1 Semester-Programm "Einführung Programmieren": Begleitdokument für Lehrpersonen

Dieses Dokument hat zum Ziel, einen Semester-Plan für die Einführung in das Programmieren auf Gymnasial-Stufe (Schweizer Bildungssystem, Sekundarstufe II, Altersstufe ca. 15-18 Jahre, obligatorisches Fach Informatik, kurz OInf) mittels Python zur Verfügung zu stellen. Das Semesterprogramm setzt keine besonderen Programmier- oder Informatikkenntnisse voraus. Das Semesterprogramm erstreckt sich über ca. 10 Lektionen und sollte den SuS ermöglichen, einfache mathematische und grafische Programme in Python zu schreiben.

Da relativ wenig Zeit für das OInf zur Verfügung steht, liegt der Fokus dieses Semester-Programms auf grundlegenden Python-Kenntnissen. Als Fortsetzung und Vertiefung der neu gewonnen Kenntnisse empfiehlt sich eine Einführung in die Robotik im OInf, beziehungsweise eine Vertiefung der Programmierung im Ergänzungsfach, beispielsweise mit Themen wie Rekursion oder objektorientierter Programmierung (in Python).

Beim untenstehenden Themenüberblick wurde insbesondere darauf geachtet, die Reihenfolge möglichst sinnvoll zu gestalten, so dass die Kenntnisse schrittweise erweitert und aufeinander aufgebaut werden können. Zudem wurde darauf geachtet, stets ein möglichst ausgeglichenes Verhältnis zwischen "theoretisch-mathematischen" und "grafisch-interaktiven" Aufgaben zu bewahren (da letztere die SuS erfahrungsgemäss stärker motivieren).

Überblick der Themen

1. **Allgemeine Motivation und Einführung ins Programmieren**
   - 1-2 Beispiele mit Bezug zur Lebenswelt und Vorwissen von SuS, einige Algorithmen und Beispiele zu logischem Denken
   - Vordefinierte Grafik-Befehle mit `turtle` (nur ein Parameter): `forward()`, `right()`
   - Einfache Schleifen mit `for i in range(...)`
2. **Datentypen und Syntax**
   - Variablen und Zuweisung vs. Mathematik
   - Arbeiten mit einfachen mathematischen Operatoren
   - Arbeiten mit Text
   - Debugging-Übungen (syntax vs. runtime vs. semantic errors), typische Fehler
3. **Verzweigungen** (`if...elif...else`) & `while`, elementare logische Operatoren (or, and, xor) und Flussdiagramme
4. **Befehle / Funktionen** (`def`)
   - Ohne Parameter, ohne `return`
   - Mit Parameter, ohne `return`
   - Mit Parameter, mit `return`
5. **Listen**
   - Koordinatengrafik, Zufallszahlen, `random`-Library (→ `turtle`),
   - Schleifen / Traversieren (`for item in mylist`)

Mögliche Schwierigkeiten, die Auftreten können:

1. **Zeitbedarf**: Übungen zu neuen Teilen könen erfahrungsgemäss viel Zeit beanspruchen, insbesondere Übungen zu Syntax oder Funktionen. Dabei kann es hilfreich sein, die Übungen von Anfang an mit einer Zeiteinschätzung zu versehen und einzuplanen, welche Übungen bei Bedarf gestrichen, beziehungsweise auf eine andere Lektion verschoben, oder als Challenge-Aufgabe für Schnelle deklariert werden sollen.
2. **Abstraktes Denken**: Gerade in Bereichen wie Verzweigungen und Flussdiagrammen, oder beim Traversieren von Listen, zeigt sich, dass das abstrakte Denken — insbesondere in tieferen Altersstufen — häufig noch nicht ausreichend entwickelt ist. Daher kann es, sofern diese Möglichkeit besteht, sinnvoll sein, das Semesterprogramm erst in einem späteren Semester auszuführen.
3. **Technische Probleme**: Shortcuts, Programme installieren, Programme abspeichern — all dies ist für die viele SuS entgegen der Erwartung vieler LP noch neu. Dazu kommen häufige Probleme beim Betreiben eigener Geräte (BYOD). Daher sollte genügend Zeit eingeplant werden, um auch auf solche Faktoren einzugehen und um "softes" Wissen zu vermitteln.

