Programmier-Anleitungen

Vollständige Dokumentation für:

- Struktog Konfiguration
- Hamstersimulator
- Debug-Funktionen

Struktog Konfigurationsübersicht

Struktog Konfigurationsübersicht

Dieses Dokument bietet eine Übersicht über alle verfügbaren Konfigurationsoptionen in der Struktog-Anwendung.

Verfügbare Konfigurationen

1. default (Standardkonfiguration)

Die Basiskonfiguration mit allen aktivierten Knotentypen. Diese dient als Standard-Fallback-Konfiguration.

2. python

Vollständige Python-Konfiguration mit allen verfügbaren Programmierkonstrukten, identisch zur Standardkonfiguration, aber speziell für die Python-Programmierausbildung angepasst.

3. python_simple

Vereinfachte Python-Konfiguration für Anfänger. Nur grundlegende Elemente sind aktiviert: - Eingabe-/Ausgabefelder - Aufgabenanweisungen - Code-Generierungsbutton

Alle Schleifentypen, Verzweigungen, Funktionen und erweiterte Konstrukte sind deaktiviert.

4. python_if

Python-Konfiguration mit Fokus auf bedingte Anweisungen: - Eingabe-/Ausgabefelder - Aufgabenanweisungen
- Verzweigungsknoten (if/else-Anweisungen) - Code-Generierungsbutton

Schleifen, Funktionen und andere erweiterte Konstrukte sind deaktiviert.

5. python_loop

Python-Konfiguration mit Fokus auf Schleifenkonstrukte: - Eingabe-/Ausgabefelder - Aufgabenanweisungen - Zählerschleifen (for-Schleifen) - Kopfschleifen (while-Schleifen) - Code-Generierungsbutton

Verzweigungen, Funktionen und andere Konstrukte sind deaktiviert.

6. python_for

Python-Konfiguration speziell für zählergesteuerte Schleifen: - Eingabe-/Ausgabefelder - Aufgabenanweisungen - Nur Zählerschleifen (for-Schleifen) - Code-Generierungsbutton

Alle anderen Schleifentypen und erweiterte Konstrukte sind deaktiviert.

7. python_while

Python-Konfiguration speziell für bedingungsgesteuerte Schleifen: - Eingabe-/Ausgabefelder - Aufgabenanweisungen - Nur Kopfschleifen (while-Schleifen) - Code-Generierungsbutton

Alle anderen Schleifentypen und erweiterte Konstrukte sind deaktiviert.

8. python_if_loop

Python-Konfiguration mit Kombinationen aus Bedingungen und Schleifen: - Eingabe-/Ausgabefelder - Aufgabenanweisungen - Verzweigungsknoten (if/else) - Zählerschleifen (for-Schleifen) - Kopfschleifen (while-Schleifen) - Code-Generierungsbutton

Funktionen und erweiterte Konstrukte sind deaktiviert.

9. python_function

Erweiterte Python-Konfiguration mit Funktionen: - Eingabe-/Ausgabefelder - Aufgabenanweisungen - Alle Schleifentypen (Zähler- und Kopfschleifen) - Verzweigungsknoten - Funktionsblöcke - Code-Generierungsbutton

Try-Catch-Blöcke und Fallunterscheidungen sind deaktiviert.

10. standard

Vollständige Konfiguration mit allen verfügbaren Knotentypen aktiviert, einschließlich: - Alle Schleifentypen (Zähler-, Kopf- und Fußschleifen) - Verzweigungsknoten - Fallunterscheidungen - Funktionsblöcke - Try-Catch-Blöcke - Code-Generierungsbutton

Funktionsvergleichsmatrix

Funktion	default	python	python_simple	python_if	
Eingabe-Knoten	[✓]	[✓]	[✓]	[✓]	
Ausgabe-Knoten	[✓]	[✓]	[✓]	[✓]	
Aufgaben-Knoten	[✓]	[✓]	[✓]	[✓]	
Zählerschleife	[✓]	[✓]	[✗]	[✗]	
Kopfschleife	[✓]	[✓]	[✗]	[✗]	
Fußschleife	[✗]	[✗]	[✗]	[✗]	
Verzweigungsknoten	[✓]	[✓]	[✗]	[✓]	
Fallunterscheidung	[✗]	[✗]	[✗]	[✗]	
Funktionsknoten	[✓]	[✓]	[✗]	[✗]	
Try-Catch-Knoten	[✓]	[✓]	[✗]	[✗]	
Code-anzeigen Button	[✓]	[✓]	[✓]	[✓]	

