren Sie die Lösung.

Speichern Sie Ihre Ergebnisse im Ordner *meineErgebnisse* unter den Namen *L1\_1\_2\_A1\_Sequenz.world* und *L1\_1\_2\_A1\_Sequenz.py*.

### Aufgabe 2

---

In einer Welt befindet sich ein Käfer (*kara*) und ein Baum (*tree*). Mit welcher Aktion kann *kara* überprüfen, ob er vor einem Baum steht?

### Aufgabe 3

---

Was passiert, wenn Sie den Käfer *kara* mittels der Aktion *move()* in einen Baum laufen lassen?

### Aufgabe 4

---

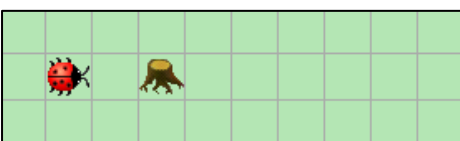
Gestalten Sie die Welt so, dass der Käfer *kara* vor einem Baum steht und speichern Sie diese unter dem Namen *L1\_1\_2\_A4\_Sequenz.world* im Ordner *meineErgebnisse*.

Der Käfer *kara* soll diesen umgehen und zwei Felder hinter dem Baum stehen bleiben.

Erstellen Sie ein Struktogramm zur Lösung des beschriebenen Problems und kodieren Sie die Lösung.

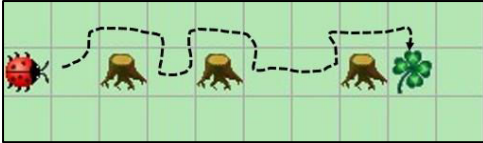
Speichern Sie Ihre Ergebnisse im Ordner *meineErgebnisse* unter den Namen *L1\_1\_2\_A4\_Sequenz.stg* (Struktogramm)

*L1\_1\_2\_A4\_Sequenz.world* und *L1\_1\_2\_A4\_Sequenz.py* (Programm)



## Aufgabe 5

Erstellen Sie abgebildete Welt und speichern Sie diese unter dem Namen *L1\_1\_2\_A5\_Sequenz.world*.



Entwickeln Sie eine Lösung, die den Käfer *kara* zum Kleeblatt führt, indem er immer wieder den Weg zwischen den Bäumen wählt (siehe Abb.). Beim Kleeblatt angekommen, soll er dieses aufheben.

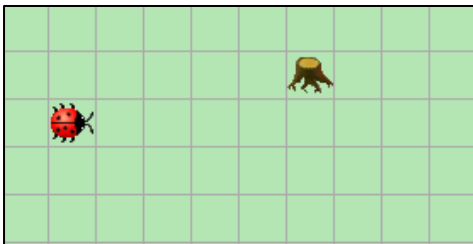
Erstellen Sie ein Struktogramm und kodieren Sie die Lösung.

Speichern Sie Ihre Ergebnisse im Ordner *meineErgebnisse* unter den Name

*L1\_1\_2\_A5\_Sequenz.stg* (Struktogramm)

*L1\_1\_2\_A5\_Sequenz.world* und *L1\_1\_2\_A5\_Sequenz.py* (Programm)

## Aufgabe 6



Der Käfer *kara* soll zu dem Baum gehen und vor ihm stehen bleiben. Dazu wurde der folgende Programmcode entwickelt.

```
10 kara.turnRight()
11 kara.move()
12 kara.turnLeft()
13 kara.move()
14 kara.move()
15 kara.move()
```

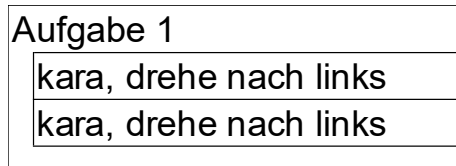
Beschreiben Sie die Wirkungsweise des Programmcodes und beurteilen Sie, ob der Käfer *kara* sein Ziel erreicht.



## L1\_1.2 Die Sequenz – Übungsaufgaben

### Aufgabe 1

Struktogramm



(L1\_1\_2\_A1\_Sequenz.stg)

(Eine Rechtsdrehung ist ebenfalls möglich.)

Programmcode

```
10 kara.turnLeft()
11 kara.turnLeft()
```

(L1\_1\_2\_A1\_Sequenz.py)

### Aufgabe 2

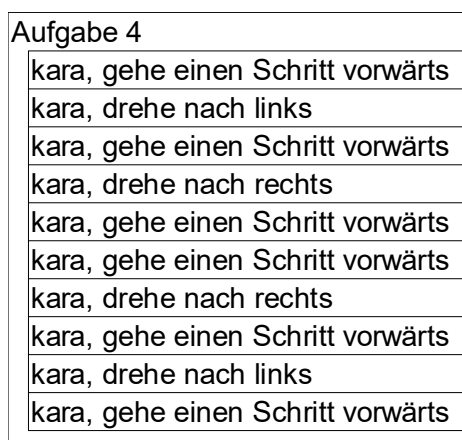
Mit der Aktion *treeFront()* kann *kara* prüfen, ob er vor einem Baum steht.

### Aufgabe 3

Der Käfer *kara* reklamiert, dass er keinen Schritt gehen kann, da er vor einem Baum steht.

### Aufgabe 4

Struktogramm



(L1\_1\_2\_A4\_Sequenz.stg)

Programmcode

```
10 kara.move()
11 kara.turnLeft()
12 kara.move()
13 kara.turnRight()
14 kara.move()
15 kara.move()
16 kara.turnRight()
17 kara.move()
18 kara.turnLeft()
19 kara.move()
```

(L1\_1\_2\_A4\_Sequenz.py)

## Aufgabe 5

### Struktogramm

Aufgabe 5
kara, gehe einen Schritt vorwärts
kara, drehe nach links
kara, gehe einen Schritt vorwärts
kara, drehe nach rechts
kara, gehe einen Schritt vorwärts
kara, gehe einen Schritt vorwärts
kara, drehe nach rechts
kara, gehe einen Schritt vorwärts
kara, drehe nach links
kara, drehe nach links
kara, gehe einen Schritt vorwärts
kara, drehe nach rechts
kara, gehe einen Schritt vorwärts
kara, gehe einen Schritt vorwärts
kara, drehe nach rechts
kara, gehe einen Schritt vorwärts
kara, drehe nach links
kara, gehe einen Schritt vorwärts
kara, drehe nach links
kara, gehe einen Schritt vorwärts
kara, drehe nach rechts
kara, gehe einen Schritt vorwärts
kara, gehe einen Schritt vorwärts
kara, drehe nach rechts
kara, gehe einen Schritt vorwärts
kara, drehe nach links
kara, hebe ein Blatt auf

(L1\_1\_2\_A5\_Sequenz.stg)

### Programmcode

```

10 kara.move ()
11 kara.turnLeft ()
12 kara.move ()
13 kara.turnRight ()
14 kara.move ()
15 kara.move ()
16 kara.turnRight ()
17 kara.move ()
18 kara.turnLeft ()
19 kara.turnLeft ()
20 kara.move ()
21 kara.turnRight ()
22 kara.move ()
23 kara.move ()
24 kara.turnRight ()
25 kara.move ()
26 kara.turnLeft ()
27 kara.move ()
28 kara.turnLeft ()
29 kara.move ()
30 kara.turnRight ()
31 kara.move ()
32 kara.move ()
33 kara.turnRight ()
34 kara.move ()
35 kara.turnLeft ()
36 kara.removeLeaf ()

```

L1\_1\_2\_A5\_Sequenz.py)

## L1\_2.1 Die Zählerschleife (for-Schleife)

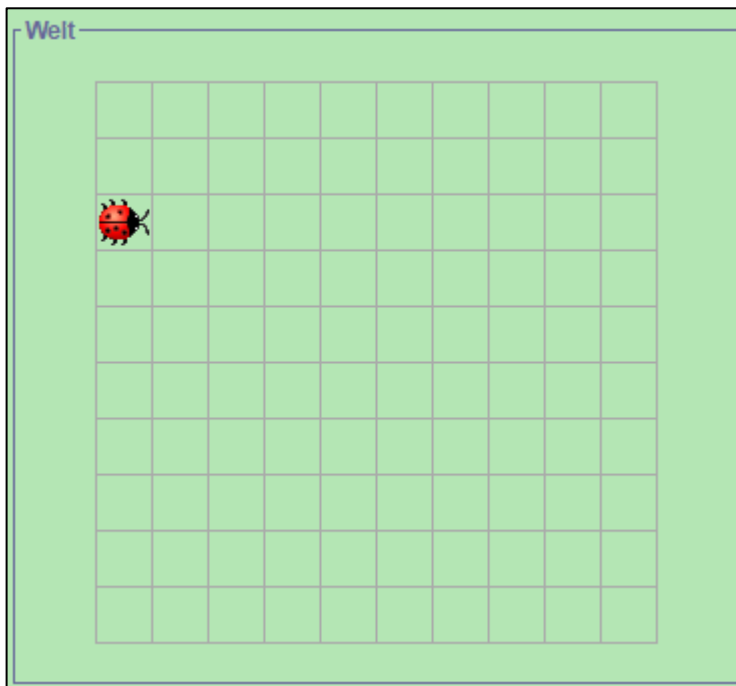
### (I) Problemstellung

---

Starten Sie die Programmierumgebung PythonKara.

Erzeugen Sie einen neuen Käfer *kara* und platzieren Sie ihn auf einer Welt mit 10 x 10 Feldern (siehe Abb.).

Der Käfer *kara* soll nun neun Schritte gehen, so dass er am Ende auf der anderen Seite steht.



1 Überlegen Sie:

- Wie lösen Sie das Problem mit den bisher bekannten Mitteln?
- Welchen Nachteil hat diese Vorgehensweise, wenn die Kara-Welt 100 x 100 Felder hätte?
- Entwickeln Sie einen geeigneten Vorschlag zur Lösung des Problems.

Hinweis: Beachten Sie zur Bearbeitung der folgenden Aufgabenstellung das Informationsmaterial *L1\_2 Information for-Schleife.docx*.

2 Erstellen Sie ein Struktogramm zur Lösung des beschriebenen Problems und kodieren Sie die Lösung.

Speichern Sie Ihre Ergebnisse im Ordner *meineErgebnisse* unter den Namen

*L1\_2\_1\_for\_Schleife.stg* (Struktogramm)

*L1\_2\_1\_for\_Schleife.world* und *L1\_2\_1\_for\_Schleife.py* (Programm).

## L1\_2 Die Zählerschleife (for-Schleife)

Beim Programmieren kommt es immer wieder vor, dass ein oder mehrere bestimmte Schritte mehrfach hintereinander ausgeführt, also wiederholt werden müssen. Das würde z.B. bei 1000 oder 100.000 nötigen Wiederholungen zu einer endlosen Kette identischer Anweisungen führen bzw. wäre vielleicht auch gar nicht mit vertretbarem Aufwand umsetzbar.

Zur Arbeitserleichterung gibt es daher für solche Wiederholungen spezielle Anweisungen. Statt z.B. 10 oder 20 Mal hintereinander den gleichen Befehl zu programmieren, kann man angeben:

„Führe diese Anweisung(en) 14 Mal aus“.

Erreicht wird das mit einer so genannten „Zählerschleife“ oder auch „for-Schleife“.