Inhaltsverzeichnis

2 DL 1: Allgemeine Motivation und Einstieg ins Programmieren

DL 1

Thema

Die Lektion soll den Einstieg in die, für die meisten SuS noch neue Welt des Programmierens machen. Dabei werden zuerst das Thema "Programmieren" im Allgemeinen und danach Python als spezifische Programmiersprache für diesen Semester-Kurs vorgestellt. Abschliessend führen einige einfache grafische Beispiele mit der `turtle`-Library die SuS an die Programmierung mit Python heran.

Operationalisierte Lernziele

- ☐ Die SuS verstehen, dass viele alltägliche Abläufe durch Algorithmen, d.h. automatisierte und logische Abläufe, modelliert und beeinflusst werden.
- ☐ Die SuS können einige Beispiele aus dem Alltag nennen, wo Algorithmen eine Rolle spielen.
- ☐ Die SuS verstehen grundsätzliche Elemente von Flussdiagrammen und können logische Prozesse aus der ihnen bekannten Alltagswelt damit modellieren.
- ☐ Die SuS haben Python sowie eine IDE installiert, mit welcher sie Python-Programme abspeichern und ausführen können.
- ☐ Die SuS können elementare Befehle mit der `turtle`-Library ausführen, um damit einfache Zeichnungen zu erstellen
- ☐ Die SuS können einfache Formen mit einer `for i in range(start, end, step)`-Schleife zeichnen, beispielsweise ein Quadrat oder ein Sechs-Eck.

Grober Lektionsablauf

Lektion 1/2: Programmieren & Algorithmen

Erste Hälfte der Lektion: Die SuS werden an das Thema Programmieren herangeführt und lernen dessen Relevanz anhand einiger alltagsnaher Beispiele kennen: Messgeräte in Spitäler (unterliegende Idee: `if...elif...else`), Sensoren in Autos (z.B. `while`), etc. Dabei lernen die SuS, die unterliegenden Algorithmen als Flussdiagramme dazustellen. Zweite Hälfte: Die SuS lösen eigenhändig Übungen und wenden ihr neu gewonnenes Wissen zu Flussdiagrammen auf die Übungen an (Abgabe auf Moodle).

Lektion 2/2: Programmieren mit Python und `turtle`

Die SuS lernen Python kennen und installieren ein lokales IDE. Danach führen sie, geleitet durch die LP, einige Übungen durch, um einfache Formen mit der Schildkröte zu zeichnen. Dabei lernen Sie die erste einfache Schleife (`for i in range(start, end, step)`) kennen.

Mögliche Schwierigkeiten & geeignete Massnahmen

1. Abstraktes Denken zu Flussdiagrammen ist komplex und herausfordernd. Der wäre es hilfreich, bereits zu Beginn des Unterrichts ein Nachschlagewerk zur Verfügung zu stellen, welches die SuS dabei unterstützt, die Übungen zu lösen, indem sie bei Bedarf nachlesen können.
2. BYOD als Herausforderung: Möglicherweise gestaltet sich der Installationsprozess komplexer und langwieriger als gedacht. Daher sollte die Installation des IDE nach Möglichkeit bereits im Voraus als Hausaufgabe (mittels Schritt-für-Schritt-Dokumentation) ausgeführt worden sein.
3. Technische Herausforderungen beim Ausführen von Code: Gerade in der ersten Lektion brauchen einige SuS noch Unterstützung beim Ausführen und Debuggen von Code ("wo ist der Run-Knopf?"). Mögliche Lösungsansätze könnten sein, genügend Zeit für die Übungen einzuplanen (s. Punkt 2), bzw. den Einstieg nach Möglichkeit in Halbklassen zu machen.

