

# GuiPy – eine Python-IDE für die Schule

Röhner, G.

DOI: 10.18420/ibis-03-01-06

## Zusammenfassung

Im Informatikunterricht der Sekundarstufe II ist die objektorientierte Modellierung bundesweit verankert. Zur Unterstützung erfolgreicher Lehr- und Lernprozesse stellt die Python-IDE GuiPy hilfreiche Modellierungswerkzeuge bereit, mit denen grundlegende objektorientierte Konzepte veranschaulicht und erarbeitet werden können. Mit dem Struktogrammeditor können Schülerinnen und Schüler Algorithmen modellieren. Der GUI-Designer gibt ihnen eine leicht nutzbare Möglichkeit, ihre eigenen Projekte mit einer modernen und vorzeigbaren grafischen Benutzeroberfläche zu versehen. Im Artikel werden neben grundsätzlichen Anmerkungen zu Python die Modellierungswerkzeuge von GuiPy vorgestellt.

## Python in der Schule

Mit einer kurzen Internetrecherche findet man zahlreiche Entwicklungsumgebungen für Python. Daher stellt sich natürlich die Frage, wieso eine weitere Python-IDE (engl. Integrated Development Environment) überhaupt sinnvoll ist. Diese Frage soll in diesem Artikel beantwortet werden. Im Kern geht es darum, dass eine IDE für die Schule Lehr- und Lernprozesse so gut wie möglich unterstützen soll. Der Schwerpunkt liegt also nicht bei professionellen Ansprüchen, sondern bei didaktischen Funktionalitäten. Diese werden im Kontext von Modellierungswerkzeugen dargestellt und unterstützen somit Informatiklehrkräfte, die GuiPy als schulgeeignete Python-IDE einsetzen.

Für Programmieranfänger sind blockbasierte Sprachen und IDEs wie *Scratch*, *Snap!* oder *App Inventor* sehr gut geeignet, da sie damit erfolgreich kreative Animationen, Geschichten und Spiele realisieren können. Blöcke wählt man einfach aus einer Palette aus, das Vokabular einer Computersprache muss man nicht kennen, sie basieren auf Wiedererkennung statt auf Erinnerung. Blöcke helfen Lernenden, Code ohne grundlegende Fehler zusammenzustellen, denn zwei inkompatible Konzepte haben keine verbindenden Teile, syntaktische Fehler werden somit vermieden (Weintrop&Grover, 2020). Spätestens zu Beginn der Oberstufe stellt sich die Frage, ob man zu einer textbasierten Sprache wechselt. Sicherlich ist es so, dass sich Schüle-

rinnen und Schülern mit dem Wechsel zu einer textbasierten Programmiersprache eine neue Welt erschließt und ihre Kompetenzen so erweitert, dass sie auf Programmieranforderungen in Ausbildung oder Studium besser vorbereitet sind.

Doch welche textbasierte Sprache soll es sein? In den letzten zwanzig Jahren wurden bevorzugt Delphi und Java in der Oberstufe und im Abitur eingesetzt. Delphi bzw. Lazarus als IDEs für die von Niklaus Wirth entwickelte Lehrsprache Pascal werden in der universitären Ausbildung nicht verwendet. Das stellt junge Lehrkräfte, die an eine Schule mit Delphi als Programmiersprache kommen, vor erhebliche Herausforderungen. Java wird an Universitäten häufig gelehrt. Allerdings ist Java syntaktisch überladen, im Wesentlichen objektorientiert und macht es den Anfängerinnen und Anfängern oft unnötig schwer, einfache Dinge wie z.B. eine Texteingabe zu erledigen. Python hingegen hat eine einfache und gut lesbare Syntax, kann objektorientiert und prozedural genutzt werden, erfordert weniger Code als z. B. Java und Programmbausteine können ganz leicht interaktiv getestet werden (Bovermann, 2024). Daher ist Python sehr beliebt und lässt sich vielfältig sowie bereits in jüngeren Jahrgangsstufen gut einsetzen. (Jugendwettbewerb Informatik, 2025).

Man kann zwar auf der Konsole in Python programmieren, aber der Einsatz einer integrierten Entwicklungsumgebung (IDE) ist empfehlenswert, weil diese vielfältige Unterstützung beim Programmieren gibt. Die existierenden Python-IDEs haben einen klaren Fokus auf dem Schreiben und Debuggen von Python-Programmen, also beim Implementieren. Das Modellieren wird hingegen nicht unterstützt. Da die Bildungsstandards Informatik für die Sekundarstufe II der GI (GI, 2016) den Prozessbereich „Modellieren und Implementieren“ ausweisen, sollten für die Unterrichtspraxis auch geeignete Modellierungswerkzeuge zur Verfügung stehen. Hier setzt die grundsätzliche Idee von GuiPy an. Es ist eine Python-IDE für die Schule, die bei der Objektorientierung neben dem Implementieren auch das Modellieren durch vielfältige Werkzeuge unterstützt.

## Klasseneditor

Mit dem Klasseneditor stellt GuiPy ein Werkzeug zur Verfügung, das die Schülerinnen und Schüler beim Modellieren einer Klasse hilft und ihnen die konkrete Implementierung einer Klasse abnimmt. Das entlastet von syntaktischen Details und konzentriert die Modellierung auf die Abstraktionsprozesse, welche für die Modellierung essentiell sind, nämlich die problemadäquate Auswahl und Gestaltung von Attributen und Methoden.

Auf der Registerkarte *Attribute* lässt sich die Sichtbarkeit für ein Attribut auswählen. Ist eine get- oder set-Methode ausgewählt, so wird beim Anlegen eines Attributs die jeweilige Methode automatisch angelegt. In der Konfiguration des Klassenmodellierers wählt man die gewünschte Voreinstellung aus. Beim Modellieren muss man darüber nachdenken, welche get- bzw. set-Methoden gebraucht werden und sinnvoll sind. Den Kilometerstand z. B. darf man zur Vermeidung einer Tachomanipulation nicht mit einer Set-Methode verändern können, das darf nur die Methode `fahren(Strecke: float)`.

Python ist eine Programmiersprache mit dynamischer Typisierung, d. h. für Variablen, Attribute oder Parameter müssen keine Datentypen angegeben werden. Trotzdem stehen im Klassenmodellierer Auswahlfelder für Datentypen bereit, denn es soll einerseits möglich sein, Klassen gemäß UML-Standard zu modellieren und andererseits sind Programme und Methoden einfacher zu lesen, zu verstehen und zu debuggen, wenn Datentypen angegeben sind. Wer

Typisierung nicht möchte, kann in der Konfiguration die Auswahl von Datentypen verbergen.