### 1 Aufbau einer Zählerschleife

Eine Zählerschleife benötigt vier Elemente:

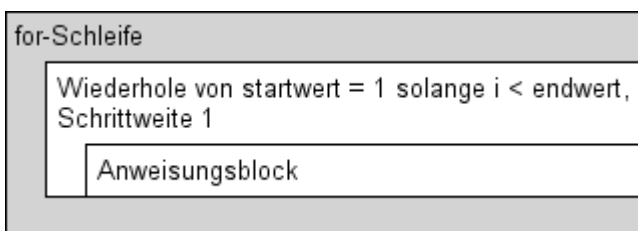
1. Eine Zählervariable, mit der die Wiederholungen gezählt werden.
2. Mit welcher Zahl soll die Zählung beginnen?
3. Bis wohin soll gezählt werden?
4. In welcher Schrittweite soll gezählt werden?

Für 14 Wiederholungen könnte z.B. mit der Zahl 1 begonnen und bis zur Zahl 14 gezählt werden. Bei jeder Wiederholung soll die Zahl um 1 erhöht werden.

Diese Informationen stehen im so genannten „**Schleifenkopf**“.

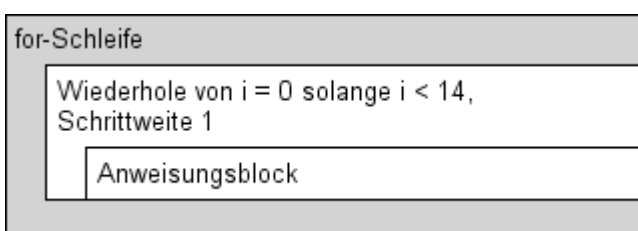
Im „**Schleifenkörper**“, **der direkt auf den Schleifenkopf folgt**, stehen genau die Anweisungen, die bei jedem Schleifendurchlauf ausgeführt werden sollen, in unserem Beispiel z.B. "Kara, gehe einen Schritt vorwärts" → `move()`.

### 2 Struktogramm einer Zählerschleife



#### Beispiel:

Für eine 14-malige Wiederholung von Programmanweisungen muss die Schleifenbedingung folgendermaßen formuliert werden:



J1	<b>BPE 5: Grundlagen der Programmierung</b> Informationsmaterial	Informatik
----	---	------------

### 3 Syntax einer Zählerschleife in PythonKara

Für die Kodierung von Zählerschleifen wird in PythonKara die *for-Schleife* verwendet.

**for Zählervariable in range(Anfangswert, Endwert [, Schrittweite]):**

**Anweisung[en], die wiederholt ausgeführt werden sollen.**

Beachte: Die Anweisungen, die wiederholt ausgeführt werden sollen, müssen eingerückt eingegeben werden.

Erfolgt keine Angabe zur Schrittweite, wird die Zählervariable jeweils um 1 erhöht.

Sollen nach der Zählerschleife weitere Anweisungen folgen, müssen diese wieder am Zeilenanfang beginnen.

**Beispiel:**            **for i in range(0, 14):**  
                          **kara.move()**

**Schleifenkopf:**    **for i in range(0, 14):**

- Im Schleifenkopf wird die Zählervariable vereinbart (i) und erhält einen Startwert (0).
- Nach jedem Durchlauf des Schleifenkörpers wird die Zählervariable um 1 erhöht.
- Die Zählervariable wird solange hochgezählt, bis sie die Zahl vor dem Endwert erreicht hat. Im vorliegenden Beispiel also solange, bis sie den Wert 13 erreicht hat.

**Schleifenkörper:** **kara.move()**

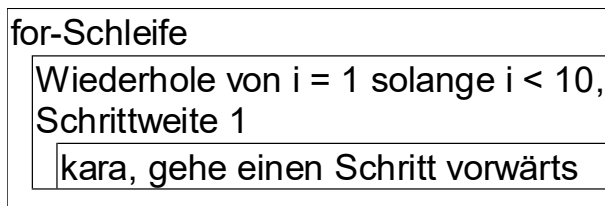
Im Schleifenkörper steht die Anweisung, die bei jeder Wiederholung ausgeführt werden soll. Hier ist es die Anweisung, dass der Käfer einen Schritt vorwärts gehen soll, also einfach *kara.move()*.

J1	<b>BPE 5: Grundlagen der Programmierung</b> Lösung	Informatik
----	---	------------

## L1\_2.1 Die Zählerschleife (for-Schleife)

- 1
  - Das Programm enthält 9 Mal den Befehl `kara.move()`.
  - Das Programm müsste 99 `kara.move()`-Befehle enthalten.
  - Anwendung einer Kontrollstruktur zur wiederholten Ausführung des `kara.move()`-Befehls.

### 2.1 Struktogramm



(L1\_2\_1\_for\_Schleife.stg)

### 2.2 Programmcode

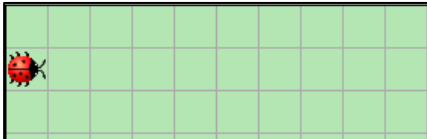
```
10 for i in range(9):  
11     kara.move()
```

(L1\_2\_1\_for\_Schleife.py)

## L1\_2.2 Übungsaufgaben zur for-Schleife – Teil 1

Starten Sie die Programmierumgebung PythonKara.

### Aufgabe 1



Erzeugen Sie einen neuen Käfer *kara* und platzieren Sie ihn auf einer Welt mit 10 x 10 Feldern (siehe Abb.).

*Kara* soll neun Schritte machen und dabei jedes Mal ein Blatt ablegen, so dass eine Art „Blattweg“ vom linken zum rechten Spielfeldrand entsteht.



Erstellen Sie ein Struktogramm zur Lösung des beschriebenen Problems und kodieren Sie die Lösung.

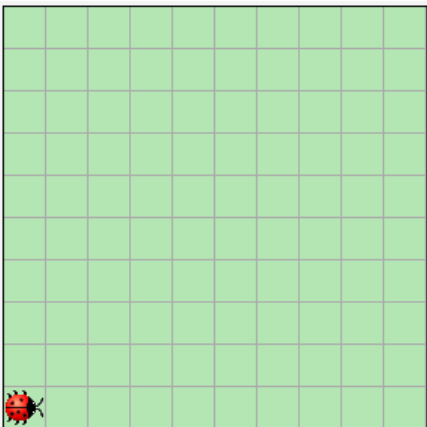
Speichern Sie Ihre Ergebnisse im Ordner *meineErgebnisse* unter den Namen

*L1\_2\_1\_A1\_for\_Schleife.stg* (Struktogramm)

*L1\_2\_2\_A1\_for\_Schleife.world* und

*L1\_2\_2\_A1\_for\_Schleife.py* (Programm).

### Aufgabe 2



Erzeugen Sie einen neuen Käfer *kara* und platzieren Sie ihn auf einer Welt mit 10 x 10 Feldern (siehe Abb.).

*Kara* soll auf dem Spielfeld die Diagonale von links unten nach rechts oben mit Blättern auslegen. In der Startposition steht *kara* in der unteren linken Ecke. In der Endposition soll kein Blatt abgelegt werden.

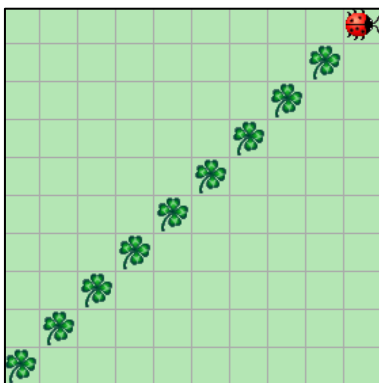
Erstellen Sie ein Struktogramm zur Lösung des beschriebenen Problems und kodieren Sie die Lösung.

Speichern Sie Ihre Ergebnisse im Ordner *meineErgebnisse* unter den Namen

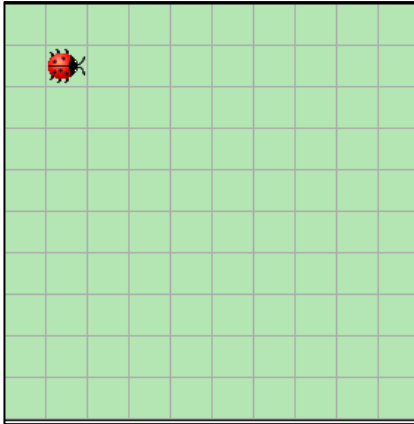
*L1\_2\_1\_A2\_for\_Schleife.stg* (Struktogramm)

*L1\_2\_2\_A2\_for\_Schleife.world* und

*L1\_2\_2\_A2\_for\_Schleife.py* (Programm).



### Aufgabe 3



Erzeugen Sie einen neuen Käfer *kara* und platzieren Sie ihn auf einer Welt mit 10 x 10 Feldern (siehe Abb.).

*Kara* soll mit Blättern ein "L" schreiben, wobei der lange Balken des "L" 7 Felder und der kurze Balken 4 Felder lang sein soll. Die Aufgabe soll mit 2 aufeinander folgenden Schleifen gelöst werden.

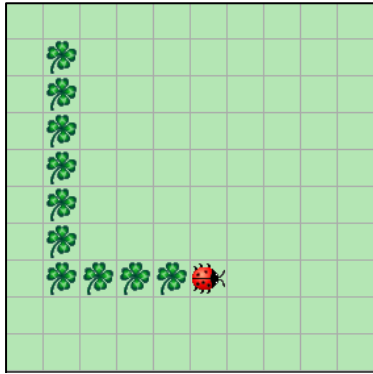
Erstellen Sie ein Struktogramm zur Lösung des beschriebenen Problems und kodieren Sie die Lösung.

Speichern Sie Ihre Ergebnisse im Ordner *meineErgebnisse* unter den Namen

*L1\_2\_1\_A3\_for\_Schleife.stg* (Struktogramm)

*L1\_2\_2\_A3\_for\_Schleife.world* und

*L1\_2\_2\_A3\_for\_Schleife.py* (Programm).



### Aufgabe 4

```

10 for zaehler in range(1, 6):
11     kara.turnLeft()
12     kara.move()
13     kara.turnLeft()
14     kara.move()
15     kara.turnLeft()
16     kara.move()
17     kara.turnLeft()
18     kara.move

```

In einer Kara-Welt wurde der abgebildete Programmcode entwickelt.

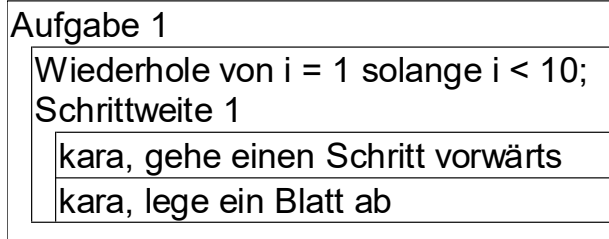
- 4.1 Erläutern Sie die syntaktischen Fehler, die dieser Programmcode enthält.
- 4.2 Beschreiben Sie die Wirkungsweise des Programmcodes, nachdem die syntaktischen Fehler korrigiert wurden.
- 4.3 Wie lässt sich der korrigierte Quellcode optimieren (weniger Zeilen Code)?