3 DL 2: Datentypen und Syntax (Variablen & Zahlen)

DL 2

Thema

In dieser Doppellektion geht es darum, die SuS an die Grundlagen von Datentypen sowie einige grundlegende syntaktische Elemente in Python heranzuführen. Dabei soll, in der ersten der beiden dafür vorgesehenen Doppellektionen, der Fokus auf mathematische Operationen gelegt werden, wodurch die SuS hoffentlich auch einen Bezug zu ihrem Vorwissen herstellen können.

Operationalisierte Lernziele

- ☐ Die SuS können das Gleichzeichen verwenden, um Variablen zu erstellen. Sie verstehen zudem den Unterschied zwischen dem Gleichzeichen in Python und dem Gleichzeichen in der Mathematik.
- ☐ Die SuS können einfache Rechnungen machen, indem Sie Variablen erstellen und einfache mathematische Operatoren verwenden.
- ☐ Die SuS können den Inhalt einer Variable `x` mit `print(x)` ausgeben.
- ☐ Die SuS können den Modulo-Operator (`%`) in sinnvoller Weise verwenden, beispielsweise um alle geraden Zahlen in einer Schleife auszugeben (s. DL 1 für einfache Schleifen).
- ☐ Die SuS können Speicherinhalte verändern, indem Sie *compound operators* in einer einfachen Schleife verwenden, also beispielsweise `+=`, `-=`, `*=` oder `/=`.
- ☐ Die SuS können Funktionen, um die Wurzel einer Zahl (z.B. `sqrt(x)`) oder deren Quadrat (z.B. `x**2`) sinnvoll verwenden und das Resultat in einer Variable speichern sowie ausgeben.
- ☐ Die SuS können einfache Formen mit Pythagoras zeichnen (beispielsweise ein schräges Dach in einem Haus)
- ☐ Die SuS können Probleme aus der Mathematik wie etwa polynomiale Gleichungen in einen Python-Code transformieren und diese damit lösen.

Grober Lektionsablauf

Lektion 1/2: Einführung und Übungen

Einfache Beispiele werden aufgezeigt, um das Interesser für mathematische Operatoren mit einigen alltagsnahen Beispielen einzuführen. Beispielsweise könnte aufgezeigt werden, wie aus einer Liste von Grössen diejenigen ausgegeben werden können, deren Körpergrösse gerade ist. Da die Doppellektion eher voll beladen ist, wird diese Sequenz jedoch relativ kurz gehalten, um genug Zeit für die Übungen zu lassen. Die Übungen sind, wie immer, im Skript, und die Erklärungen zu den neuen Konzepten, welche mit dem einführenden Beispiel vermittelt wurden, können dort bei Bedarf nochmals nachgelesen werden.

Lektion 2/2: Übungen

Lektion 2 widmet sich hautpsächlich den Übungen. Am Schluss wird, je nach dem welche Schwierigkeiten bei den SuS während den Übungen beobachtet wurde, nochmals auf einige herausfordernde Aufgaben eingegangen, und danach wird eine Ergebnissicherung (neue Konzepte) durchgeführt.

Mögliche Schwierigkeiten & geeignete Massnahmen

1. Die SuS könnten verwirrt sein, weshalb mit dem Gleichzeichen nun etwas anderes gemeint ist als in der Mathematik (Zuweisung statt Gleichheit). Dies könnte durch klare grafische Darstellungen gelöst werden, etwa wie folgt: `a` = `10+3` Die Erklärung dazu könnte lauten: "Wir werten `10+3 = 13` aus und speichern den Wert `13` in der Variable `a`".
2. Die Lektion könnte zu kurz sein für alle Inhalte. Allenfalls müsste nach einer ersten Durchführung entschieden werden, ob Teile der Lektion in DL 3 durchgeführt werden, welche aktuell daher etwas weniger umfangreich geplant ist.