Ab Python 3.5 stehen mit PEP 484 (engl. Python Enhancement Proposal, van Rossum, 2014) die sogenannten *Type Hints* zur Verfügung, welche statische Typisierung durch eine Syntaxerweiterung ermöglicht. Hier ist beispielsweise eine einfache Funktion, bei der der Datentyp für den Parameter und Rückgabebetyp deklariert ist:

```
def toString(num: int) → str:
    return str(num)
```

Trotz dieser Typisierung findet zur Laufzeit keine Typprüfung statt. Dafür sind zusätzliche Werkzeuge wie z. B. *mypy* vorgesehen. In der Konfiguration lässt sich *mypy* als Werkzeug einrichten.

In der linken Strukturansicht des Klasseneditors wählt man Attribute oder Methoden zur Bearbeitung aus. Per Drag&Drop lässt sich auch die Reihenfolge ändern, was sich auch auf den im Hintergrund erzeugten Quelltext auswirkt. Mit *Übernehmen* werden Änderungen wirksam, *Neu* schafft Platz für ein neues Attribut oder eine neue Methode.

Parallel zur Modellierung wird im Hintergrund der Pythoncode für die modellierte Klasse erstellt. Alle get- und set-Methoden, sowie der Konstruktor sind am Ende fertig implementiert. Die anderen Methoden müssen später im Editor implementiert werden. Schließt man den Klasseneditor, so wird der erzeugte Pythoncode ausgeführt. Hat man den Klasseneditor von einem UML-Fenster aus aufgerufen, so wird als

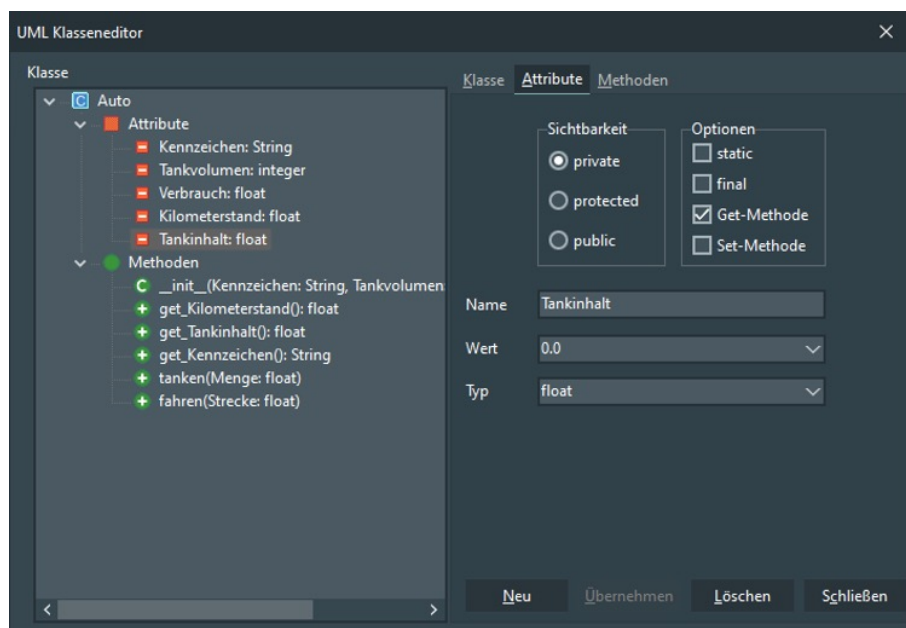


Abbildung 1: Der Klasseneditor mit geöffneter Registerkarte Attribute

Ergebnis das Klassendiagramm dargestellt, ansonsten der erzeugte Pythoncode in einem Editorfenster.

## UML-Fenster

### Klassen und Klassenbeziehungen im UML-Fenster

Ein UML-Fenster stellt ein oder mehrere Klassen als Klassendiagramm grafisch dar. GuiPy erkennt automatisch Assoziationen zwischen den Klassen, sofern statische Typisierung benutzt wird. Im Beispiel hat die Klasse `Person` das Attribut `Konten` mit dem Typ `list[Konto]`. Daraus ermittelt GuiPy, dass eine Aggregation zur Klasse `Konto` besteht. Ohne statische Typisierung modelliert man Assoziationen über den Befehl *Verbinden mit* im Kontextmenü einer Klasse. Nachträglich lassen sich Assoziationen per Doppelklick editieren, so dass man die Art einer Assoziation ändern, sowie zusätzliche Angaben wie Rollen oder Multiplizitäten angeben kann.

Im Kontextmenü einer Klasse stehen einige Optionen zur Einstellung der Anzeige von Sichtbarkeit, Parameter und Datentypen bereit. Dies ist für die bessere Übersicht bei einem Diagramm mit vielen Klassen oder zur Gestaltung eines Aufgabenblatts hilfreich.

Hat man mehrere Klassen in jeweils eigenen Dateien modelliert, so verwendet man den Befehl *Diagramm aus geöffneten Dateien* um ein gemeinsames Klassendiagramm für alle geöffneten py-Dateien zu erzeugen. Alternativ dazu

gibt es noch den Befehl *Öffne Ordner*, der aus allen Klassendateien eines Ordners ein Klassendiagramm erstellt. Für ein Pythonprojekt, das man beispielsweise im Internet gefunden hat, kann man so ganz einfach ein Klassendiagramm erstellen und damit einen Überblick über die Projektstruktur erhalten.

Auf Basis der Klassenbeziehungen lässt sich automatisch ein passendes Layout berechnen. Wählt man mehrere Klassen aus, so richtet man diese über das Kontextmenü zueinander aus. Für die Verwendung in einem Arbeitsblatt, einer Dokumentation, einer Lernumgebung oder Webseite speichert man das fertige Klassendiagramm als skalierbare SVG-Grafik ab.