## L1\_2.2 Übungsaufgaben zur for-Schleife – Teil 1

### Aufgabe 1

#### Struktogramm



(L1\_2\_2\_A1\_for\_Schleife.stg)

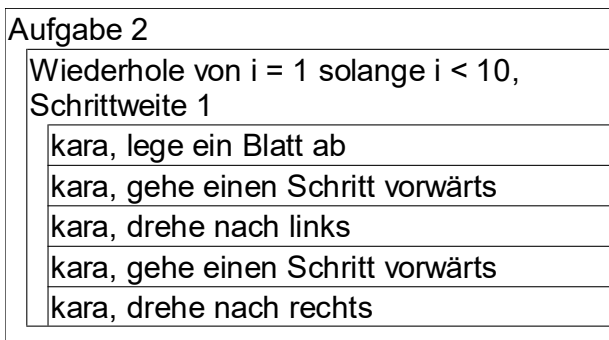
#### Programmcode

```
10 for i in range(1, 10):
11     kara.move()
12     kara.putLeaf()
```

(L1\_2\_2\_A1\_for\_Schleife.py)

### Aufgabe 2

#### Struktogramm



(L1\_2\_2\_A2\_for\_Schleife.stg)

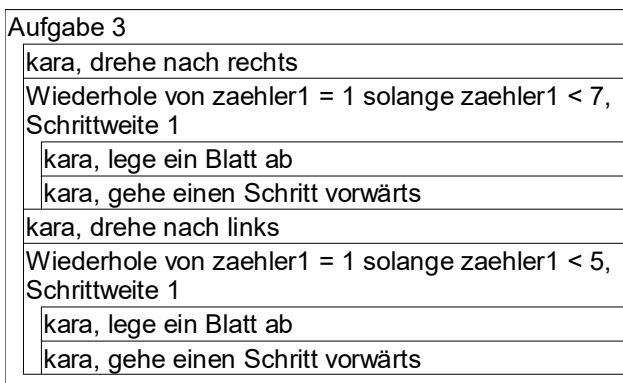
#### Programmcode

```
10 for i in range(1, 10):
11     kara.putLeaf()
12     kara.move()
13     kara.turnLeft()
14     kara.move()
15     kara.turnRight()
```

(L1\_2\_2\_A2\_for\_Schleife.py)

### Aufgabe 3

#### Struktogramm



(L1\_2\_2\_A3\_for\_Schleife.stg)

#### Programmcode

```
10 kara.turnRight()
11 for zaehler1 in range(1, 7):
12     kara.putLeaf()
13     kara.move()
14 kara.turnLeft()
15 for zaehler2 in range(1, 5):
16     kara.putLeaf()
17     kara.move()
```

(L1\_2\_2\_A3\_for\_Schleife.py)

J1	<b>BPE 5: Grundlagen der Programmierung</b> Lösung	Informatik
----	---	------------

## Aufgabe 4

---

- 4.1 - Die Anweisungen des Schleifenkörpers sind nicht eingerückt.  
- Die letzte Anweisung im Schleifenkörper muss lauten: *kara.move()*.

4.2 Der Käfer *kara* läuft fünf Mal im Kreis.

4.3  

```
for zaehler in range(1,21):  
    kara.turnLeft()  
    kara.move()
```

Hinweis: Anstatt 8 Zeilen Code 5 mal zu wiederholen, kann man auch 2 Zeilen Code 20 mal wiederholen.

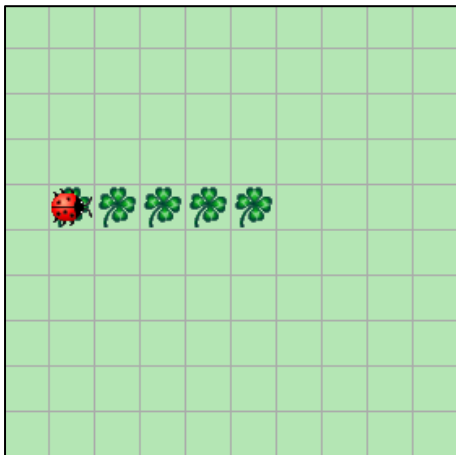
J1	<b>BPE 5: Grundlagen der Programmierung</b> Arbeitsauftrag	Informatik
----	---	------------

## L1\_3.1 Die while-Schleife

### (I) Problemstellung

---

Starten Sie die Programmierumgebung PythonKara und platzieren Sie eine Reihe mit fünf Kleeblättern und einen Käfer in einer Welt mit 10 x 10 Feldern (siehe Abb.).



Der Käfer *Kara* steht auf dem ersten Kleeblatt und soll so lange vorwärts gehen, bis er nicht mehr auf einem Blatt steht.

1 Überlegen Sie:

- Wie können Sie das Problem mithilfe der bisher kennengelernten Programmstrukturen lösen?
- Funktioniert Ihre Lösung – ohne Veränderungen am Programm vorzunehmen – auch dann, wenn der Käfer *kara* 3 (oder 5, oder 8 oder 10) Kleeblätter vor sich hat?
- Wie könnte ein Programm aussehen bzw. was müsste eine Programmstruktur können, damit Sie das Problem lösen können?

Hinweis: Beachten Sie zur Bearbeitung der folgenden Aufgabenstellung das Informationsmaterial *L1\_3 Information while-Schleife.docx*.

2 Erstellen Sie ein Struktogramm zur Lösung des beschriebenen Problems und kodieren Sie die Lösung.

Speichern Sie Ihre Ergebnisse im Ordner *meineErgebnisse* unter den Namen

*L1\_3\_1\_while\_Schleife.stg* (Struktogramm)

*L1\_3\_1\_while\_Schleife.world* und *L1\_3\_1\_while\_Schleife.py* (Programm)

## L1\_3 Die while-Schleife

Die „while“-Schleife löst Problemstellungen, bei denen ein Vorgang solange wiederholt wird, bis eine Bedingung nicht mehr erfüllt ist.

Ein Beispiel wäre die Problemstellung: „Erhöhe den Wert der Variablen *anzahl* um 1 und gebe sie aus, solange sie den Wert 10 nicht erreicht hat“.

### 1 Vergleichsoperatoren

Vergleichsoperatoren werden immer zum Vergleich zweier Werte benutzt und finden ihre Anwendung zumeist beim Einsatz von Wiederholungen oder Alternativen. Als Ergebnis des Vergleichs wird ein boolescher Wert (true / false) zurückgegeben. Meist wird eine Variable mit einem festen Wert oder eine Variable mit einer anderen Variablen verglichen. Je nach Ergebnis des Vergleichs wird das Programm eine andere Reaktion zeigen.

Operator	Beispiel	Beschreibung
<	a < b	a ist kleiner als b
<=	a <= b	a ist kleiner oder gleich b
>	a > b	a ist größer als b
>=	a >= b	a ist größer oder gleich b
==	a == b	a ist gleich b
!=	a != b	a ist ungleich b

### 2 Logische Operatoren in Python

Der Käfer *kara* kann durch die Nutzung seiner Sensoren auf Ereignisse reagieren. Dem Käfer *kara* ist es auch möglich, auf mehrere Sensoren gleichzeitig zu reagieren. Hier eine kurze Übersicht über die Verwendung von Operatoren.

Operator	Beispiel	Beschreibung	Erläuterung
and	kara.treeFront() <b>and</b> kara.treeRight()	und	Die Bedingung ist erfüllt (true), wenn beide Aussagen zutreffen, d.h. wenn der Käfer <i>kara</i> einen Baum vor und rechts von sich hat.
or	kara.treeFront() <b>or</b> kara.treeRight()	oder	Die Bedingung ist erfüllt (true), wenn die eine <b>oder</b> die andere <b>oder</b> beide Aussage/n erfüllt sind.
not	<b>not</b> kara.treeFront()	nicht	Das Ausrufezeichen ändert einen Ausdruck von true in false und umgekehrt. Hier ist die Bedingung erfüllt, wenn der Käfer <i>kara</i> <b>nicht</b> vor einem Baum steht.

### 3 Aufbau einer while-Schleife

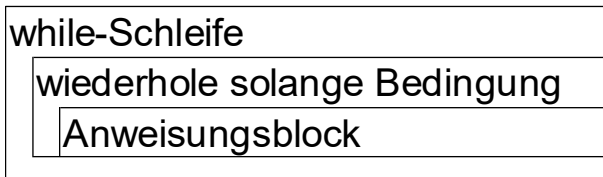
Die while-Schleife benötigt zuerst eine Bedingung, die dafür sorgt, dass die Schleife durchlaufen wird oder nicht. Diese wird im Schleifenkopf hinter dem Schlüsselwort *while* in Klammern angegeben.

Beispiel: `while anzahl < 10`

Danach folgt der Schleifenkörper. Hier stehen die Anweisungen, die abgearbeitet werden sollen, falls es zu einem Schleifendurchlauf kommt.

Im Schleifenkörper stehen die Anweisungen, die bei jedem Schleifendurchlauf ausgeführt werden.

### 4 Das Struktogramm einer while-Schleife



Wiederholungsstruktur mit vorausgehender Bedingungsprüfung. Der Schleifenkörper (Anweisungsblock) wird nur durchlaufen, solange die Bedingung erfüllt (wahr) ist.

### 5 Syntax einer while-Schleife

```
10 while anzahl < 10:  
11     kara.move()  
12     anzahl = anzahl +1
```

Beachte: Die Anweisungen, die wiederholt ausgeführt werden sollen, müssen eingerückt eingegeben werden.

Im **Schleifenkopf** wird die Bedingung festgelegt.

Der **Schleifenkörper** enthält die Anweisungen, die bei jeder Wiederholung ausgeführt werden.

Im dargestellten Beispiel wird im Schleifenkopf geprüft, ob der Wert der Variablen *anzahl* < 10 ist.

Ist dies der Fall, wird der Schleifenkörper abgearbeitet:

- Der Käfer *kara* geht einen Schritt vorwärts.
- Der Wert der Variablen *anzahl* wird um 1 erhöht.

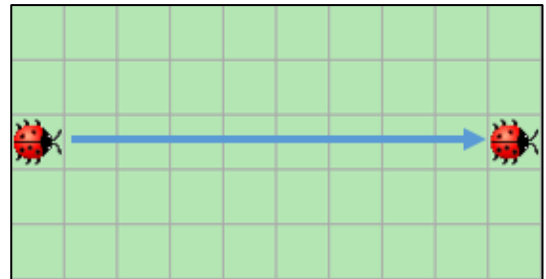
Danach wird wieder geprüft, ob der Wert der Variablen *anzahl* immer noch kleiner als 10 ist ...

## 6 Realisierung der Problemstellung

6.1 Mit der Schleifenbedingung *while* *anzahl* < 10 kann der Käfer *kara* vom linken Rand einer 10 x 10 Felder großen Welt an den rechten Rand bewegt werden.