Beschreibung der Knotentypen

Grundlegende Knoten

- **Eingabe-Knoten**: Eingabe-Feld - für Benutzereingabe-Operationen
- **Ausgabe-Knoten**: Ausgabe-Feld - für die Anzeige von Ausgaben

- **Aufgaben-Knoten**: Anweisung - für allgemeine Programmanweisungen

Schleifenknoten

- **Zählerschleife**: Zählergesteuerte Schleife - for/for-in-Schleifen
- **Kopfschleife**: Kopfgesteuerte Schleife - while-Schleifen
- **Fußschleife**: Fußgesteuerte Schleife - do-while-Schleifen

Kontrollfluss-Knoten

- **Verzweigungsknoten**: Verzweigung - if/else-Bedingungsanweisungen
- **Fallunterscheidung**: Fallunterscheidung - switch/case-Anweisungen

Erweiterte Knoten

- **Funktionsknoten**: Funktionsblock - Funktionsdefinitionen
- **Try-Catch-Knoten**: Try-Catch-Block - Ausnahmebehandlung

Spezielle Elemente

- **Einfügeknoten**: Platzhalter für das Einfügen neuer Elemente
- **Fall einfügen**: Spezieller Fall für switch-Anweisungen
- **Platzhalter**: Leerer Platzhalter in der Struktur

Farbcodierung

Jeder Knotentyp hat eine spezifische Farbe für visuelle Unterscheidung: - **Eingabe-/Ausgabe-/Aufgabenknoten**: `rgb(253, 237, 206)` (Hellgelb) - **Schleifenknoten**: `rgb(220, 239, 231)` (Hellgrün) - **Verzweigungs-/Fall-/Try-Catch-Knoten**: `rgb(250, 218, 209)` (Hellorange) - **Funktionsknoten**: `rgb(255, 255, 255)` (Weiß) - **Einfüge-/Platzhalterknoten**: `rgb(255, 255, 243)` (Sehr hellgelb)

Verwendungsempfehlungen

- **Anfänger**: Beginnen Sie mit `python_simple` für grundlegende

Programmierkonzepte

- **Bedingungen**: Verwenden Sie `python_if` beim Unterrichten von if/else-Anweisungen
- **Schleifen**: Verwenden Sie `python_for` oder `python_while` für spezifische Schleifentypen, oder `python_loop` für beide
- **Fortgeschrittene**: Verwenden Sie `python_if_loop` für die Kombination von Bedingungen und Schleifen
- **Erweiterte Funktionen**: Verwenden Sie `python_function` oder `standard` für vollständige Programmierkonstrukte
- **Alle Funktionen**: Verwenden Sie `standard`, wenn alle Sprachfeatures benötigt werden

Hamstersimulator Anleitung

DefaultSubmission Anleitung für Hamstersimulator

Eine gültige `defaultSubmission` für den Hamstersimulator sollte sowohl die Weltkonfigurationen als auch einen Standard-Python-Code enthalten.

Vollständiges Beispiel

```
{
  "configurations": [
    {
      "name": "Einfach",
      "hamsterX": 1,
      "hamsterY": 1,
      "hamsterDirection": "east",
      "hamsterGrains": 0,
      "territory": [
        "w|0|0|w",
        "0|2|1|0"
      ]
    },
    {
      "name": "Komplex",
      "hamsterX": 0,
      "hamsterY": 0,
      "hamsterDirection": "south",
      "hamsterGrains": 3,
      "territory": [
        "w|3|0|w|0",
        "0|w|1|0|2",
        "1|0|0|w|0"
      ]
    }
  ],
  "defaultContent": "# Hamster-Programm - Beispiel\n# Der Hamster sam
}
```


Aufbau der Konfiguration

1. `configurations` Array

Array mit Weltkonfigurationen, zwischen denen der Schüler wechseln kann.