4 DL 3: Datentypen und Syntax (Text & Debugging)

DL 3

Thema

In dieser zweiten, den Datentypen und Grundlagen sowie Syntax gewidmeten Doppellektion geht es darum, elementare Text-Operationen kennezulernen und diese sinnvoll zu verwenden. Ebenfalls soll sichergestellt werden, dass die SuS geläufige Laufzeit-Probleme aufgrund von Datentypen selbstständig entdecken und beheben können.

Operationalisierte Lernziele

- ☐ Die SuS wissen, was eine Zeichenkette in Python ist und können eine solche selbstständig erstellen.
- ☐ Die SuS können Zeichenketten mit `"..." + "..."` verketten.
- ☐ Die SuS können Zeichenketten mit `"..." * [zahl]` beliebig viele Male wiederholen.
- ☐ Die SuS können mithilfe der genannten Befehle, Spezialzeichen (z.B. `\n`) sowie `print()` einfache ASCII-Zeichnungen machen, wie etwa einen Pfeil.
- ☐ Die SuS können Zeichenketten sowohl innerhalb einfacher wie doppelter Klammern (`'` oder `"`) schreiben und verstehen wozu dies dient (*escape character*, etwa in folgendem Befehl: `print('Meine Freunde nennen mich "Mr. X"')`.
- ☐ Die SuS können geläufige Laufzeit-Probleme wie etwa die nicht funktionalen Ausdrück `print("1+2")` oder `print(Hallo Welt)` selbstständig als solche erkennen und beheben.
- ☐ Die SuS könenn die Funktion `input("Fragetext")` verwenden, um einen Wert (Text oder Zahl) während der Laufzeit zu generieren. Diesen können die SuS in einer Variable abspeichern und weiterverwenden.

Grober Lektionsablauf

Ähnlich wie die vorherige Doppellektion beginnt auch diese Doppellektion mit einigen alltagsnahen Beispielen, die direkt im Code-Editor vorgezeigt und ausgeführt werden. Beispielsweise könnte mit `input` nach einem Namen gefragt werden, der dann in der Konsole 20 mal als ASCII-Zeichnung ausgegeben wird. Nach einer relativ kurzen Einführungssequenz wird der Rest der Doppellektion den Übungen sowie einer Ergebnissicherung gewidmet, wobei letztere insbesondere nochmals Debugging-relavante Aufgaben aufgreift (welche im Skript an letzter Stelle dieser Doppellektion vorkommen).

Mögliche Schwierigkeiten & geeignete Massnahmen

1. Die SuS könnten unter umständen verwirrt sein, weshalb `turtle`-Grafiken in einem separaten Fenster gezeichnet werden, währenddem der `print()`-Befehl in der Konsole ausgegeben wird. Darauf sollte in der Einführung eingegangen werden.
2. Die SuS könnten Mühe haben mit dem Abstraktionsgrad von Zeichenketten (Wiederholungen etc.) und durch die Syntax verwirrt sein (beispielsweise die Möglichkeit, Zeichenketten mit einem einfachen oder doppelten Anführungszeichen zu schreiben). Diesen Aspekten sollte daher sowohl während der Präsentation wie auch im Skript besondere Aufmerksamkeit gewidmet werden.

5 DL 4: Verzweigungen mit `if...elif...else`

DL 4

Thema

Die SuS lernen das Programmieren mit konditionaler Logik kennen, indem Sie mit den Befehlen `if`, `elif` und `else` vertrautgemacht werden. Dabei wird nochmals das in DL 1 angeeignete Wissen zu Flussdiagrammen aufgefrischt, um die neuen Konzepte grafisch (im Sinne der *dual-code*-Theorie) zu unterstützen und aufzuwerten.