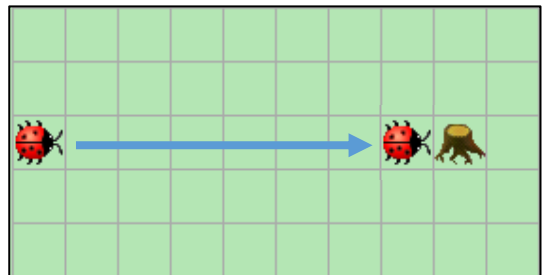
Damit die Bedingung *anzahl* < 10 einen Wert (true oder false) zurückliefern kann, benötigt die Variable *anzahl* einen Anfangswert. Dieser muss vor dem Schleifenbeginn festgelegt werden:

```
10 anzahl = 1
11 while anzahl < 10:
12     kara.move()
13     anzahl = anzahl + 1
```



6.2 Mit der Schleifenbedingung *while not kara.treeFront()* kann der Käfer *kara* so lange Anweisungen ausführen, bis er vor einem Baum steht.

```
10 while not kara.treeFront():
11     kara.move()
```



J1	<b>BPE 5: Grundlagen der Programmierung</b> Lösung	Informatik
----	---	------------

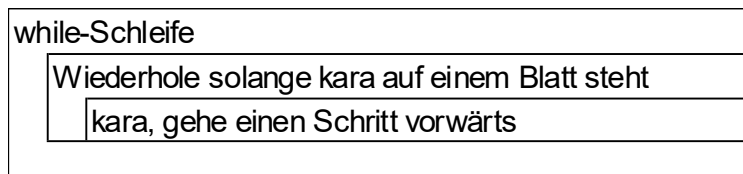
## L1\_3.1 Die while-Schleife

---

- 1
  - Anwendung einer for-Schleife, in der die Anweisung *gehe einen Schritt* 5 Mal wiederholt wird.
  - Da die Anzahl der Wiederholungen auf fünf festgelegt wird, kann die Lösung auch nur dann funktionieren, wenn der Käfer *kara* vier Blätter vor sich hat.
  - Anwendung einer Kontrollstruktur, bei der die wiederholte Ausführung des *move()*-Befehls davon abhängig ist, ob der Käfer *kara* auf einem Blatt steht.

### 2 Programm

#### 2.1 Struktogramm



(L1\_3\_1\_while\_Schleife.stg)

#### 2.2 Programmcode

```
10 while kara.onLeaf():  
11     kara.move()
```

(L1\_3\_1\_while\_Schleife.py)

## L1\_3.2 Übungsaufgaben zur while-Schleife

### Aufgabe 1



Ausgangssituation: Siehe nebenstehende Abbildung.

Der Käfer *kara* soll den Bäumen entlang gehen, bis rechts von ihm kein Baum mehr ist.

Verwenden Sie aus dem Ordner Aufgaben/Vorlagen die Welt *L1\_3\_2\_Aufgabe1\_Vorlage.world* als Vorlage.

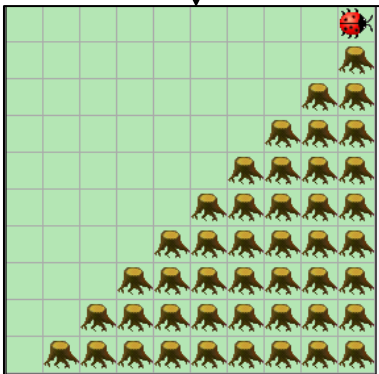
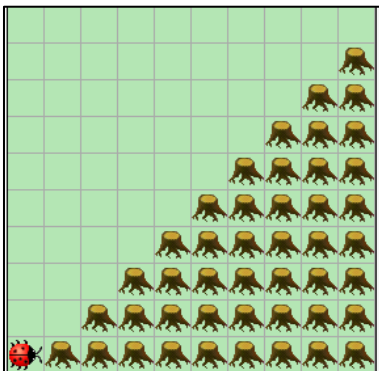
Speichern Sie Ihre Ergebnisse im Ordner *meineErgebnisse* unter den Namen

*L1\_3\_2\_A1\_while\_Schleife.stg* (Struktogramm)

*L1\_3\_2\_A1\_while\_Schleife.world* und

*L1\_3\_2\_A1\_while\_Schleife.py* (Programm).

### Aufgabe 2



Ausgangssituation: Siehe nebenstehende Abbildung.

Der Käfer *kara* soll die Treppe aus Bäumen hochlaufen.

Verwenden Sie aus dem Ordner Aufgaben/Vorlagen die Welt *L1\_3\_2\_Aufgabe2\_Vorlage.world* als Vorlage.

Speichern Sie Ihre Ergebnisse im Ordner *meineErgebnisse* unter den Namen

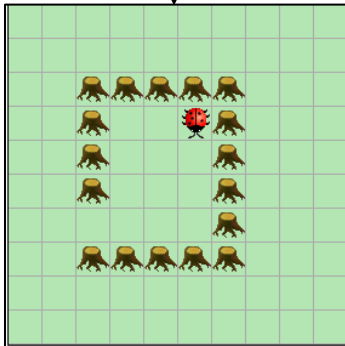
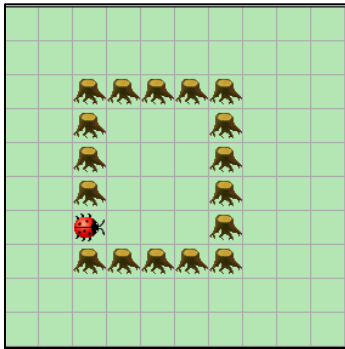
*L1\_3\_2\_A2\_while\_Schleife.stg* (Struktogramm)

*L1\_3\_2\_A2\_while\_Schleife.world* und

*L1\_3\_2\_A2\_while\_Schleife.py* (Programm).



### Aufgabe 3



Ausgangssituation: Siehe nebenstehende Abbildung.

Der Käfer *kara* steht im Eingang eines rechteckigen Kara-Baus. *Kara* soll in die obere rechte Ecke des Baus laufen und nach unten schauen. (siehe Abb.).

Beachten Sie, dass der Kara-Bau verschiedene Ausmaße haben kann.

Verwenden Sie aus dem Ordner Aufgaben/Vorlagen die Welt *L1\_3\_2\_Aufgabe3\_Vorlage.world* als Vorlage.

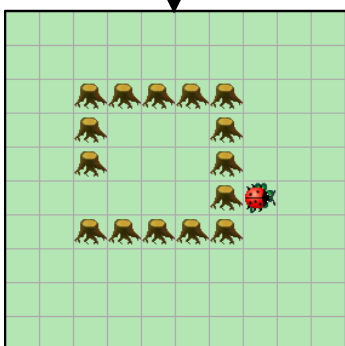
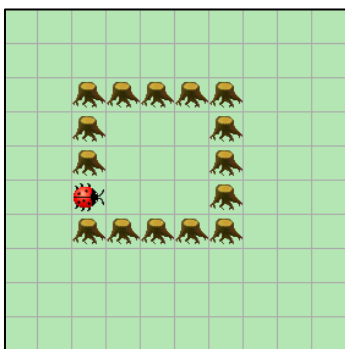
Speichern Sie Ihre Ergebnisse im Ordner *meineErgebnisse* unter den Namen

*L1\_3\_2\_A3\_while\_Schleife.stg* (Struktogramm)

*L1\_3\_2\_A3\_while\_Schleife.world* und

*L1\_3\_2\_A3\_while\_Schleife.py* (Programm).

### Aufgabe 4



Der Käfer *kara* steht im Eingang seines rechteckigen Baus. *Kara* soll im Uhrzeigersinn um den Bau laufen und an der gegenüberliegenden Seite ein Blatt ablegen. Der Käfer *kara* soll dort dieselbe Blickrichtung einnehmen wie in seiner Startposition.

Verwenden Sie aus dem Ordner Aufgaben/Vorlagen die Welt *L1\_3\_2\_Aufgabe4\_Vorlage.world* als Vorlage.

Speichern Sie Ihre Ergebnisse im Ordner *meineErgebnisse* unter den Namen

*L1\_3\_2\_A4\_while\_Schleife.stg* (Struktogramm)

*L1\_3\_2\_A4\_while\_Schleife.world* und

*L1\_3\_2\_A4\_while\_Schleife.py* (Programm).

J1	<b>BPE 5: Grundlagen der Programmierung</b> Arbeitsauftrag	Informatik
----	---	------------

## Aufgabe 5

---

```
10 zaehler = 0
11 while (zaehler < 10):
12     kara.move()
13     kara.turnLeft()
14     kara.move()
15     kara.turnLeft()
16     kara.move()
17     kara.turnLeft()
18     kara.move()
19     kara.turnLeft()
```

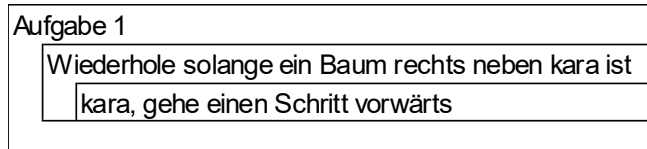
In einer Kara-Welt wurde der abgebildete Programmcode entwickelt.

- 5.1 Erläutern Sie den logischen Fehler, den dieser Programmcode enthält.
- 5.2 Beschreiben Sie die Wirkungsweise des Programmcodes, nachdem der Fehler korrigiert wurde.

## L1\_3.2 Übungsaufgaben zur while-Schleife

### Aufgabe 1

#### Struktogramm



(L1\_3\_2\_A1\_while\_Schleife.stg)

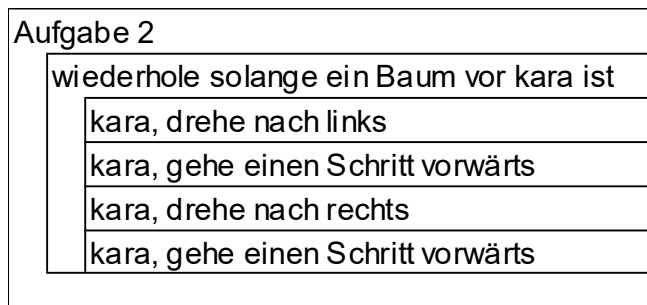
#### Programmcode

```
10 while kara.treeRight():
11     kara.move()
```

(L1\_3\_2\_A1\_while\_Schleife.py)

### Aufgabe 2

#### Struktogramm



(L1\_3\_2\_A2.stg)

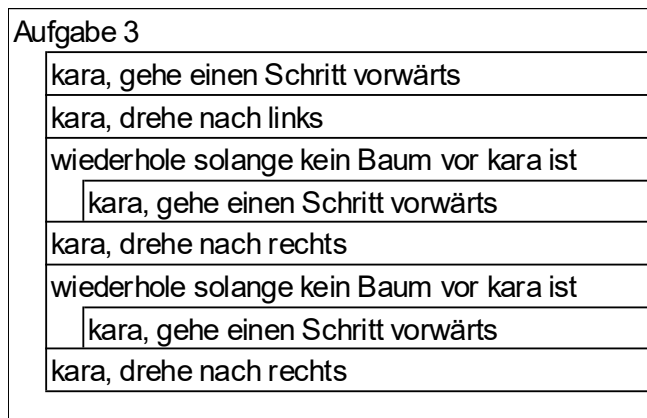
#### Programmcode

```
10 while kara.treeFront():
11     kara.turnLeft()
12     kara.move()
13     kara.turnRight()
14     kara.move()
```

(L1\_3\_2\_A2\_while\_Schleife.py)

### Aufgabe 3

#### Struktogramm



(L1\_3\_2\_A3\_while\_Schleife.stg)

