

## B.IV.7

### Algorithmen – Objektorientierte Programmierung

# Einheit: Programmieren mit *Python* – Grundlagen und Erstellen eines Textadventures

Christina Hund



© RAABE 2024

© Maria Votonra/Stock/Getty Images Plus

In dieser Einheit erlernen Ihre Lernenden nicht nur die Grundlagen der Programmierung, sondern auch wichtige Funktionen der Sprache *Python*. Sie lernen anhand dieser textbasierten Programmiersprache die wichtigsten Befehle (Text-Eingabe, Variablen, Wenn/Dann-Funktionen und Schleifen) kennen und wenden diese *Python*-Grundfunktionen an. Schließlich festigen sie ihre Kenntnisse durch eine vertiefte Anwendung der *Python*-Grundfunktionen in der Entwicklung eines Textadventures als eigenes Spiel. Anhand eines Kriterienkatalogs werden die Entwicklungsergebnisse von den Mitschülerinnen und Mitschülern bewertet.

---

#### KOMPETENZPROFIL

<b>Klassenstufe:</b>	8/9
<b>Dauer:</b>	8 Unterrichtsstunden
<b>Lernziele:</b>	Die Lernenden ... 1. recherchieren Grundkenntnisse zur Programmierung, 2. wenden <i>Python</i> -Grundfunktionen an, 3. definieren die wichtigsten Befehle in <i>Python</i> , 4. entwerfen ein eigenes Textadventurespiel und setzen dessen Entwicklung in <i>Python</i> um.
<b>Thematische Bereiche:</b>	Programmierung, textbasierte Programmiersprache, <i>Python</i> , Computerspiel, Spielentwicklung, Algorithmen, Textadventure
<b>Kompetenzbereiche:</b>	Implementieren, Darstellen und Interpretieren, Produzieren und Präsentieren, Analysieren und Reflektieren

---



## Fachliche Hinweise

### Was sollten Sie zum Thema wissen?

Mit dieser Einheit werden die Grundlagen von *Python* vermittelt, sodass kein spezielles Vorwissen nötig ist. *Python* ist mittlerweile eine der führenden Programmiersprachen, was vor allem daran liegt, dass sie einfach zu lernen, aber schwer zu meistern ist. Sie ist vielseitig einsetzbar und deshalb in vielen Bereichen angesiedelt. Es ist hilfreich sich vorab als Lehrkraft mit der Programmiersprache *Python* zu beschäftigen, damit eventuelle Fehlerquellen schneller erkennbar sind. Hier kann auch Grundwissen aus anderen Programmiersprachen unterstützen.

### Welches Vorwissen sollten die Lernenden mitbringen?

Prinzipielle Programmierkenntnisse vonseiten der Lernenden sind nicht zwingend vorausgesetzt, aber definitiv eine große Stütze. Vor allem schwächere Lernende können davon profitieren zunächst mit einer *Blockly*-Sprache wie *Scratch* oder *Robot Karol* zu arbeiten. Auf dieser Grundlage aufbauend können die Schülerinnen und Schüler die visuellen Bausteine einfacher in Textprogrammierung übersetzen. Verwenden Sie hierfür beispielsweise unsere hierzu bereits veröffentlichten RAABE-Unterrichtsmaterialien, u. a.:

- Programmierung mit *Scratch* – Erste Schritte in der visuellen Programmierumgebung
- Erste kleine Programme in *Scratch* entwickeln – Einsatz als Einzelprojekte oder Stationenarbeit
- Programmierung mit *Scratch* – Eigene Spiele programmieren
- Programmieren eines *Scratch*-Spiels zum Klimawandel und was wir dagegen tun können
- Selbstlerneinheit: Lerne die visuelle Programmierumgebung *Robot Karol* kennen
- Einführung in die Programmierumgebung *Robot Karol* – Mit Selbstlernstationen

## Didaktisch-methodische Hinweise

### Vorbereitung

- Stellen Sie ausreichend Laptops/PCs/mobile Endgeräte im Klassenraum zur Verfügung, idealerweise ein Gerät pro Schüler/-in oder mindestens ein Gerät pro Schülerpaar.
- Sorgen Sie für die Bereitstellung von Internet im Klassenraum.
- Stellen Sie sicher, dass *IDLE3* als Entwicklungsumgebung auf den Endgeräten installiert ist.