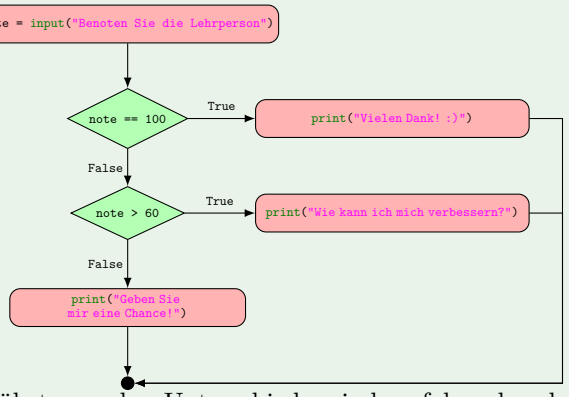
Operationalisierte Lernziele

- ☐ Die SuS können konditionale Logik mittels den Befehlen `if`, `elif` und `else` anwenden, um Befehle nur unter gewissen Bedingungen auszuführen
- ☐ Die SuS verstehen die Bedeutung der Einrückung in Python
- ☐ Die SuS können einen gegebenen Code als Flussdiagramm veranschaulichen
- ☐ Die SuS können erklären, weshalb ein bestimmter Code (nicht) ausgeführt wird.
- ☐ Die SuS können logische Tests formulieren, indem Sie die Operatoren `!=`, `<`, `<=`, `>=`, `>` und `==` korrekt auf Zahlen oder Text anwenden.
- ☐ Die SuS können logische Tests korrekt negieren, indem sie den Operatoren `not` verwenden.
- ☐ Die SuS können zwischen der Negation und dem Gegenteil einer (logischen) Aussage unterscheiden, indem sie eigene, alltagsnahe Beispiele erstellen.

Grober Lektionsablauf

Ähnlich wie die vorherigen Doppellektionen beginnt auch diese Doppellektion mit einigen alltagsnahen Beispielen, die direkt im Code-Editor vorgezeigt und ausgeführt werden. Dabei wird parallel zum Code ebenfalls eine Visualisierung des Codes als Flussdiagramm gezeigt, wie folgt:

```
1 note = input("Benoten Sie die Lehrperson
      (0-100).")
2 if note == 100:
3     print("Vielen Dank! :)")
4 elif note > 60:
5     print("Wie kann ich mich verbessern?")
6 else:
7     print("Geben Sie mir eine Chance!")
```



Das Beispiel würde aber Schritt für Schritt eingeführt, um den Unterschied zwischen folgenden drei Dingen klar aufzuzeigen: 1. `if` *ohne* `elif` und *ohne* `else` 2. `if` *ohne* `elif` aber *mit* `else` 3. `if` *mit* `elif` und *mit* `else`. Ebenfalls können weitere alltagsnahe Beispiele zumindest konzeptuell aufgezeigt werden, um aufzuzeigen, wozu die konditionale Logik dienen kann (medizinische Bereiche, selbstfahrende Autos etc.). Danach üben die SuS das neu gewonnene Wissen, indem Sie es auf Zeichenketten, Texte und mit `turtle` anwenden. Am Schluss findet wie immer eine Ergebnissicherung statt.

Mögliche Schwierigkeiten & geeignete Massnahmen

1. Die SuS finden es schwierig zu erkennen, ob Code nach einem `elif` oder `else` noch ausgeführt wird, da sie nicht verstehen, dass diese Blöcke nur dann ausgeführt werden, wenn vorher das `if False` war. Im Unterricht sollten daher einige Beispiele aufgezeigt werden, die dies veranschaulichen
2. Die Negation einer Aussage ist nicht dasselbe wie deren Gegenteil. Dies sollte anhand von alltagsnahen Beisielen (z.B. der Aussage “der Schrank ist voll”) aufgezeigt werden.