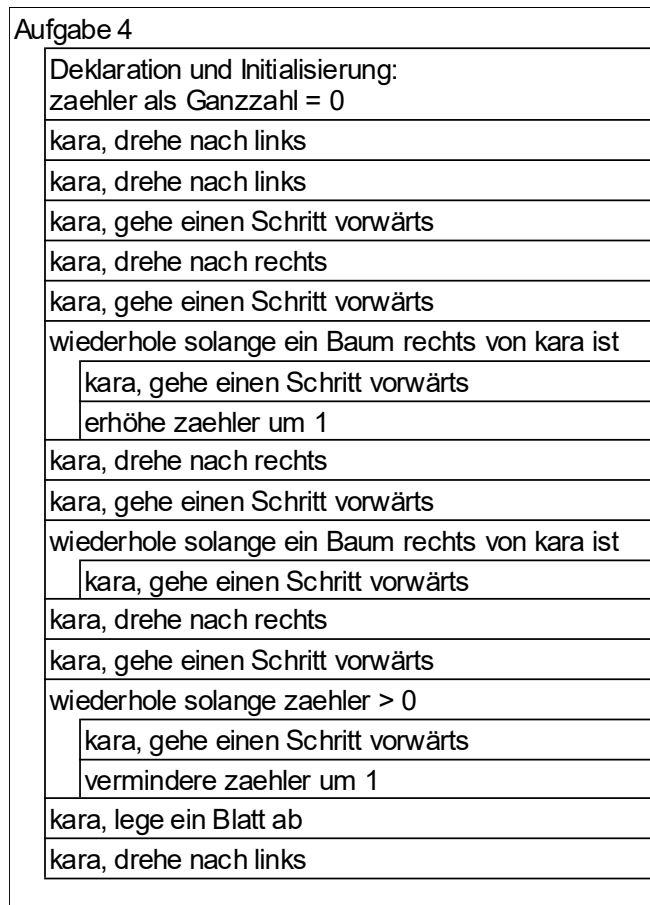
#### Programmcode

```
10 kara.move()
11 kara.turnLeft()
12 while not kara.treeFront():
13     kara.move()
14 kara.turnRight()
15 while not kara.treeFront():
16     kara.move()
17 kara.turnRight()
```

(L1\_3\_2\_A3\_while\_Schleife.py)

## Aufgabe 4

### Struktogramm



(L1\_3\_2\_A4\_while\_Schleife.stg)

### Programmcode

```

10 zaehler = 0
11 kara.turnLeft()
12 kara.turnLeft()
13 kara.move()
14 kara.turnRight()
15 kara.move()
16 while kara.treeRight():
17     kara.move()
18     zaehler = zaehler + 1
19 kara.turnRight()
20 kara.move()
21 while kara.treeRight():
22     kara.move()
23 kara.turnRight()
24 kara.move()
25 while (zaehler > 0):
26     kara.move()
27     zaehler = zaehler - 1
28 kara.putLeaf()
29 kara.turnLeft()

```

(L1\_3\_2\_A4\_while\_Schleife.py)

## Aufgabe 5

5.1 Die Schleifenbedingung besagt, dass die Anweisungen im while-Block wiederholt werden sollen, solange die Zählervariable *zaehler* den Wert 10 nicht erreicht hat. Da die Zählervariable im Programmablauf nie erhöht wird, werden die Anweisungen im while-Block unendliche Male wiederholt (Endlosschleife).

➔ Am Ende des Schleifenkörpers muss folgende Anweisung eingefügt werden:

zaehler = zaehler + 1

5.2 Der Käfer *kara* läuft zehn Mal im Kreis.

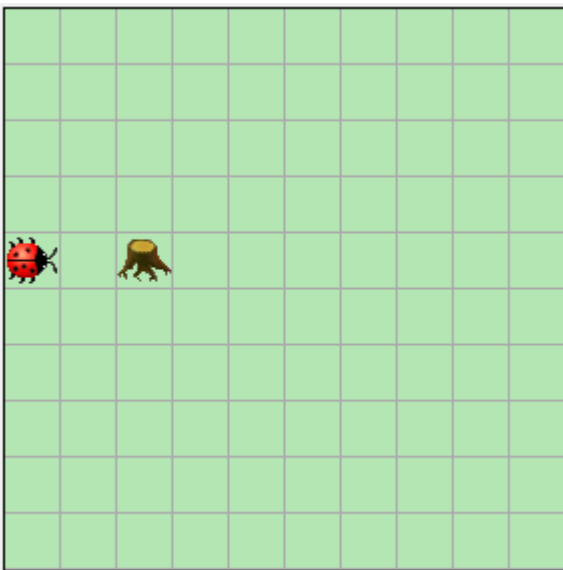
J1	<b>BPE 5: Grundlagen der Programmierung</b> Arbeitsauftrag	Informatik
----	---	------------

## L1\_4.1 Die Alternative (if-Anweisung)

### (I) Problemstellung

---

Starten Sie die Programmierumgebung PythonKara und platzieren Sie – wie abgebildet – einen Käfer und einen Baum in einer Welt mit 10 x 10 Feldern (siehe Abb.).



Der Käfer *kara* soll einen Schritt machen und dann prüfen, ob ein Baum im Weg steht.

Ist dies der Fall, soll der Käfer *kara* den Baum rechts umgehen. Hinter dem Baum soll *kara* wieder zum Stehen kommen und in die gleiche Richtung schauen wie in der Startposition.

Ist dies nicht der Fall, soll der Käfer *kara* nichts tun.

1 Überlegen Sie:

- Welche unterschiedlichen Aktionen sind zu beachten?
- In welcher Reihenfolge soll der Käfer *kara* die Aktionen durchführen?

Hinweis: Beachten Sie zur Bearbeitung der folgenden Aufgabenstellung das Informationsmaterial *L1\_4 Information Alternative.docx*.

2 Erstellen Sie ein Struktogramm zur Lösung des beschriebenen Problems und kodieren Sie die Lösung.

Speichern Sie Ihre Ergebnisse im Ordner *meineErgebnisse* unter den Namen

*L1\_4\_1\_Alternative.stg* (Struktogramm)

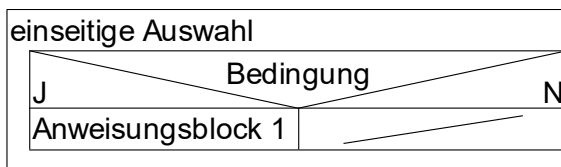
*L1\_4\_1\_Alternative.world* und *L1\_4\_1\_Alternative.py* (Programm)

## L1\_4 Die Alternative

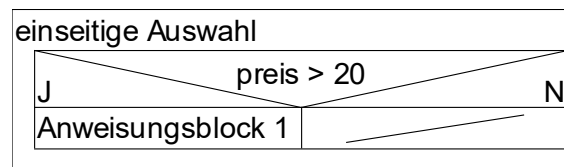
Häufig soll ein Programm etwas ausführen, das von einer aktuellen Situation abhängig ist. Beispielsweise könnte der Käfer *kara* prüfen, ob er auf einem Blatt steht und dieses aufheben, falls er tatsächlich auf einem Blatt steht, oder einen Schritt vorwärts gehen, wenn er nicht auf einem Blatt steht.

Dieses Vorgehen wird als „Fallunterscheidung“ bezeichnet: In dem einen „Fall“ soll das Programm etwas anderes machen als in einem anderen „Fall“. Bekannt ist die Situation vom Einsatz der Wenn-Funktion in der Tabellenkalkulation.

### 1.1 Die einseitige Auswahl (bedingte Verarbeitung)



(L1\_4\_1a\_einseitige\_Auswahl.stg)

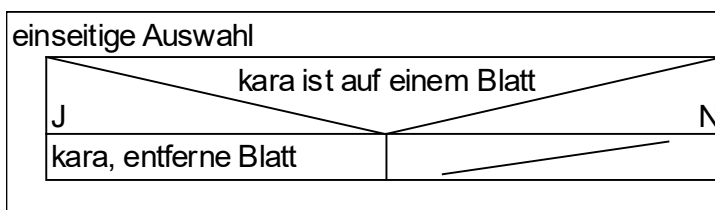


(L1\_4\_1b\_einseitige\_Auswahl.stg)

Nur wenn die Bedingung zutreffend (wahr) ist, wird der Anweisungsblock 1 durchlaufen. Ein Anweisungsblock kann aus einer oder mehreren Anweisungen bestehen. Trifft die Bedingung nicht zu (falsch), wird der Durchlauf ohne eine weitere Anweisung fortgeführt (Austritt unten).

### 1.2 Syntax einer einseitigen Auswahl

In der Bedingung wird unterstellt, dass nach der positiven Erfüllung gesucht wird. Auf die Angabe „ist wahr“ wird verzichtet.



(L1\_4\_1a\_einseitige\_Auswahl.stg)

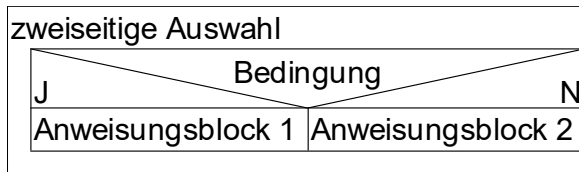
```
10 if kara.onLeaf():
11     kara.removeLeaf()
```

Im dargestellten Beispiel wird die Aktion *removeLeaf()* ausgeführt, wenn die Bedingung *onLeaf()* eintritt. Ansonsten passiert nichts.

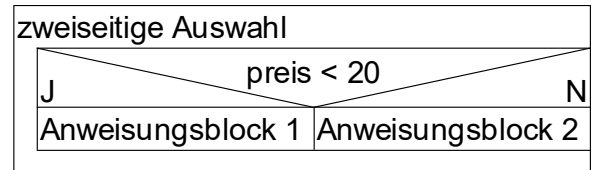
Das heißt, wenn der Käfer *kara* auf einem Blatt steht, liefert der Sensor *onLeaf()* den Wert *wahr* zurück. Tritt dieser Fall ein, bewirkt die Aktion *removeLeaf()*, dass er das Blatt entfernt.

In Python wird eine Fallunterscheidung mit der Anweisung *if - else* umgesetzt. Dabei wird die *if*-Bedingung mit einem Doppelpunkt abgeschlossen und in der/den Folgezeile/n die Anweisungen für den Fall, dass die Bedingung erfüllt ist, eingerückt notiert.

## 2.1 Die zweiseitige Auswahl (alternative Verarbeitung)



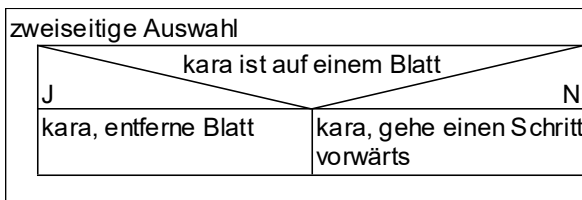
(L1\_4\_2a\_einseitige\_Auswahl.stg)



(L1\_4\_2b\_einseitige\_Auswahl.stg)

Wenn die Bedingung zutreffend (wahr) ist, wird der Anweisungsblock 1 durchlaufen. Trifft die Bedingung nicht zu (falsch), wird der Anweisungsblock 2 durchlaufen. Ein Anweisungsblock kann aus einer oder mehreren Anweisungen bestehen. Der Verzweigungsblock wird nach der Ausführung des jeweiligen Anweisungsblocks verlassen

## 2.2 Syntax einer zweiseitigen Auswahl



(L1\_4\_2c\_einseitige\_Auswahl.stg)

```

10 if kara.onLeaf():
11     kara.removeLeaf()
12 else:
13     kara.move()

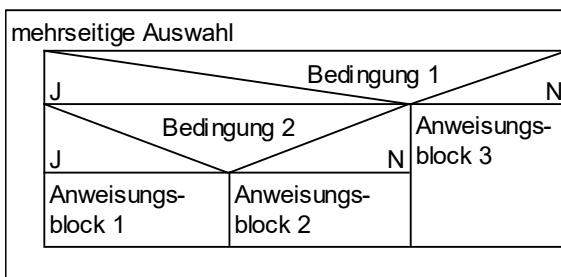
```

Im dargestellten Beispiel wird die Aktion *removeLeaf()* ausgeführt, wenn die Bedingung eintritt. Ansonsten geht der Käfer *kara* einen Schritt - *move()* - vorwärts.