In einem UML-Fenster können Schülerinnen und Schüler somit ein Klassendiagramm aus mehreren Klassen und Klassenbeziehungen modellieren. Das hat aus didaktischer Perspektive den Vorteil, dass sie sich selbst aktiv handelnd mit Klassen und Klassenbeziehungen auseinandersetzen. Es geht nicht nur um vorgegebene Klassendiagramme, sondern sie können auch eigene Klassendiagramme leicht erstellen. Gegenüber auf Papier modellierten Klassen hat das UML-Fenster als Werkzeug den Vorteil, dass Entwürfe einfach bearbeitet werden können und man professionelle Ergebnisse produziert. Durch die Integration des UML-Fensters in die GuiPy-IDE können Schülerinnen und Schüler sowie Lehrkräfte innerhalb einer Umgebung mit Klassendiagrammen und Pythoncode arbeiten. Da kein externer UML-Editor notwendig ist, wird der Unterricht entlastet.

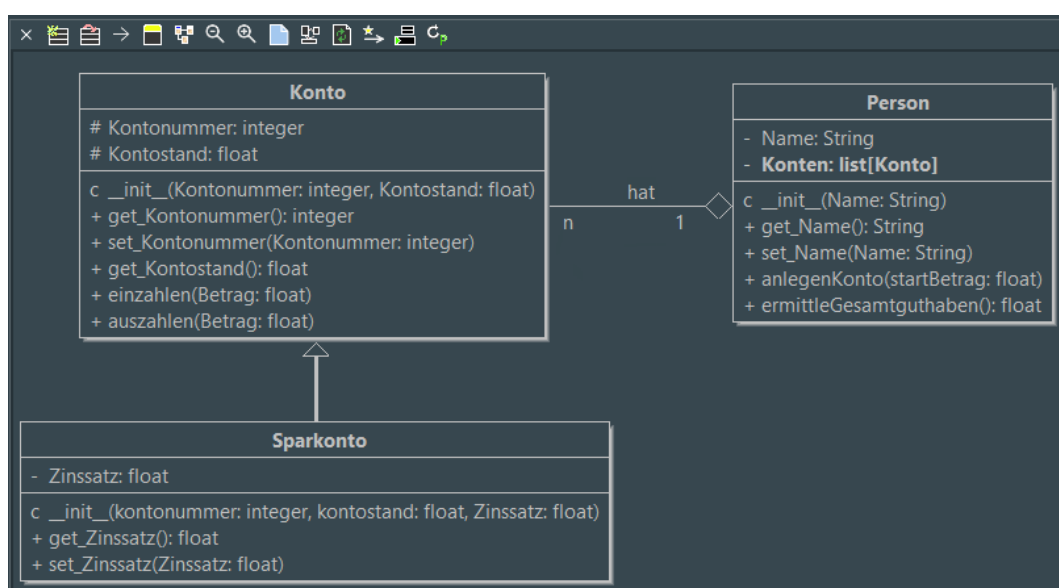


Abbildung 2: Klassen und Klassenbeziehungen im UML-Fenster

## Objekte im UML-Fenster


Aus der Literatur ist bekannt, dass beim objektorientierten Ansatz Lernende Schwierigkeiten haben zwischen Klassen und Objekten zu unterscheiden (Eckerdal, 2004). Da dies aber grundlegende Konzepte des OO-Ansatzes sind, muss man didaktische Ansätze entwickeln, um diese Schwierigkeiten zu meistern. Mit sprachlichen Erklärungen alleine, lässt sich das Problem nicht lösen. Auch die Umsetzung auf der Programmierenebene reicht nicht aus. Beispielsweise hilft die Erstellung eines Objekts der Klasse `Auto` mittels `meinAuto = Auto('DA RR 1050', 60, 7.3)` mit den Parameterwerten für Kennzeichen, Verbrauch und Tankkapazität nicht wirklich, das Objektkonzept zu verstehen.

Erst das Zusammenspiel von Erklärungen, Visualisierungen und selbstständige Auseinandersetzung mit konkreten Beispielen hilft, die Hürden bei der Unterscheidung von Klassen und Objekten zu überwinden. Aus diesem Grunde bieten didaktisch gestaltete Werkzeuge wie `GuiPy`, `JavaEditor` bzw. `BlueJ` Visualisierungen anhand eigener Beispiele an. Im Kontextmenü einer Klasse ruft man den Konstruktor auf, um ein Objekt zu erzeugen. Eventuell erforderliche Parameterwerte werden abgefragt. Erstellt man zu einer Klasse mindestens zwei Objekte, so lassen sich im kombinierten Klassen- und Objektdiagramm gut verständlich Unterschiede und Gemeinsamkeiten zwischen Klassen und Objekten in einer Unterrichtssituation besprechen.

Wir haben eine Klasse `Auto`, aber zwei `Auto`-Objekte und weitere könnten erstellt werden. Bei den Objekten werden nur die Attribute und deren Werte angegeben, die sich zwischen den Objekten unterscheiden. Bei der Klasse `Auto`

werden stattdessen die Attribute und deren Datentypen sowie der Konstruktor und die Methoden angegeben. Man erkennt, dass die Objekte auf Basis der Klasse `Auto` erstellt sind.

Aus dem Kontextmenü der erstellten Objekte kann man die Methoden der Klasse aufrufen. Da der Klasseneditor `get-` und `set-`Methoden automatisch implementiert, lassen sich diese interaktiv testen, ohne von der Modellierungsebene zur Programmierenebene wechseln zu müssen.

Die Methoden `tanken(Menge: float)` und `fahren(Strecke: float)` müssen im Editorfenster der Klasse `Auto` implementiert werden. Der Test der Implementierung erfolgt im UML-Fenster. Beim Wechsel in das UML-Fenster wird die Klasse mit violettem Kopf als ungültig dargestellt, weil Quellcodeänderungen vorliegen und diese noch nicht vom Python-Interpreter verarbeitet wurden. Durch das Ausführen des UML-Fensters über das grüne Dreieck wird die Klasse gültig, aber die Objekte gehen verloren, weil sie nicht mehr zur geänderten Klasse gehören. Zur Entlastung des Unterrichts öffnet das Icon  ein Fenster, das die interaktiven Aktionen protokolliert. Über dessen Ausführen-Symbol lassen sich die Objekte schnell wieder herstellen. Nach jedem Aufruf einer Methode wird der aktuelle Objektzustand dargestellt, eine Inspektion wie in `BlueJ` ist nicht erforderlich. So kann man die Wirkung von Methodenaufrufen auf den Objektzustand leicht erkennen und die Implementierung der Methoden testen.