6 DL 5: Verzweigungen mit `while`

DL 5

Thema

In dieser Doppellektion soll die `while`-Schleife eingeführt werden, die gewissermassen die bisher gewonnenen Kenntnisse zu Schleifen sowie konditionaler Logik kombiniert. Dabei wird das Augenmerk wieder vermehrt auf "unterhaltsame", grafische Übungen gelegt, wie etwa dem Zeichnen von Spiralen mit `turtle` und mit dem `while`-Befehl. Als gegenüber zur "kopfgesteuerten" `while`-Schleife wird der Befehl `break` eingeführt, sozusagen als "fussgesteuerte" Schleife. Die Visualisierung von `while`-Schleifen als Flussdiagramm hilft, den Code besser zu verstehen.

Operationalisierte Lernziele

- ☐ Die SuS können `while`-Schleifen, bzw. `repeat`-Befehle verwenden, um Schleifen zu schreiben, die unter bestimmten Konditionen enden
- ☐ Die SuS können erklären, weshalb ein Code der nach einem ausgeführten `repeat`-Ausdruck steht, nicht mehr ausgeführt wird
- ☐ Die SuS können für ein einfaches, gegebenes Beispiel bestimmen, wie häufig eine bestimmte `while`-Schleife ausgeführt wird.
- ☐ Die SuS können die geeignetste Schleifenart auswählen, um ein bestimmtes Problem zu lösen (kopf- oder fussgesteuerter Abbruch, bzw. `for`-Schleife mit vorgegebener Anzahl Wiederholungen)
- ☐ Die SuS können Flussdiagramme um die neu erlernten Schleifentypen erweitern.

Grober Lektionsablauf

Als Einführung in die Lektion werden einige Beispiele ausgewählt, die die Schwierigkeiten, bzw. Komplexitäten gewisser Aufgaben *ohne* `while`-Schleife aufzeigen. Beispielsweise könnte dies das Zeichnen einer Schleife sein, *solange* die Seitenlänge unter, bzw. über einem gewissen Wert liegt. Das gegebene Problem soll einem als `for`-Schleife ohne Abbruch (mit berechneter Anzahl Wiederholungen) gelöst werden, einmal mit `while` und einmal mit `repeat`. Jede Lösung wird als Flussdiagramm aufgezeichnet. Die SuS sollen sich überlegen, welche der Lösungen am geeignetsten ist für ihr Problen. Wie in den Vorgängerlektinonen wird danach der Grossteil der DL den Übungen gewidmet.

Mögliche Schwierigkeiten & geeignete Massnahmen

1. Die SuS finden es schwierig, zwischen einem `break`- und einem `while`-Konstrukt zu unterscheiden, bzw. die richtige Lösung für die richtige Situation auszuwählen. Zur Linderung dieses Problems könnte beitragen, dass die SuS zuerst eine Reihe von einfachen Beispielen von der einen in die andere Variante umgeschrieben werden müssen. Dabei sollen sie sich stets überlegen, was sie bevorzugen. Ihre Antworten fliessen in die Diskussion vor der Ergebnissicherung ein.
2. Die SuS verstehen nicht, wie sich eine Variable innerhalb einer `while`-Schleife verändert. Dazu sollen sie spezielle Tabellen anfertigen, welche die Entwicklung der Variable in jedem Durchlauf der Schleife veranschaulichen.

7 DL 6: Befehle / Funktionen ohne Parameter, ohne `return`

DL 6

Thema

Diese Doppellektion soll als Einführung in die modulare Programmierung dienen. Dabei soll klargestellt werden, dass wir für das Schreiben eines komplexen Programms zunächst kleinere "Bausteine" benötigen, die zum Lösen von Teilproblemen nötig sind. Die verschiedenen Programme, die sie bisher geschrieben haben, sollen somit für den zukünftigen Gebrauch als Befehle festgehalten werden. Für die Einführung verwenden wir weder Parameter noch das `return`-statement.