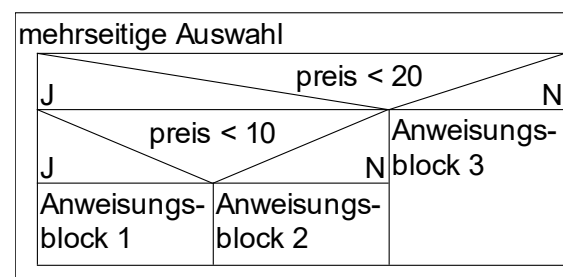
Das heißt, wenn der Käfer *kara* auf einem Blatt steht, liefert der Sensor *onLeaf()* den Wert *wahr* zurück. Tritt dieser Fall ein, bewirkt die Aktion *removeLeaf()*, dass er das Blatt entfernt. Wenn der Käfer *kara* auf keinem Blatt steht, geht er einen Schritt vorwärts.

In Python werden die Anweisungen für den Nein-Fall der if-Bedingung mit dem Schlüsselwort *else:* eingeleitet. Die Anweisung für den Fall, dass die Bedingung nicht erfüllt ist, werden in der/den Folgezeile/n eingerückt notiert.

## 3.1 Die mehrseitige Auswahl



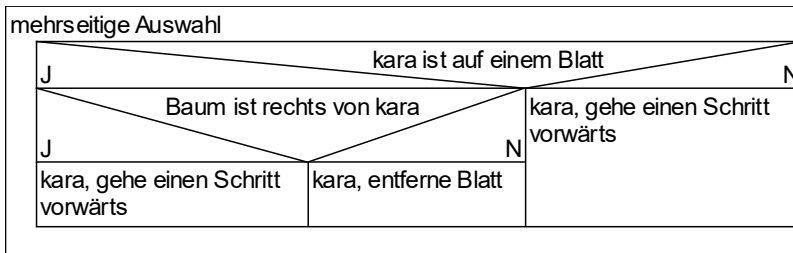
(L1\_4\_3a\_einseitige\_Auswahl.stg)



(L1\_4\_3b\_einseitige\_Auswahl.stg)

Die mehrseitige Auswahl wird auch „verschachtelte“ Auswahl genannt, da eine weitere Bedingung folgt. Die Verschachtelung ist ebenso im Nein-Fall möglich.

### 3.2 Syntax einer mehrseitigen Auswahl



(L1\_4\_3c\_einseitige\_Auswahl.stg)

```

10 if kara.onLeaf():
11     if kara.treeRight():
12         kara.move()
13     else:
14         kara.removeLeaf()
15 else:
16     kara.move()

```

Im dargestellten Beispiel wird die Aktion *move()* ausgeführt, wenn beide Bedingungen eingetreten sind. Ist nur die erste, nicht aber die zweite Bedingung erfüllt, wird die Aktion *removeLeaf()* ausgeführt. Wenn die erste Bedingung nicht erfüllt ist, wird die Aktion *move()* ausgeführt.

Das heißt, wenn der Käfer *kara* auf einem Blatt steht, liefert der Sensor *onLeaf()* den Wert *wahr* zurück. Ist dies der Fall, wird zusätzlich geprüft, ob ein Baum rechts von ihm steht. Trifft auch das zu, liefert der Sensor *treeRight()* den Wert *wahr* zurück. Die Aktion *move()* bewirkt, dass er einen Schritt geradeaus geht.

Liefert der Sensor *treeRight()* den Wert *false*, bewirkt die Aktion *removeLeaf()*, dass der Käfer *kara* das Blatt entfernt.

Wenn der Sensor *onLeaf()* den Wert *false* zurückgibt, führt die Aktion *move()* dazu, dass der Käfer *kara* einen Schritt vorwärts geht.

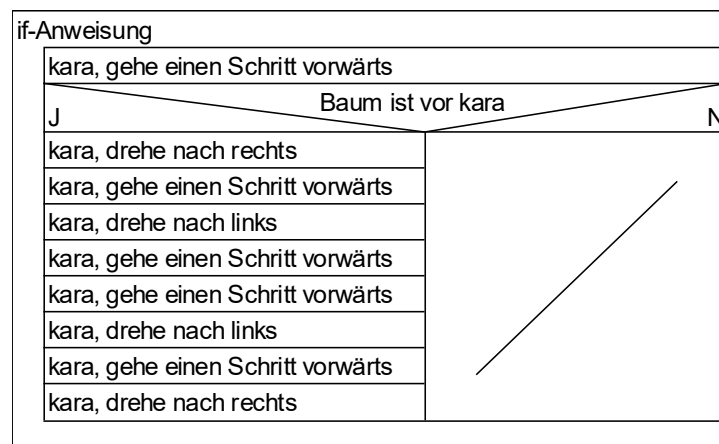


## L1\_4.1 Die Alternative (if-Anweisung)

- 1
  - Der Käfer *kara* muss
    - ♦ geradeaus gehen,
    - ♦ nach links drehen,
    - ♦ nach rechts drehen,
    - ♦ prüfen, ob ein Baum im Weg steht.
  - Die Aktionen müssen in folgender Reihenfolge durchgeführt werden:
    - ♦ einen Schritt gehen
    - ♦ prüfen, ob ein Baum im Weg steht
    - falls ja:
      - nach rechts drehen
      - einen Schritt vorwärts gehen
      - nach links drehen
      - zwei Schritte vorwärts gehen
      - nach links drehen
      - einen Schritt vorwärts gehen
      - nach rechts drehen
    - falls nein:
      -

### 2 Programm

#### 2.1 Struktogramm



(L1\_4\_1\_Alternative.stg)

#### 2.1 Programmcode

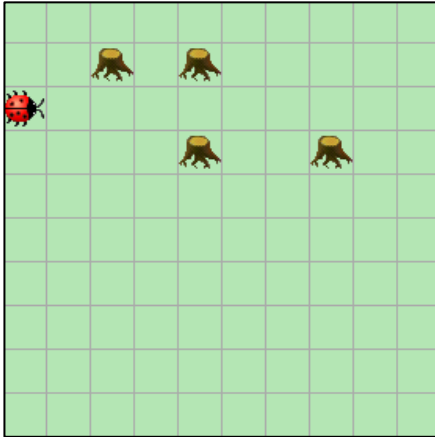
```

10 kara.move ()
11 if kara.treeFront () :
12     kara.turnRight ()
13     kara.move ()
14     kara.turnLeft ()
15     kara.move ()
16     kara.move ()
17     kara.turnLeft ()
18     kara.move ()
19     kara.turnRight ()
  
```

(L1\_4\_1\_Alternative.py)

## L1\_5.1 Übungsaufgaben zu Kontrollstrukturen

### Aufgabe 1



Der Käfer *kara* soll neun Schritte über eine Welt mit 10 x 10 Feldern laufen und jedes Mal prüfen, ob links oder rechts von ihm ein Baum steht. Ist dies der Fall, soll er ein Blatt ablegen. Ist dies nicht der Fall, soll er ohne etwas zu tun einen Schritt vorwärts gehen.



Verwenden Sie aus dem Ordner Aufgaben/Vorlagen die Welt *L1\_5\_1\_Aufgabe1\_Vorlage.world* als Vorlage.

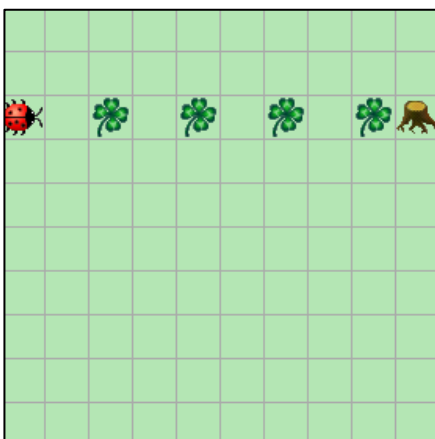
Erstellen Sie ein Struktogramm zur Lösung des beschriebenen Problems und kodieren Sie die Lösung. Testen Sie Ihr Ergebnis, indem Sie die Standorte der Bäume variieren.

Speichern Sie Ihre Ergebnisse im Ordner *meineErgebnisse* unter den Namen

*L1\_5\_1\_A1\_Uebung\_Kontrollstruktur.stg* (Struktogramm)

*L1\_5\_1\_A1\_Uebung\_Kontrollstruktur.world* und *L1\_5\_1\_A1\_Uebung\_Kontrollstruktur.py* (Programm).

### Aufgabe 2



Der Käfer *kara* soll über eine Welt mit 10 x 10 Feldern laufen bis er vor einem Baum steht. Nach jedem Schritt soll er prüfen, ob er auf einem Blatt steht. Ist das nicht der Fall, soll er ein Blatt ablegen. Wenn er auf einem Blatt steht, soll er das Blatt aufheben.



Verwenden Sie aus dem Ordner Aufgaben/Vorlagen die Welt *L1\_5\_1\_Aufgabe2\_Vorlage.world* als Vorlage.

Erstellen Sie ein Struktogramm zur Lösung des beschriebenen Problems und kodieren Sie die Lösung. Testen Sie Ihr Ergebnis, indem Sie die Standorte der Akteure variieren.

Speichern Sie Ihre Ergebnisse im Ordner *meineErgebnisse* unter den Namen

*L1\_5\_1\_A2\_Uebung\_Kontrollstruktur.stg* (Struktogramm)

J1	<b>BPE 5: Grundlagen der Programmierung</b> Arbeitsauftrag	Informatik
----	---	------------

*L1\_5\_1\_A2\_Uebung\_Kontrollstruktur.world* und *L1\_5\_1\_A2\_Uebung\_Kontrollstruktur.py* (Programm).

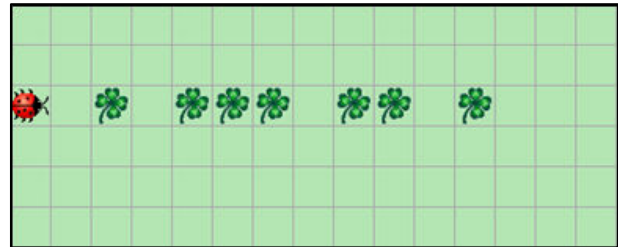
### Aufgabe 3

In einer Kara-Welt wurde folgender Programmcode erfasst.