**Hinweis:** *IDLE* steht für *Integrated Development and Learning Environment*. Es handelt sich dabei um eine leichtgewichtige integrierte Entwicklungsumgebung (IDE) für Installer der Programmiersprache *Python*. *IDLE* ist als einfache IDE ohne Überladung mit komplizierten Funktionalitäten gedacht, die sich auch für Anfängerinnen und Anfänger, gerade auch im Bildungsumfeld, eignet. Sie funktioniert plattformübergreifend unter *Windows*, *Unix* und *maxOS*.

### Benötigte Dateien

- ggf. Installationsdateien für *Python* und *IDLE3*: [python.org](https://python.org)
- *print\_input.py*
- *labyrinth\_1.py* & *labyrinth\_2.py*
- *notenabfrage.py* & *notenabfrage\_lsg.py*
- *rechner.py*

## Einstieg

Bevor in die textbasierte Programmierung eingetaucht wird, aktivieren die Lernenden ihr Vorwissen und finden die Grundlagen von Programmiersprachen heraus (**M 1**). Sollten Ihre Schülerinnen und Schüler schon eine visuelle Programmiersprache, z. B. *Scratch*, kennengelernt haben, ist es der ideale Einstieg über Programmierung allgemein zu sprechen. Ansonsten sammeln die Lernenden im Plenum Programmiersprachen, die sie kennen. Hierbei können sie auch Theorien aufstellen, wozu man eine Programmiersprache überhaupt braucht. Mithilfe der Fragen des Arbeitsblattes vertiefen sie dann ihre Kenntnisse.

## Erarbeitung 1: Grundlagen der *Python*-Programmierung

Nach dem Einstieg geht es direkt über zu den Grundlagen der Programmiersprache *Python*, indem die Lernenden den Steckbrief von **M 2 (Aufgabe 1)** ausfüllen. Besonders im Plenum zu besprechen ist der Vergleich von *Python* mit *Scratch* und *Java*. Denn Programmiersprachen können sehr unterschiedlich aussehen und komplex sein. Das Ergebnis ist immer „Hallo Welt!“, doch man sieht, dass der Weg dorthin sehr unterschiedlich sein kann.

Damit die Lernenden die grundlegende Syntax von *Python* kennen lernen, ist es sinnvoll, mit ihnen im Plenum den klassischen ProgrammierEinstieg zu zeigen: Die Ausgabe von „Hallo Welt!“. Der Befehl wird später genauer untersucht und dient hier vorrangig dazu die Struktur eines Befehls zu zeigen: Es gibt einen Befehl (hier: `print`) mit einem Inhalt, der in den Klammern steht. Da wir in den folgenden Aufgaben mit IDLE3, der offiziellen *Python*-Programmieroberfläche, arbeiten werden, finden die Lernenden anhand von **M 2, Aufgabe 2** heraus, wozu eine integrierte Entwicklungsumgebung (IDE) dienlich ist. Denn theoretisch kann man *Python* wie die meisten Programmiersprachen in jeglichen Text-Editoren schreiben, was aber vor allem im Lernprozess schwierig sein könnte.

Nach dem Sammeln der Ergebnisse aus **M 2** wird im Plenum die IDLE3-Oberfläche gezeigt. Hier ist es wichtig, dass es zwei Oberflächen gibt: Die Konsole und die Datei. Die Konsole kann nur einzelne Befehle ausführen, einen speicherbaren Programmiercode kann man nur über die Datei erreichen. Diese Präsentation kann auch dafür genutzt werden, die Farben in IDLE3 zu präsentieren und zu zeigen, wie Fehler ausgegeben werden, indem man wissentlich in einer Zeile eine fehlerhafte Funktion einbaut.

Damit die Lernenden jederzeit eine Hilfe zur Hand haben, erhalten sie die IDLE3-Anleitung **M 2a**. Hier werden auch häufige Fehler angesprochen.