Unterrichtliche Erfahrungen zeigen, dass mit diesem Ansatz auch schwächeren Schülerinnen und Schülern der Zugang zu den Konzepten Klasse und Objekt gut gelingt. Insbesondere werden sie auch durch das Protokollfenster beim Übergang vom Modellieren zum Imple-

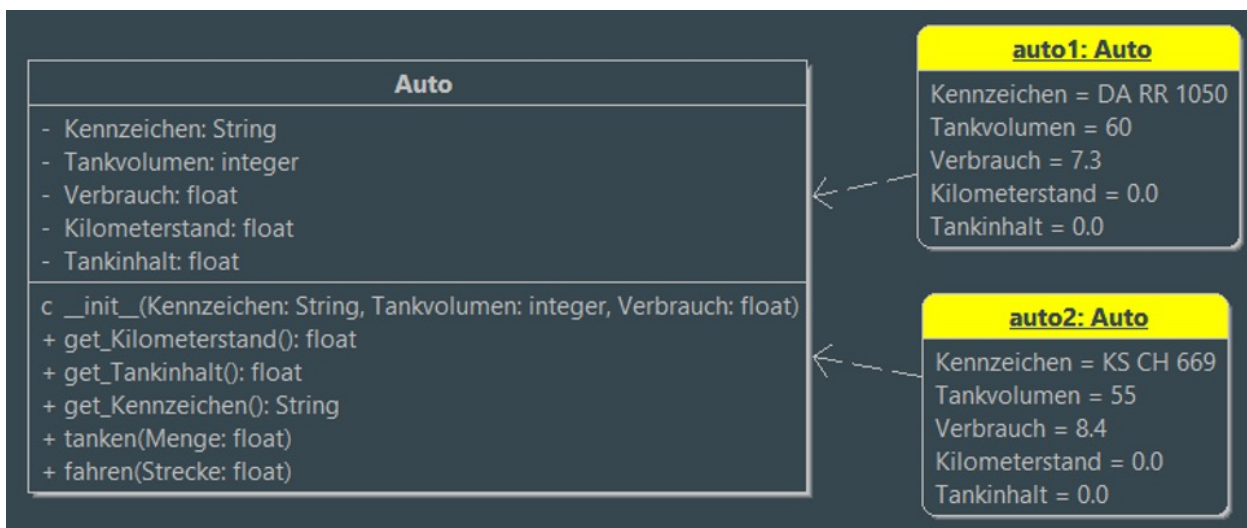


Abbildung 3: Kombiniertes Klassen- und Objektdiagramm zur Klärung des Unterschieds zwischen Klasse und Objekt

mentieren unterstützt. Bei interaktiven Aufrufen von Konstruktoren und Methoden wählt man wie bei blockbasierten Sprachen nur aus bzw. füllt eine Tabelle für die Parameterwerte aus, benötigt aber keine Syntaxkenntnisse. Im Protokollfenster können sie dann sehen, wie Konstruktor- und Methodenaufrufe in Python implementiert werden.

## Dynamische Datenstrukturen im UML-Fenster

Wenn im Unterricht dynamische Datenstrukturen wie Listen oder Bäume behandelt werden, so bietet das UML-Fenster eine ausgezeichnete Darstellung, in der alle Referenzen zwischen den Objekten sichtbar sind. Im Beispiel wird das für eine einfach verkettete Liste mit drei Listenelementen gezeigt, welche mit `verketteteListe1 = VerketteteListe([4,13,2])` erzeugt wurde.

Jede Objektreferenz wird als Pfeil dargestellt, wodurch die lineare Struktur der Liste deutlich wird. Das Attribut `Anfang` der Klasse `VerketteteListe` verweist auf den Kopf der Liste und mit `Nächster = None` ist `knoten3` das Ende der Liste. Bei einer doppelt verketteten Liste, werden auch die Pfeile rückwärts dargestellt. In dieser Umgebung können die Schülerinnen und Schüler einzelne Methoden der Klasse `VerketteteListe` implementieren und testen. Nach z. B. dem Programmieren der Methode `löschen(Daten)` und dem erneuten Erstellen der Liste mit Hilfe des Protokollfensters kann man beim Objekt `verketteteListe1` die Methode `löschen(13)` aufrufen. Das sich ergebende Objektdiagramm zeigt, ob korrekt gelöscht wurde oder nicht. Das Ergebnis muss allerdings interpretiert und die Implementierung möglicherweise überarbeitet werden.

Aber warum sollte man die lineare Liste als dynamische Datenstruktur überhaupt behandeln, wenn Python den in der Sprache eingebauten Datentyp `Liste` hat? Die linear verkettete Liste kann man als einfaches Einführungsbeispiel für eine dynamische Datenstruktur ansehen, die mit Hilfe von Objektreferenzen realisiert wird. Damit werden grundlegende Kompetenzen für die Nutzung und Anwendung des Referenzkonzepts erworben. Auf dieser Basis können weitere Datenstrukturen wie zum Beispiel Stapel, Warteschlange, binärer Suchbaum oder Graph verstanden und realisiert werden.

Mit der Darstellung einer dynamischen Datenstruktur als Objektdiagramm und das einfache Testen einer Methodenimplementierung durch Aufruf im UML-Fenster mit anschließender visueller Überprüfung des Ergebnisses wird für die Schülerinnen und Schüler eine sehr große Zugänglichkeit zum Thema dynamische Datenstrukturen geschaffen. Sie können selbständig implementieren, Testergebnisse reflektieren, Fehler finden sowie korrigieren und damit die Kompetenzen erwerben, die für die Nutzung von Referenzstrukturen nötig sind.

Die didaktische Konzeption des UML-Fensters geht weit über das hinaus, was BlueJ zu bieten hat. Es steht ein vollwertiger Editor für Klassendiagramme zur Verfügung, mit dem alle Klassenbeziehungen modelliert werden können. Objekte werden in einem gemeinsam Klassen- und Objektdiagramm dargestellt, wobei alle Attributwerte direkt einzusehen sind. Die erzeugte Objektstruktur wird aus den Beziehungen zwischen den Objekten vollständig repräsentiert. Die hohe Transparenz und einfachen Zugänglichkeit ermöglicht den Schülerinnen und Schüler einen erfolgreichen Einstieg in die objektorientierte Modellierung und Programmierung.