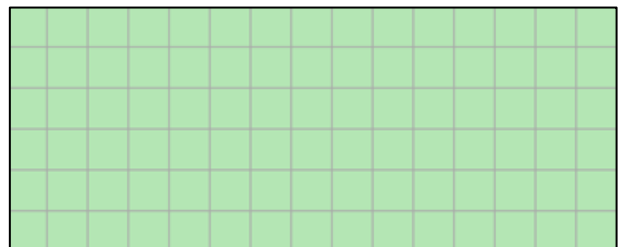
```

10 zaehler = 0
11 for i in range(1, 15):
12     kara.move()
13     if kara.onLeaf():
14         kara.removeLeaf()
15         zaehler = zaehler + 1
16 kara.turnRight()
17 kara.turnRight()
18 for i in range(0, zaehler):
19     kara.putLeaf()
20     kara.move()

```



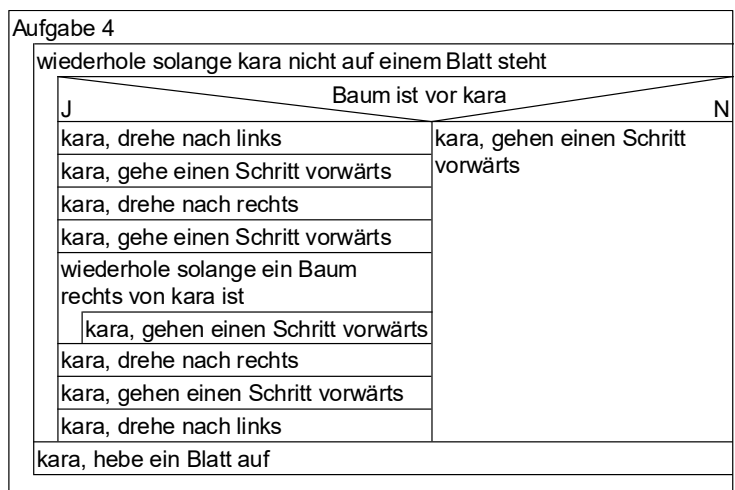
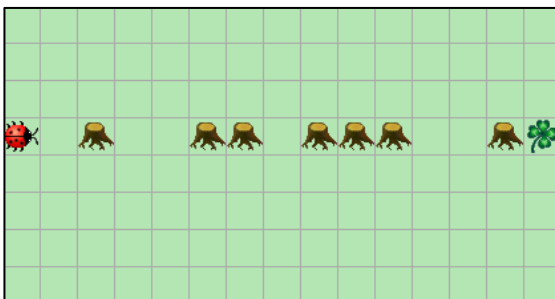
- 3.1 Beschreiben Sie die Wirkungsweise dieses Codes.
- 3.2 Skizzieren Sie das Ergebnis des Programmablaufs in der nebenstehenden Abbildung.
- 3.3 Nennen Sie den Wert, den die Variable *zaehler* nach Ablauf des Programms aufweist.



### Aufgabe 4

Für ein Programm in der Kara-Welt wurde folgendes Struktogramm entwickelt:

Beschreiben Sie die Wirkungsweise des daraus zu entwickelnden Programmcodes und skizzieren Sie in der folgenden Abbildung den Weg, den der Käfer *kara* geht.



J1	<b>BPE 5: Grundlagen der Programmierung</b> Arbeitsauftrag	Informatik
----	---	------------

## Aufgabe 5

Für ein Programm in der Kara-Welt wurde folgender Programmcode entwickelt.

Analysieren Sie den Programmablauf und erläutern Sie die syntaktischen und logischen Fehler.

```
5 while not kara.onTree():
6     if kara.onLeaf():
7         kara.putLeaf()
8         kara.move()
9     else:
10        kara.removeLeaf()
11        kara.move()
```

## Aufgabe 6

Für ein Programm in der Kara-Welt wurde der abgebildete Programmcode entwickelt.

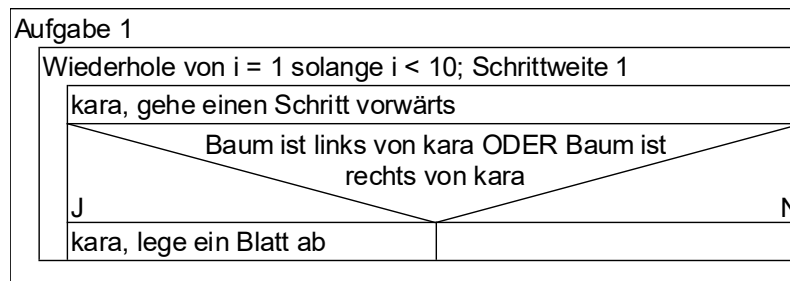
Analysieren Sie den Programmablauf und erläutern Sie die logischen Fehler.

```
10 while kara.onLeaf() and not kara.onLeaf():
11     kara.move()
12     if kara.onLeaf():
13         kara.putLeaf()
14     else:
15         kara.removeLeaf()
```

## L1\_5.1 Übungsaufgaben zu Kontrollstrukturen

### Aufgabe 1

#### Struktogramm



(L1\_5\_1\_A1\_Uebung\_Kontrollstruktur.stg)

#### Programmcode

```

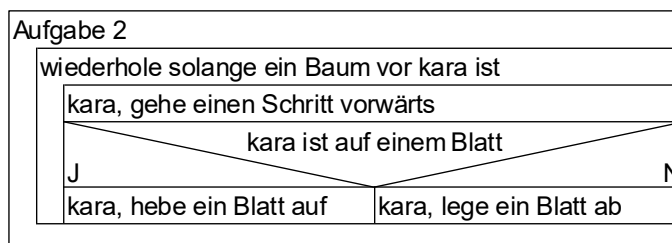
10 for i in range(1, 10):
11     kara.move()
12     if kara.treeLeft() or kara.treeRight():
13         kara.putLeaf()

```

(L1\_5\_1\_A1\_Uebung\_Kontrollstruktur.py)

### Aufgabe 2

#### Struktogramm



(L1\_5\_1\_A2\_Uebung\_Kontrollstruktur.stg)

#### Programmcode

```

10 while not kara.treeFront():
11     kara.move()
12     if kara.onLeaf():
13         kara.removeLeaf()
14     else:
15         kara.putLeaf()

```

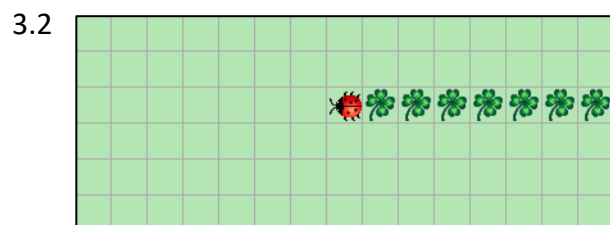
(L1\_5\_1\_A2\_Uebung\_Kontrollstruktur.py)

J1	<b>BPE 5: Grundlagen der Programmierung</b> Lösung	Informatik
----	---	------------

### Aufgabe 3

---

- 3.1 ▶ Zu Beginn wird eine Zählervariable *zaehler* deklariert und ihr Wert auf 0 gesetzt.
- ▶ Der Käfer *kara* geht insgesamt 14 Mal einen Schritt vorwärts.
  - ▶ Nach jedem Schritt wird geprüft, ob er auf einem Blatt steht.
  - ▶ Ist dies der Fall, wird die Variable *zaehler* um 1 erhöht und ein Blatt aufgehoben.
  - ▶ Nach den 14 Schritten dreht sich *kara* zwei Mal nach rechts und steht in Richtung Startposition.
  - ▶ Danach legt *kara* ein Blatt ab und geht einen Schritt vorwärts.
  - ▶ Diese beiden Vorgänge werden so oft wiederholt, wie der Wert der Variable *zaehler* ist.

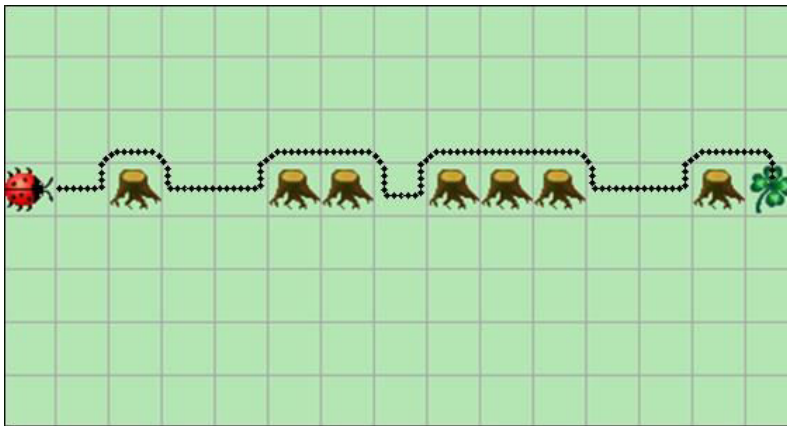


- 3.3 Die Variable *zaehler* weist nach Ablauf des Programms den Wert 7 auf.

J1	<b>BPE 5: Grundlagen der Programmierung</b> Lösung	Informatik
----	---	------------

## Aufgabe 4

- ▶ Zunächst wird geprüft, ob der Käfer *kara* vor einem Baum steht.
- ▶ Ist dies der Fall, geht er um den Baum und bleibt links neben ihm stehen.
- ▶ Anschließend geht er einen Schritt vorwärts, solange ein Baum rechts von ihm steht.
- ▶ Steht rechts von *kara* kein Baum, geht er weiter und stellt sich hinter den Baum.
- ▶ Sofern kein Baum vor *kara* steht, geht er einen Schritt vorwärts
- ▶ Diese Vorgänge werden so oft wiederholt, bis der Käfer *kara* auf einem Blatt steht.
- ▶ Sobald *kara* auf einem Blatt steht, hebt er es auf.



## Aufgabe 5

- ▶ Syntaktischer Fehler: Die Schleifenbedingung enthält einen Sensor, der nicht existiert (*kara.onTree()*).
- ▶ Logischer Fehler: Die Bedingung der Verzweigung ist so formuliert, dass der Käfer *kara* ein Blatt ablegt, wenn er bereits auf einem Blatt steht bzw. ein Blatt aufheben soll, wenn er nicht auf einem Blatt steht.

## Aufgabe 6

- ▶ Die verknüpften Bedingungen der while-Schleife verlangen, dass sich der Käfer *kara* sowohl auf einem Blatt als auch nicht auf einem Blatt befindet.
- ▶ Die beiden Alternativen der if-Verzweigung sind vertauscht. Wenn der Käfer *kara* auf einem Blatt steht, kann er kein Blatt ablegen. Ebenso ist es nicht möglich ein Blatt aufzuheben, wenn *kara* nicht auf einem Blatt steht.



## L1\_5.2 Vertiefungsaufgaben zu Kontrollstrukturen

### Aufgabe 1



Der Käfer *kara* steht am Ende einer Blätterspur und möchte zum Baum gehen (siehe Abb.). Unterwegs sammelt er die Blätter ein. Die Blätterspur kann beliebig lang sein.

Wenn *kara* vor dem Baum steht, sollen alle Blätter aufgehoben sein.

Verwenden Sie aus dem Ordner Aufgaben/Vorlagen die Welt *L1\_5\_2\_Aufgabe1\_Vorlage.world* als Vorlage.