## Erarbeitung 2: Ein- und Ausgabe von Text in *Python*

Nun erkunden die Lernenden die Grundfunktionen in *Python*. Nutzen Sie zur Erarbeitung der Ein- und Ausgabefunktion das Arbeitsblatt **M 3**. In der Konsole wird vorgeführt, wie mathematische Grundfunktionen auch ohne vorherige Programmierung möglich sind. Doch da die Lernenden in dieser Einheit hauptsächlich mit Text-Eingabe und -Darstellung arbeiten, muss man hier auf ein paar Dinge achten: Die richtigen Befehle (`print` und `input`) und die Deklaration von Text durch Anführungszeichen. Nachdem die Lernenden schrittweise durch die Beispiele zu den Befehlen geführt werden, schreiben sie den Text in *Python* um. Hier sind vor allem folgende Fehlerquellen zu beachten:

- Ein Textumbruch erfolgt über einen neuen `print`-Befehl.
- Alle Klammern müssen geschlossen werden.
- Alle Texte müssen mit Anführungszeichen derselben Art umschlossen sein (" oder ').



**Erarbeitung 3: Variablen in *Python***

Mit `input` haben wir zwar nun eine Eingabemöglichkeit, aber noch keine Möglichkeit diese Eingaben zu verarbeiten. Hierfür brauchen wir Variablen. Die Lernenden kennen den Begriff aus der Mathematik, wo er allerdings meist ausschließlich mit Zahlen verwendet wird. Als eine nützliche visuelle Unterstützung hierfür können einfache Boxen oder Kartons dienen. Den Lernenden werden Boxen mit verschiedenen Namen präsentiert, wie zum Beispiel „x“ oder „Antwort“. In diese kann man dann Murmeln legen um Zahlen zu symbolisieren. Dabei werden die Anweisungen klar verbalisiert:

- „x soll 2 sein!“ → Zwei Murmeln werden in die Box „x“ gelegt.
- „Zu x sollen 3 addiert werden.“ → Drei weitere Murmeln kommen in „x“, x ist dann 5.
- „Es werden 4 von x abgezogen.“ → Vier Murmeln werden entfernt, x ist dann 1.

Bei jedem Zwischenschritt geben die Lernenden wieder, welcher Wert aktuell in x liegt. Hiermit wird gezeigt, dass der Inhalt variabel und mathematisch berechenbar ist. Abschließend werden Wortkarten präsentiert und in die Box gelegt, damit ersichtlich ist, dass auch Buchstaben in einer Variable vorhanden sein können. Mit dem Arbeitsblatt **M 4** erhalten die Lernenden dann ein Beispiel aus *Python*, welches nach und nach im Plenum präsentiert werden kann. Anhand von Leitfragen wird der Code dann genauer überprüft.

Hier ist wichtig auf die Funktion `int` hinzuweisen. Denn bei der Verwendung von Zahlen ist dieser Befehl unumgänglich, da man sonst Gefahr laufen kann, dass *Python* die eingegebenen Zahlen nicht als Zahl, sondern als Text liest.

In **Aufgabe 2 von M 4** programmieren die Lernenden schließlich einen kleinen Rechner.

**Hinweis zur Binnendifferenzierung:** Stärkere Lernende können den Code natürlich erweitern: Mehrere Variablen oder mehrere Rechenwege können den Code ausbauen.

**Erarbeitung 4: Wenn/Dann-Abfragen in *Python***

Eine der wichtigsten Funktionen der Programmierung ist die Wenn/Dann-Abfrage. Bei *Python* heißt diese `if/elif/else` und kommt in unserem Fall bei dem Abgleich von Antworten zum Einsatz. Als Beispiel gibt es hier eine Notenabfrage (**M 5**). Je nach Note kommt eine andere Antwort. Vor allem ist hier ein Augenmerk auf „`==`“ zu setzen, denn der Abgleich einer Variable ist nicht „`=`“. In der Programmierung bedeutet „`==`“ ein Abgleich, „`=`“ würde den Wert einer Variable ändern. Die Lernenden schreiben dann selbst eine Notenabfrage mit den Kriterien des Arbeitsblattes.

**Hinweis zur Binnendifferenzierung:** Es können hier auch weitere Kriterien hinzugefügt werden: Wie sieht es zum Beispiel mit Viertelnoten aus?