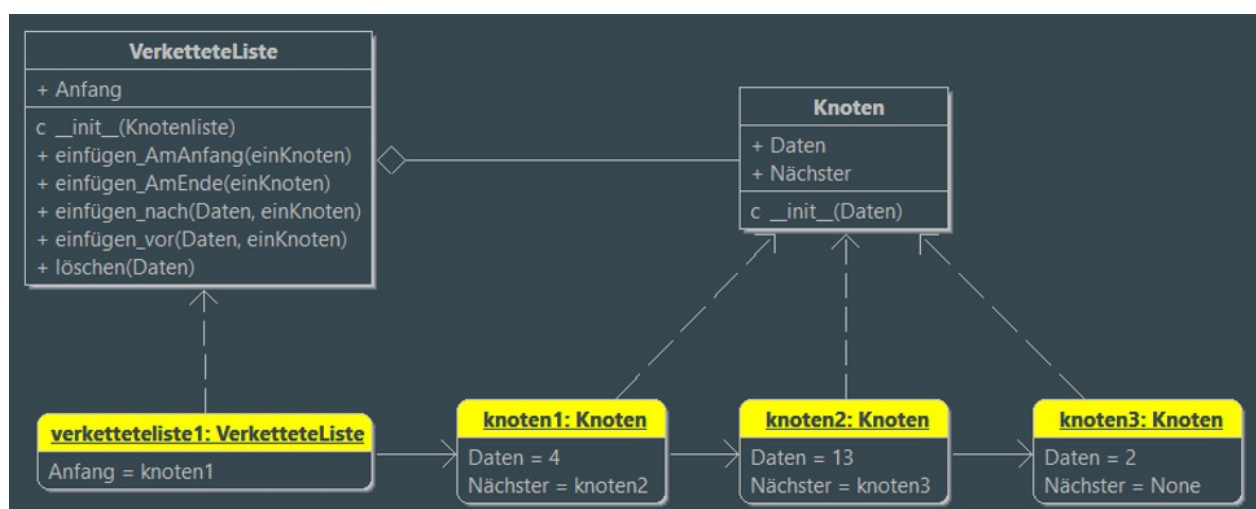


Abbildung 4: Darstellung einer 3-elementigen Liste als Objektdiagramm

## Struktogrammeditor

Indem man dem Implementieren eine Modellierungsphase voranstellt, kann man auf einer abstrakten und dem eigenen Denken entsprechenden Ebene arbeiten. Insbesondere muss man beim Modellieren nicht auf syntaktische Details bzw. Möglichkeiten einer Programmiersprache Rücksicht nehmen. In einem Top-Down- oder Bottom-Up-Ansatz kann ein Algorithmus zur Lösung eines Problems schrittweise entwickelt werden. Der Einsatz eines Struktogrammeditors unterstützt die Lernenden beim Modellieren, weil die algorithmischen Grundstrukturen aus einer Palette ausgewählt und Änderungen im Entwurf leicht durchzuführen sind. Sie erhalten im Gegensatz zu einer handschriftlichen Herangehensweise ein vorzeigbares und bearbeitbares digitales Ergebnis, wodurch auch ihre Kompetenzen beim Umgang mit digitalen Werkzeugen erweitert werden.

Der Algorithmus zum Fahren einer Strecke könnte in einem ersten Entwurf so aussehen:

Er ist sprachlichen und gedanklichen Strukturen angelehnt, welche beim Nachdenken über das Fahren in den Sinn kommen können. Es ist

Im nächsten Schritt wird der modellierte Algorithmus verfeinert. Die bisherigen Formulierungen werden mit den Attributen Tankinhalt, Kilometerstand und Verbrauch der Klasse Auto in Beziehung gesetzt und in Berechnungen überführt. Umgangssprachliche und mehrdeutige Formulierungen wie z.B. „Reicht dieser Verbrauch?“ werden in eindeutige Anweisungen bzw. Ausdrücke überführt.

In dieser Form kann der Algorithmus mit dem entsprechenden Befehl aus dem Kontextmenü in eine Python-Funktion übertragen werden. Der erzeugte Quellcode bildet die algorithmische Struktur des Algorithmus richtig in Python-Kontrollstrukturen ab, muss in der Regel aber syntaktisch überarbeitet werden. Für Schülerinnen und Schüler, die sich beim Programmieren schwertun, ist die automatische Übertragung eines Algorithmus in Programmcode eine gute Hilfe. Platziert man den Cursor in die Kopfzeile einer Pythonfunktion oder markiert man einige Pythonzeilen im Codeeditor, so kann man umgekehrt auch Pythoncode in ein Struktogramm übertragen.

Der Struktogrammeditor stellt auf einer Symbolleiste alle Struktogrammelemente bereit.

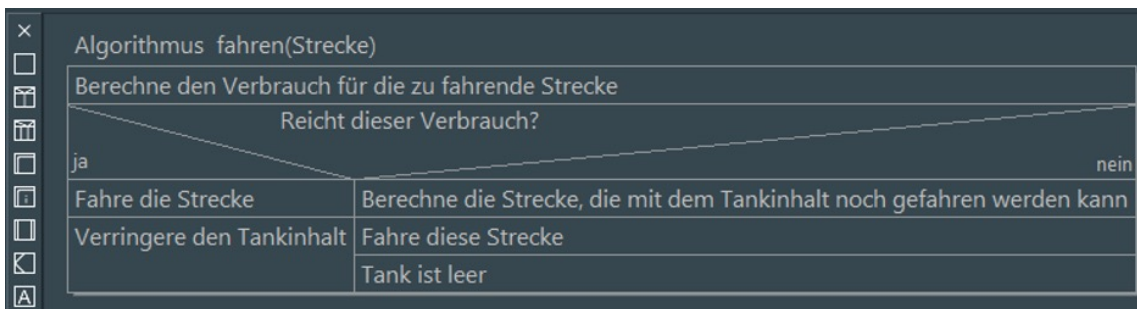


Abbildung 5: Umgangssprachlicher Entwurf für den fahren-Algorithmus

auch schon eine wesentliche Modellierungsentscheidung getroffen worden, nämlich wie mit Strecken umgegangen wird, die die aktuelle Reichweite überschreiten: gar nicht erst losfahren oder fahren bis der Tank leer ist. Auf dieser Ebene ist es aber noch nicht nötig, sich Gedanken über einzelne Berechnungen zu machen.