Erstellen Sie ein Struktogramm zur Lösung des beschriebenen Problems und kodieren Sie die Lösung.

Testen Sie Ihr Ergebnis, indem Sie die Standorte der Akteure variieren.

Speichern Sie Ihre Ergebnisse im Ordner *meineErgebnisse* unter den Namen

*L1\_5\_2\_A1\_Vertiefung.stg* (Struktogramm)

*L1\_5\_2\_A1\_Vertiefung.world* und *L1\_5\_2\_A1\_Vertiefung.py* (Programm).

### Aufgabe 2



Der Käfer *kara* steht am Ende der Blätterspur und möchte zum Baum gehen. Unterwegs sammelt er die Blätter ein. Die Blätterspur kann beliebig lang sein und biegt nur nach links ab.

Wenn *kara* vor dem Baum steht, sollen alle Blätter aufgehoben sein.

Verwenden Sie aus dem Ordner Aufgaben/Vorlagen die Welt *L1\_5\_2\_Aufgabe2\_Vorlage.world* als Vorlage.

Erstellen Sie ein Struktogramm zur Lösung des beschriebenen Problems und kodieren Sie die Lösung.

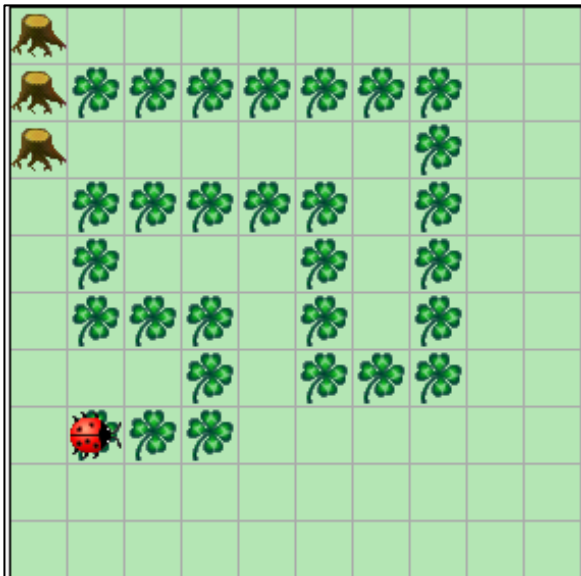
Testen Sie Ihr Ergebnis, indem Sie die Standorte der Akteure variieren.

Speichern Sie Ihre Ergebnisse im Ordner *meineErgebnisse* unter den Namen

*L1\_5\_2\_A2\_Vertiefung.stg* (Struktogramm)

*L1\_5\_2\_A2\_Vertiefung.world* und *L1\_5\_2\_A2\_Vertiefung.py* (Programm).

### Aufgabe 3



Der Käfer *kara* spielt Pacman:

In der Startposition steht *kara* vor dem ersten Blatt einer langen Spur von Blättern. Die Blätterspur kann beliebig gelegt sein, verläuft aber nicht direkt nebeneinander.

Wenn *kara* vor dem Baum steht, sollen alle Blätter aufgehoben sein.

Verwenden Sie aus dem Ordner Aufgaben/Vorlagen die Welt *L1\_5\_2\_Aufgabe3\_Vorlage.world* als Vorlage.

Erstellen Sie ein Struktogramm zur Lösung des beschriebenen Problems und kodieren Sie die Lösung.

Testen Sie Ihr Ergebnis, indem Sie die Standorte der Akteure variieren.

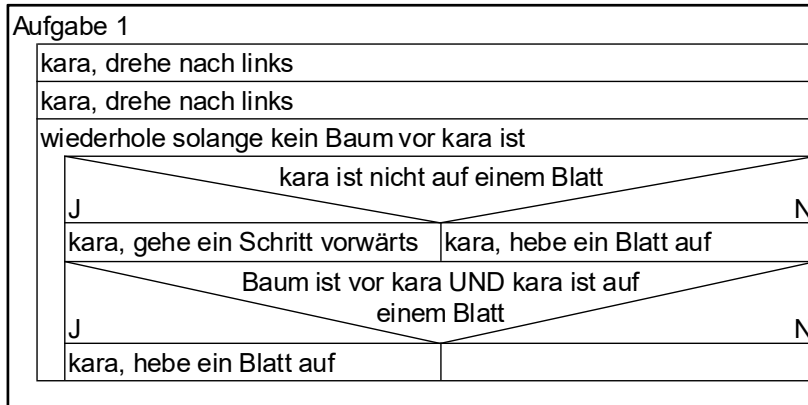
Speichern Sie Ihre Ergebnisse im Ordner *meineErgebnisse* unter den Namen *L1\_5\_2\_A3\_Vertiefung.stg* (Struktogramm)

*L1\_5\_2\_A3\_Vertiefung.world* und *L1\_5\_2\_A3\_Vertiefung.py* (Programm).

## L1\_5.2 Lösungen zu Vertiefungsaufgaben Kontrollstrukturen

### Aufgabe 1

#### Struktogramm



(L1\_5\_2\_A1\_Vertiefung.stg)

#### Programmcode

```

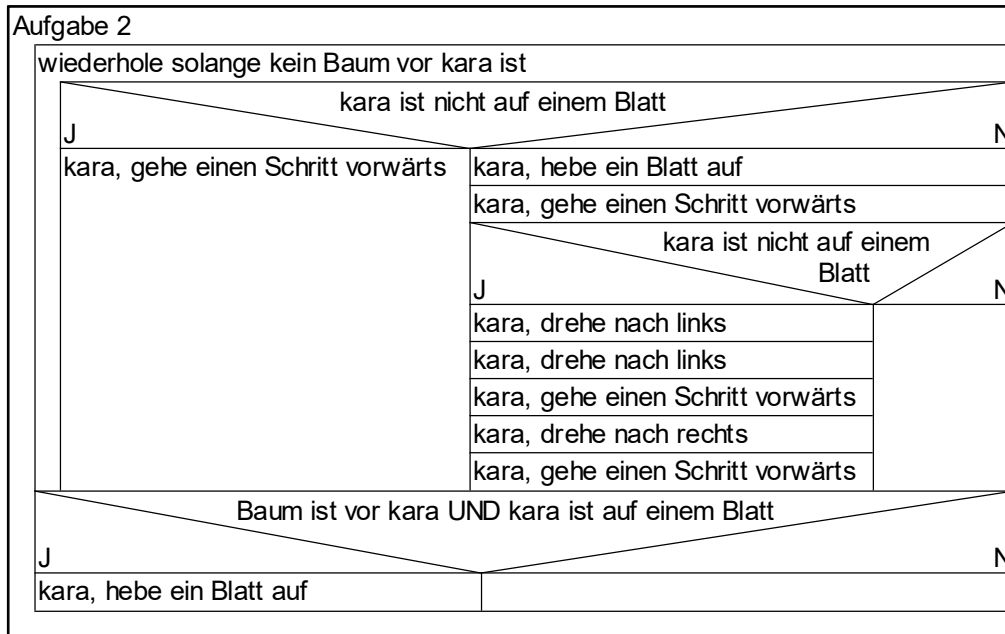
10  kara.turnLeft()
11  kara.turnLeft()
12  while not kara.treeFront():
13      if not kara.onLeaf():
14          kara.move()
15      else:
16          kara.removeLeaf()
17  if kara.treeFront() and kara.onLeaf():
18      kara.removeLeaf()
```

(L1\_5\_2\_A1\_Vertiefung.py)

## Aufgabe 2

### Struktogramm

### Programmcode



(L1\_5\_2\_A2\_Vertiefung.stg)

### Programmcode

```

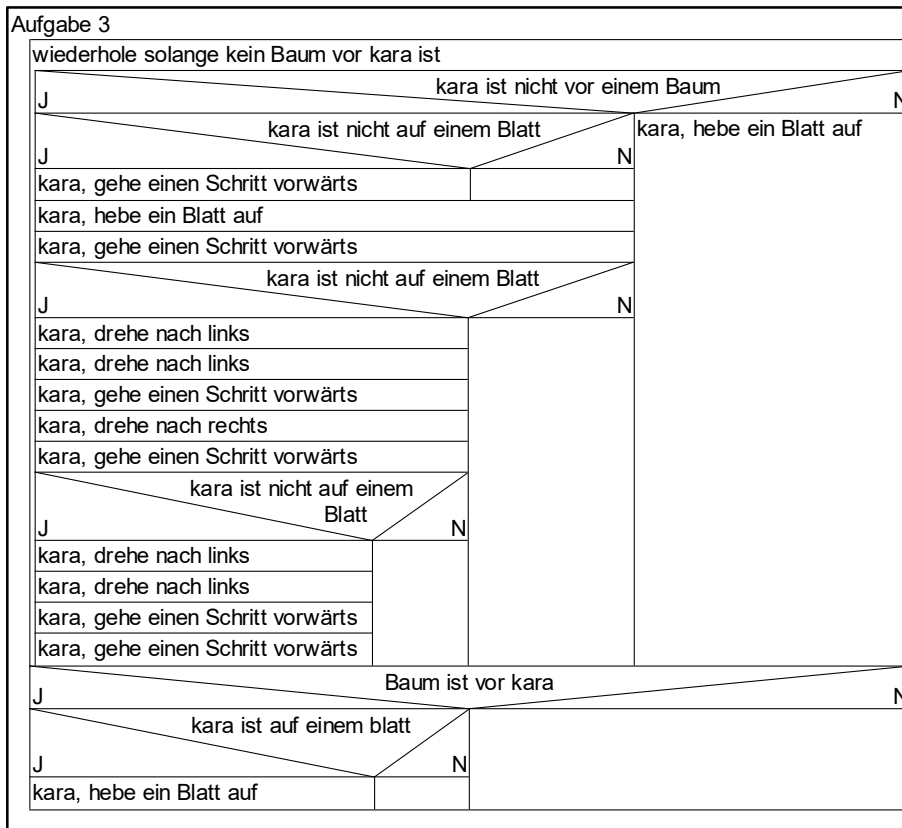
10 while not kara.treeFront():
11     if not kara.onLeaf():
12         kara.move()
13     else:
14         kara.removeLeaf()
15         kara.move()
16         if not kara.onLeaf():
17             kara.turnLeft()
18             kara.turnLeft()
19             kara.move()
20             kara.turnRight()
21             kara.move()
22 if kara.treeFront and kara.onLeaf():
23     kara.removeLeaf()

```

(L1\_5\_2\_A2\_Vertiefung.py)

### Aufgabe 3

#### Struktogramm



(L1\_5\_2\_A3\_Vertiefung.stg)

#### Programmcode

```

10 while not kara.treeFront():
11     if not kara.treeFront():
12         if not kara.onLeaf():
13             kara.move()
14             kara.removeLeaf()
15             kara.move()
16         if not kara.onLeaf():
17             kara.turnLeft()
18             kara.turnLeft()
19             kara.move()
20             kara.turnRight()
21             kara.move()
22         if not kara.onLeaf():
23             kara.turnLeft()
24             kara.turnLeft()
25             kara.move()
26             kara.move()
27     else:
28         kara.removeLeaf()
29 if kara.treeFront():
30     if kara.onLeaf():
31         kara.removeLeaf()
  
```

(L1\_5\_2\_A3\_Vertiefung.py)