Jede Konfiguration enthält: - **`name`**: Bezeichnung der Welt (z.B. "Einfach",

"Komplex") - **`hamsterX/Y`**: Startposition des Hamsters (0-basiert) - **`hamsterDirection`**: Startrichtung (`"north"`, `"east"`, `"south"`, `"west"`) - **`hamsterGrains`**: Anzahl Körner im Maul zu Beginn - **`territory`**: Array mit Strings, die das Spielfeld beschreiben

2. Territory Format

Das `territory` ist ein Array von Strings, wobei jeder String eine Zeile repräsentiert:

```
"territory": [
  "w|3|0|w|0",    // Zeile 0: Wand, 3 Körner, leer, Wand, leer
  "0|w|1|0|2",    // Zeile 1: leer, Wand, 1 Korn, leer, 2 Körner
  "1|0|0|w|0"     // Zeile 2: 1 Korn, leer, leer, Wand, leer
]
```

Feldtypen: - `w` = Wand - `0` = leeres Feld
- `1,2,3,4...` = Anzahl Körner auf dem Feld - Getrennt durch `|` - Leerzeichen werden ignoriert

3. `defaultContent`

Python-Code der geladen wird, wenn `currentContent` leer ist.

Verfügbare Hamster-Befehle: - `vor()` - Hamster bewegt sich vorwärts - `linksUm()` - Hamster dreht sich nach links - `nimm()` - Hamster nimmt ein Korn auf - `gib()` - Hamster legt ein Korn ab - `vornFrei()` - Prüft ob der Weg frei ist (boolean) - `kornDa()` - Prüft ob ein Korn da ist (boolean) - `maulLeer()` - Prüft ob das Maul leer ist (boolean)

Beispiel für einfache Welt

```
{
  "configurations": [
    {
      "name": "Erste Schritte",
      "hamsterX": 0,
      "hamsterY": 0,
      "hamsterDirection": "east",
      "hamsterGrains": 0,
      "territory": [
        "0|1|0",
        "0|0|2",
        "3|0|0"
      ]
    }
  ],
  "defaultContent": "# Mein erstes Hamster-Programm\n\n# Gehe vorwärt
}
```


Wichtige Hinweise

1. Das JSON muss valide sein (keine Kommentare, richtige Anführungszeichen)
2. Koordinaten sind 0-basiert (beginnen bei 0)
3. Territory-Größe wird automatisch aus dem Array ermittelt
4. Außen um das Territory sind automatisch Wände
5. Der `defaultContent` ersetzt `currentContent` nur wenn dieser leer ist
6. Hamster-Funktionen sind automatisch verfügbar und zeigen keine Fehler im Editor

Debug Content Viewer

Debug Content Viewer

Zweck

Ermöglicht das Anzeigen des gespeicherten Inhalts von Submissions für den aktuell eingeloggten Benutzer zu Debug-Zwecken.

URL-Format

```
/debug/content/{taskId}?version={versionNumber}
```

Parameter

- `taskId` (Pfad): ID der Aufgabe
- `version` (Query, optional): Spezifische Versionsnummer. Wenn nicht angegeben, wird die neueste Version angezeigt

Beispiele

```
# Neueste Version für Task 10 (aktueller Benutzer)
/debug/content/10

# Spezifische Version 3 für Task 10 (aktueller Benutzer)
/debug/content/10?version=3
```

Funktionalität

- Zeigt den rohen Inhalt einer Submission des aktuell eingeloggten Benutzers an
- Verwendet automatisch den aktuell authentifizierten Benutzer
- Validiert Task-Existenz und Benutzer-Anmeldung
- Unterstützt sowohl neueste als auch spezifische Versionen

- Zeigt Metadaten wie Submission-Zeit und Version an

Sicherheit

- Nur eingeloggte Benutzer können ihre eigenen Submissions einsehen
- **[!] Nur für Entwicklung** - Diese Funktion ist nirgends verlinkt und sollte nur in der Entwicklungsumgebung verwendet werden.