Diese können ähnlich wie in Scratch per Drag&Drop benutzt werden, um ein Struktogramm zu erstellen. Durch Abziehen nach unten lassen sich ganze Blöcke ablösen und dann wieder neu zusammensetzen. Nach einem Doppelklick gibt man Text direkt in ein Strukturelement ein. Die Return-Taste erzeugt mehrzeiligen

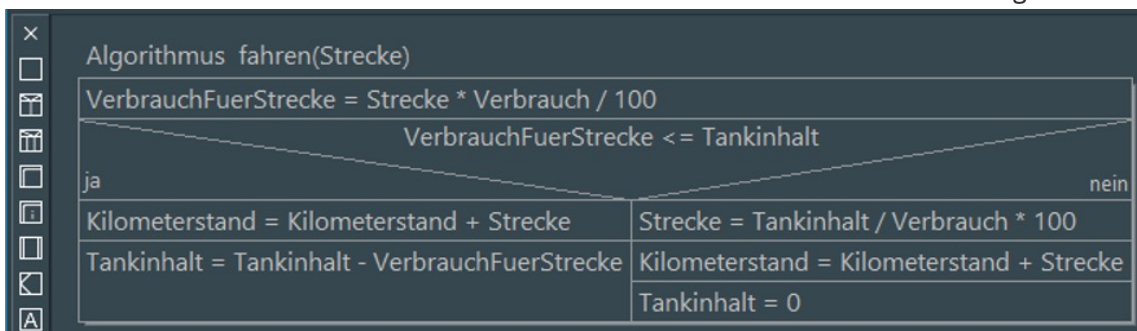


Abbildung 6: Implementierbarer Entwurf des fahren-Algorithmus

Text, was bei Fallunterscheidungen oft sinnvoll ist. In der Konfiguration von GuiPy gibt es eine Seite für den Struktogrammeditor, auf der unter anderem Texte angepasst werden können.

Über das Kontextmenü eines fertigen Struktogramms schaltet man in den Puzzlmodus um. Im Puzzlmodus zerlegt eine Lehrkraft das Struktogramm in einzelne Puzzleteile. Der weitere Befehl *Speichere Puzzle Dateien* erzeugt Struktogrammpuzzles der Schwierigkeitsstufen *leicht, mittel, schwer* und *sehr schwer*. Schülerinnen und Schüler können sich eine für sie passende Schwierigkeitsstufe auswählen und ihre Lösung mit dem Puzzle-Symbol überprüfen.

## Sequenzdiagramme

Mit einem Sequenzdiagramm wird der zeitliche Ablauf einer Interaktion modelliert. Jedes beteiligte Objekt erhält eine abwärts verlaufende gestrichelte Lebenslinie als Zeitachse. Die Kommunikation der Objekte wird durch horizontale Pfeile zwischen den Lebenslinien dargestellt, welche jeweils eine Nachricht übermitteln. Handelt es sich um Objekte als Exemplare von Klassen, so entsprechen Methodenaufrufe den Nachrichten.

Das Beispiel zeigt das Sequenzdiagramm für eine mögliche Interaktion im UML-Fenster, bei der zunächst ein Auto-Objekt erzeugt, erfolglos gefahren, betankt und dann erfolgreich gefahren wird. Das sind die zugehörigen Methodenaufrufe:

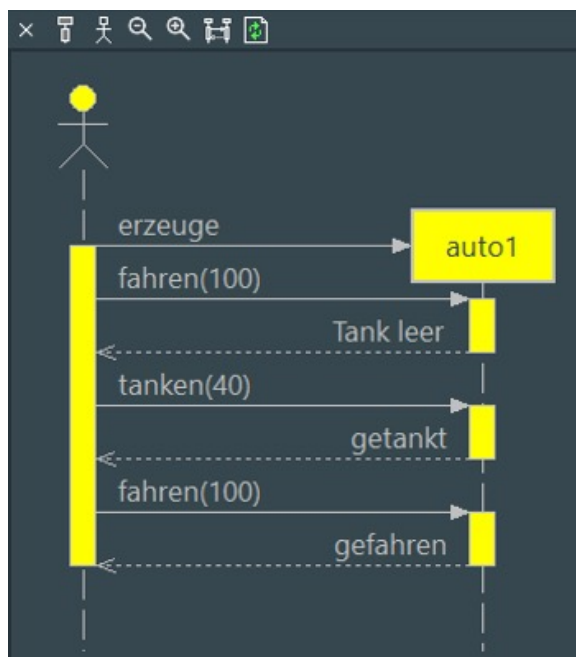


Abbildung 7: Sequenzdiagramm zum Erzeugen, Tanken und Fahren eines Autos

```

auto1 = Auto('DA RR 1023', 60, 7.3)
auto1.fahren(100)
auto1.tanken(40)
auto1.fahren(100)
  
```

Lebenslinien erzeugt man mit den betreffenden Symbolen der Symbolleiste. Über das Kontextmenü einer Lebenslinie erstellt man Nachrichten zu anderen Objekten.

Sequenzdiagramme können Kommunikationsvorgänge, wie sie beispielsweise bei der Client-Server-Kommunikation auftreten, sehr gut darstellen. Für die objektorientierte Modellierung in der Schule haben sie keine so große Relevanz, weswegen sie in vielen Bundesländern nicht zum Curriculum gehören.

## GUI-Designer

Scratch verfügt eine Bühne und Greenfoot über eine Welt, in der man mit vorgegebenen oder abgeleiteten Grafikobjekten arbeiten kann. Das hat den Vorteil, dass die Entwicklung interaktiver Animationen und Simulationen vielfältig unterstützt wird, von der Gestaltung der Bühne bzw. Welt bis hin zur interaktiven Ablaufsteuerung. GuiPy verfolgt einen anderen Ansatz, bei dem die Beschränkungen durch eine Bühne bzw. Welt aufgehoben und durch eine vollkommen freie Gestaltung der Benutzeroberfläche ersetzt wird. Dieser Ansatz wird durch den GUI-Designer ermöglicht, mit dem man interaktiv Widgets aus Paletten in ein GUI-Formular platziert und deren Attribute im Objektinspektor konfiguriert, wobei synchron dazu der entsprechende Pythoncode automatisch erzeugt wird. Grafische Benutzeroberflächen direkt durch Programmierung zu erzeugen, wie es beispielsweise bei *inf-schule.de* vorgeschlagen wird (Becker, 2023), erfordert sehr viel Detailwissen über Attribute und Methoden der Widgets und ist somit ein sehr mühsamer und nicht zielführender Ansatz.