**Festigung: Erstellen eines eigenen Textadventures**

Bevor es an das eigene Textadventure geht, bekommen die Lernenden als kleines Beispiel das Labyrinthspiel **M 6**. Sie erhalten hierfür die Datei `labyrinth_1.py`, welche funktioniert, aber ein großes Manko hat: Wenn man sich vertippt ist das Spiel direkt vorbei und man muss es umständlich neu laden. Mithilfe der Grundkenntnisse zu den Abfragen sammeln die Lernenden zunächst mit Partnern, dann im Plenum Möglichkeiten, wie man das umgehen könnte. Mit Abfragen kommt man aber schnell an die Grenzen, da man sehr oft mit einem `else` arbeiten müsste.

Schließlich erhalten sie das erweiterte `labyrinth_2.py`. Diesen Code testen die Lernenden aus und merken, dass das Spiel erst vorbei ist, wenn es auch vorbei sein sollte. Das wird mit der `while`-Schleife ermöglicht. Dies ist eine fortlaufende Schleife, die an einen Zustand geknüpft ist. In diesem Beispiel läuft die Schleife so lange, bis sie zu einem `break` kommt. So können viele umständliche Abfragen umgangen werden. Schließlich erkunden die Lernenden selbst die Funktionsweise und



können sich am Labyrinth versuchen: Wie kann man es erweitern? Was könnte noch passieren? Gibt es mehrere Enden?

Dies erleichtert den Einstieg in die Textadventures: Die Lernenden programmieren zu zweit oder in Kleingruppen anhand von **M 7** eine interaktive Geschichte. Sie beinhaltet nur Text, der Inhalt kann aber frei sein.

**Hinweise zur Binnendifferenzierung:** Zur Inspiration können sich die Lernenden bei [ifwizz.de](https://ifwizz.de) Spiele anschauen. Weitere Hilfen sind die Befehlsübersicht **M 8** und die Themenideen für Textadventures **M 7a**.

Schließlich werden die Projekte mithilfe des Kriterienkatalogs **M 7b** bewertet. Dies kann durch die Lehrperson geschehen oder durch Rückmeldung der anderen Gruppen. Damit die Ergebnisse gewürdigt werden können, präsentieren die Gruppen in einem Rundgang ihre Spiele. Hier sollte die Möglichkeit gegeben werden, dass jeder die anderen Spiele ausprobieren kann.



### Weiterführende Medien

- ▶ Pratzner, A. *Python* Kurs: Mit Python programmieren lernen für Anfänger und Fortgeschrittene: <https://www.python-lernen.de>
- ▶ W3Schools: *Python* Tutorial (englisch): <https://www.w3schools.com/python>

[Letzter Abruf aller Links am 07.05.2024]

### Erklärung zu den Symbolen



Dieses Symbol markiert differenziertes Material. Wenn nicht anders ausgewiesen, befinden sich die Materialien auf mittlerem Niveau.

## Auf einen Blick

### Benötigte Materialien und Dateien

- PC/Laptop/mobiles Endgerät mit Internetzugang und Installation von *IDLE3 für Lehrkraft und pro Lernenden oder pro Schülerpaar*
- ggf. Installationsdateien für Python und IDLE3: [python.org](https://python.org)
- print\_input.py*
- labyrinth\_1.py* & *labyrinth\_2.py*
- notenabfrage.py* & *notenabfrage\_lsg.py*
- rechner.py*

### Einstieg: Programmierung

Thema: Grundlagen der Programmierung

M 1 Wozu braucht man Programmiersprachen?

### Erarbeitung: Programmieren mit Python

Thema: Python-Grundlagen

M 2 Grundlagen der Programmiersprache Python

M 2a IDLE3-Anleitung

M 3 Python: Ein- und Ausgabe von Text mit den Befehlen print und input

Benötigt:  *HTML\_Tags.html*

M 4 Python: Variablen

Benötigt:  kleine Kartons, Karten

*rechner.py*

M 5 Abfragen mit Python: Wenn, dann, sonst ...!