Python verfügt von Hause aus über die beiden Grafikbibliotheken *Tkinter* und *TTK*. Sie stehen in Python direkt zur Verfügung. Die modernere Grafikbibliothek *Qt* muss zusätzlich installiert werden, wofür GuiPy das Werkzeug *Pakete mit PIP installieren* bereitstellt. Für *Qt* benötigt man das Paket *PyQt6*. Weitere Installationshinweise finden Sie auf der GuiPy-Homepage *gui.py.de* (Röhner, 2024b). Auf der Registerkarte *Programm* wählen Sie *Tkinter/TTK* oder *Qt* aus, wenn Sie eine GUI-Anwendung erstellen wollen. *Qt* als moderne Bibliothek mit konsistenter

Schnittstelle für die Ein- und Ausgabe wird empfohlen.

Die wichtigsten Widgets stehen auf der Registerkarte *Qt Base* bereit. Per Drag&Drop platziert man sie auf dem GUI-Formular. Die Attribute eines Widgets bearbeitet man im Objektinspektor. Eine didaktische Besonderheit besteht darin, dass er nicht wie bei professionellen IDEs üblich die komplette Liste aller Attribute anzeigt, sondern zunächst nur die wichtigsten Attribute, sodass die Schülerinnen und Schüler deutlich entlastet werden. Bei Bedarf kann man sich mehr bzw. alle Attribute anzeigen lassen. Platziert man mehrere Buttons auf einem Formular, so werden diese fortlaufend durchnummeriert, z.B. `pushButton1`, `pushButton2`, usw. Bei mehr als drei bis vier Buttons wird es dann im Quellcode unübersichtlich, weil die Nummerierung keinen Bezug zur Bedeutung eines Buttons hat. Da die Schülerinnen und Schüler jedem Button eine Beschriftung geben, wird dieses Problem didaktisch so gelöst, dass der Name eines Buttons aus der eingegebenen Beschriftung abgeleitet wird.

Der visuelle Entwurf eines GUI-Formulars mit Widgets wird automatisch als Pythoncode in die Methode `create_widgets` übertragen, wobei statt eines Layout-Managers das absolute Layout benutzt wird. Standardmäßig sind `create_widgets` und der Konstruktor eingeklappt, so dass der Programmcode für die Schülerinnen und Schüler leicht überschaubar bleibt und oft nur aus den Ereignismethoden besteht.

Diesem Beispiel einer Aufzug-Simulation liegt als Fachkonzept die Klasse `Aufzug` zugrunde. Sie muss importiert und im Konstruktor ein Aufzug-Objekt erzeugt werden, mit dem dann im Weiteren gearbeitet wird. Die Ein- und Ausgabe erfolgt über die Widgets, die Verarbeitung im Objekt `einAufzug` (EVA-Prinzip). Widgets stellen Methoden für die Ein-/Ausgabe bereit. Die wenigen relevanten Methoden sind auf der `GuiPy`-Homepage `gui.py.de` dokumentiert. Dort findet man auch hilfreiche Anwendungsbeispiele. Gemäß dem Model-View-Controller-Konzept aktualisiert die Ereignismethode `bWeiterfahren_clicked` als Controller das Modell `einAufzug` und zeigt anschließend über die Widgets im GUI-Formular als View mit `setText`, `setValue` und `setChecked` den aktuellen Zustand des Aufzugs.

## Debugging

Beim Programmieren der Ereignismethoden können leicht Fehler gemacht werden. Beispielsweise fehlerhafte Groß-/Kleinschreibung bei Variablen und Attributen, falsche Bezeichnung von Widgets, Aufrufe mit Datentyp `int` statt `String` oder vergessene Klammern (Henning, Michaeli, 2023). Da Python interpretiert und nicht compiliert wird, treten solche Fehler erst während der Programmausführung auf, also typischerweise wenn bei GUI-Programmen eine Ereignisprozedur ausgeführt wird. Die Fehlermeldungen des Python-Interpreters sind in der Regel nicht hilfreich.

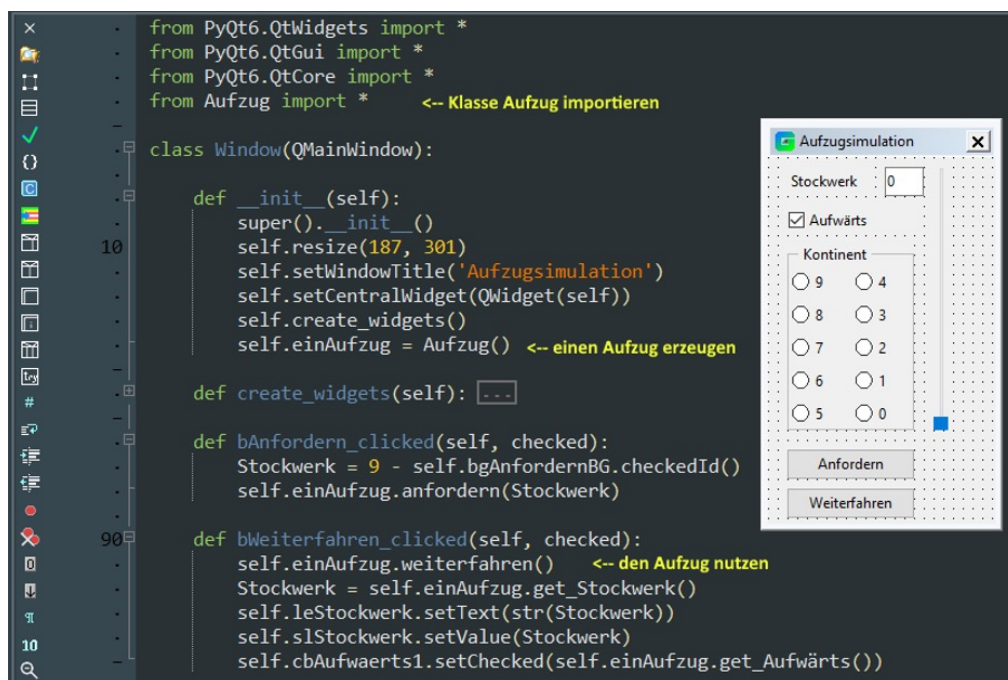


Abbildung 8: Ein- und Ausgabe in einem GUI-Formular

Man benötigt eine funktionierende Debuggingstrategie, um Fehler in GUI-Programmen zu finden und zu beheben. Dazu setzt man bei einer nicht funktionierenden Ereignismethode einen Haltepunkt auf die erste Anweisung nach dem Methodenkopf und führt dann das Programm im Debuggingmodus aus. Über die GUI-Oberfläche klickt man auf den Button, dem die Ereignismethode zugeordnet ist. Der Debugger hält dann das Programm am Haltepunkt an. Jetzt führt man schrittweise das Programm aus. Wird an der nächsten Anweisungszeile angehalten, war die vorherige fehlerfrei. Ansonsten enthält die Zeile, an der nicht mehr angehalten wurde, einen Fehler. Hier sollte man nach den typischen oben genannten Fehler Ausschau halten.

## Python und OOP

Die Realisierung des Klassenkonzepts in Python weist einige Schwächen auf, die man beim Programmieren bald bemerkt. Der Zugriff auf ein Attribut benötigt mindestens ein `self`. Setzt man das Geheimnisprinzip konsequent um und gibt allen Attributen die Sichtbarkeit `private` so muss jeder Zugriff auf ein Attribut sogar mit `self.__` eingeleitet werden, was aufwändig ist. Gemäß dem Zen von Python (Peters, 2004) gilt allerdings: *Explicit is better than implicit*. Mit der expliziten Angabe von `self` werden Kollisionen mit implizit angelegten lokalen Variablen oder mit gleichnamigen Parametern vermieden. In Java kann man in solchen Fällen `this` explizit benutzen, ansonsten kann man `this` weglassen.

In Python gibt es trotz *Explicit is better than implicit* keine Möglichkeit, Attribute explizit zu deklarieren. Sie werden implizit angelegt, wenn eine Wertzuweisung stattfindet. Das findet typischerweise im Konstruktor statt, kann aber

auch in einer normalen Methode passieren. Bei explizitem Anlegen würde man die Attribute einer Klasse direkt erkennen können. Außerdem könnte man die Sichtbarkeit festlegen und müsste diese nicht bei jedem Zugriff angeben.

Bei rekursiv definierten Klassen gibt es Probleme mit der Typisierung. In diesem Beispiel weist der Python-Interpreter darauf hin, dass der Typ `Tree` des Parameters `left` nicht definiert ist (van Rossum, 2014).

```
class Tree:
    def __init__(self, left: Tree, right: Tree):
        self.left = left
        self.right = right
```

Man kann dieses Problem durch die Anweisung `from __future__ import annotations` beheben, weswegen GuiPy diese Anweisung bei Bedarf automatisch ergänzt.

## KI-Schnittstelle

Künstliche Intelligenz wird das Programmieren auch in der Schule verändern, denn sie hat das Potential jedem Schüler und jeder Schülerin individuelle Rückmeldungen zu geben. Wenn man sich bei OpenAI oder Google Gemini einen API-Schlüssel erzeugt, kann man in GuiPy mit KI-Unterstützung arbeiten. Das Chat-Fenster ist eine in GuiPy integrierte Alternative zu einer browserbasierten KI-Nutzung. Der KI-Assistent steht im Quelltextfenster zur Verfügung. Markiert man Sourcecode, so hat man im Kontextmenü des Editorfenster diese drei Optionen: *Erkläre*, *Fehler beheben* und *Optimieren*.

Abbildung 9: KI-Assistent: Erklärung des markierten Sourcecodes

## Sonstiges

GuiPy basiert auf dem open source project PyScripiter, das Kiriakos Vlahos seit 2005 als Python-IDE entwickelt (Vlahos, 2024). Daher verfügt GuiPy über eine Fülle von Funktionen für die Programmierung u. a. Editor mit Syntaxhervorhebung, Code-Faltung, Codevervollständigung, Syntaxprüfung während der Eingabe, integrierter Interpreter und Debugger, Variablenfenster, externe Tools uvm. GuiPy erweitert PyScripiter um die in diesem Artikel dargestellten Modellierungswerkzeuge, welche vom JavaEditor (Röhner, 2024a) übernommen und an die Sprache Python angepasst wurden. Zusätzlich kann in GuiPy die Sichtbarkeit von Menüs, Registerkarten und Symbolen konfiguriert werden. Auch gibt es Unterstützung für die Installation und Konfiguration in einem schulischen Netzwerk.

Ein besonderer Dank gilt Peter Ehrlich, der mit seinen unermüdlichen Rückmeldungen zu Fehlern und Problemen maßgeblich zur Verbesserungen von GuiPy beiträgt.

## Quelltexte

Die im Beitrag referenzierten Quelltexte werden auf der ibis-Webseite zur Verfügung gestellt: <https://t1p.de/l25rl>



## Literatur

Alle Webseiten/Links wurden zuletzt geprüft am 01.08.2024.

Becker, K. (2023). GUI-Entwicklung mit tkinter. [https://inf-schule.de/software/gui/entwicklung\\_tkinter](https://inf-schule.de/software/gui/entwicklung_tkinter)

Bovermann, K. (2014). Graphen in der Schule mit Python und Jupyter-Notebooks. IBIS Jg. 2, Nr. 1

Eckerdal, A. (2004): On the understanding of Object and Class.

<https://www2.it.uu.se/research/publications/reports/2004-058/2004-058-nc.pdf>

Gesellschaft für Informatik, 2016: Bildungsstandards Informatik für die Sekundarstufe II

Hennig, H. & Michaeli, T. (2023). Entwicklung eines Prozessmodells für Diagnose- und Intervention von Lehrkräften beim Debugging, S. 157-166. In: Informatikunterricht zwischen Aktualität und Zeitlosigkeit, Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023

Jugendwettbewerb Informatik (2025).

Trainingsaufgaben Python. <https://jwinf.de/contest/training>

Peters, T. (2004). PEP 20 – The Zen of Python. <https://peps.python.org/pep-0020/>

Röhner, G. (2024a). Java-Editor. <https://javaeditor.org>

Röhner, G. (2024b). GuiPy. <https://gui.py.de>

van Rossum, G. (2014). PEP 484 – Type Hints. <https://peps.python.org/pep-0484/>

Vlahos, K. (2024). PyScripiter. <https://github.com/pyscripiter>

Weintrop, D. & Grover, S. (2020). JavaScript, Python, Scratch, or Soemthing Else? Navigating the Busting World of Introductory Programing Languages. In Grover, S.: Computer Science in K-12. An A to Z handbook on teaching programming. S.99-112.

## Lizenz



Dieser Artikel steht unter der Lizenz CC BY NC 4.0 zur Verfügung.

## Kontakt

Gerhard Röhner

[groehner@t-online.de](mailto:groehner@t-online.de)