Benötigt:  *notenabfrage.py*

### Festigung: Textadventures

Thema: Ein eigenes Spiel gestalten

M 6 Python: Die Spiel-Schleife

Benötigt:  *labyrinth\_1.py*

*labyrinth\_2.py*

M 7 Erstelle dein eigenes Textadventure mit Python

M 7a Themenideen für Textadventures /Hilfekarte

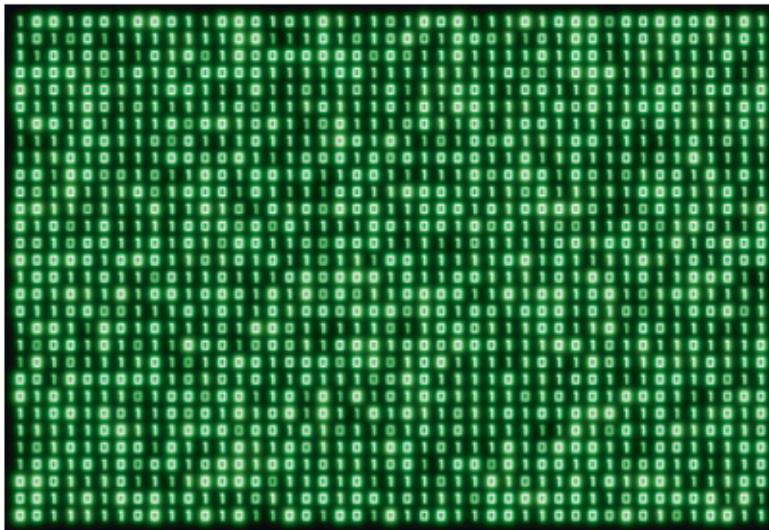
M 7b Kriterienkatalog zur Bewertung von Textadventures

M 8 Übersicht der wichtigsten Python-Befehle



## Wozu braucht man Programmiersprachen?

M 1



© Flavio Coelho/Moment

Computer haben ein sehr kleines Vokabular: Sie kennen nur 1 und 0. Wörter verstehen sie nicht. Doch wie bekomme ich meinen Computer dazu überhaupt etwas zu tun? Muss ich dann in 1en und 0en mit ihm sprechen?

Theoretisch: Ja. Praktisch: Zum Glück gibt es Programmiersprachen! Programmiersprachen dienen dazu eine Sprache zu bieten, die sich in 1en und 0en übersetzen kann. So kann der Computer durchaus auch Anweisungen in Wortform verarbeiten.

### Aufgabe

Recherchiert im Internet zu der Frage: Was macht eine Programmiersprache aus?

1. Welche Grundfunktionen haben die meisten Programmiersprachen? Nenne vier Stück.

---

---

2. Was ist ein „Compiler“?

---

---

3. Welcher Text wird bei den meisten Programmiersprachen als Einstieg ausgegeben?

---



## *Python*: Ein- und Ausgabe von Text mit den Befehlen `print` und `input`

M 3

### Vorinstallierte Programme in *Python*

Prinzipiell können Programme komplett ohne Ausgabe funktionieren. *Python* hat hier auch ein paar einfache Grundfunktionen, denn Mathematik muss man nicht erst definieren. Deshalb kann man in der Konsole direkt rechnen, z. B.:

```
>>> 3+2
5
>>> 3-2
1
```

### Befehl `print`: Text auf den Bildschirm drucken

Manchmal möchte man auch Text ausgeben. Einfach nur mit der Eingabe des Textes scheint es aber nicht zu funktionieren:

```
>>> Hallo Welt!
SyntaxError: invalid syntax
```

Hierfür gibt es den Befehl `print()`. Alles in den Klammern wird ausgegeben, jedoch muss man bei Text immer noch genau aufpassen. So funktioniert es trotzdem nicht:

```
>>> print(Hallo Welt!)
SyntaxError: invalid syntax
```

Denn Text ist nicht gleich Text für eine Programmiersprache. Wenn es sich um Text handelt, muss man diesen mit Anführungszeichen kennzeichnen.

```
>>> print(„Hallo Welt!“)
Hallo Welt!
```

### Befehl `input`: Frage? Antwort!

Manchmal möchte man auch eine Frage stellen, die die Eingabe einer Antwort benötigt. Hierfür kann man den Befehl `input()` verwenden, z. B.:

```
>>> input(„Hallo Welt?“)
Hallo Welt? Ja klar! |
```

### Aufgabe

Probiere nun selbst den folgenden Text mit `print` und `input` darzustellen. Achte auf die Textumbrüche.

```
Hallo Programmierer!
Wie heißt du? (EINGABE)
Das ist ein toller Name.
Wie alt bist du? (EINGABE)
Oha, noch so jung?
Dann hast du ja noch viel Zeit etwas über Python zu lernen!
```