Operationalisierte Lernziele

- ☐ Die SuS können mithilfe der `turtle`-Library einen eigenen Befehl ohne Parameter schreiben und aufrufen, der eine einfache geometrische Form (bspw. Dreieck, Quadrat usw.) erzeugt.
- ☐ Die SuS können anhand eines vorgegeben Befehls herleiten, was nach dem aufrufen des Befehls auf dem Bildschirm zu sehen ist. Der vorgegebene Befehl kann folgenden Aufbau haben: Sequentielle Folge von einfachen `turtle`-Befehlen, wie bspw. `forward()`, `left()`, `right()`. `while` -Schleifen oder Verzweigungen, die einfache `turtle`-Befehle beinhalten.

Grober Lektionsablauf

Lektion 1/2: `def`-Statement

Zu Beginn das Konzept der Schleifen in Erinnerung rufen (als Automatisierung eines sich wiederholenden Vorgangs), Übertragung dieses Konzeptes auf den Aufbau des Codes ("Bündelung von vorprogrammierten Befehlen durch die Nutzung des `def`-Statements, um Wiederholungen zu vermeiden".) Anschliessend sollen die SuS bereits verwendete Befehlsfolgen von `turtle`-Befehlen aus den vorherigen Lektionen in einen neuen, eigenen Befehl zusammenfassen und aufrufen.

Lektion 2/2: Verdichtung

Die SuS sollen zunächst neue Befehle schreiben, die sich von den bisher im Unterricht behandelten Programmen unterscheiden. Prinzipiell sollen sie sich zunächst mit einfach geometrischen Formen auseinandersetzen. Im Anschluss an diese Übung erhalten die SuS verschieden vorgegebene Befehle und sollen diese auf ihren Zweck untersuchen. Ihre Vermutungen können sie durch das Ausführen des Befehls überprüfen.

Mögliche Schwierigkeiten & geeignete Massnahmen

1. Es könnte hier leicht zu Problemen mit der Verwendung von Parametern kommen. Durch den Einsatz von bereits existierenden `turtle`-Befehlen wissen sie bereits, dass man Parameter in den Klammern eines Befehls eingeben muss, bzw. kann. Daher könnte es für sie naheliegend sein, dies auch beim Aufrufen eines eigenen Befehls zu versuchen. Die LP könnte bereits andeuten, dass das Verwenden von Parametern in unserem Fall noch nicht möglich ist, bzw. klarstellen, dass bei den eigenen Befehlen die Klammer leer sein muss.
2. Eine weitere Schwierigkeit wird die Syntax sein (fortlaufendes Problem bei der Einführung der Programmierung). In Python muss die Verschachtelung des Codes strikt eingehalten werden, d.h., nach dem Doppelpunkt muss die nächste Zeile einen zusätzlichen Abstand enthalten. IDEs wie Thonny, machen diesen Abstand in der Regel automatisch.
3. Zudem könnte der Unterscheide zwischen der Definition und dem Aufrufen des Befehls ein Problem darstellen: Bspw. könnte ein SuS versuchen, den Befehl aufzurufen, bevor er ihn definiert hat, bzw. das Statement `def` nutzen wollen, um den Befehl aufzurufen, oder nach dem Aufrufen des Befehls einen Doppelpunkt platzieren. Die Syntax muss daher besonders deutlich erklärt werden.

8 DL 7: Befehle / Funktionen mit Parameter, ohne `return`

DL 7

Thema

Die bereits erworbenen Erkenntnisse zu den Befehlen soll erweitert werden, indem man Parameter einführt. Zunächst werden Ein-Parameter-Befehle und danach Mehr-Parameter-Befehle behandelt. In dieser Doppellektion gehen wir noch nicht auf das `return`-Statement ein.

Operationalisierte Lernziele

- ☐ Die SuS können den Unterschied zwischen Parametern und Variablen in eigenen Worten erklären.
- ☐ Die SuS können eigene Befehle mit einem Parameter schreiben und aufrufen, um einfache Tätigkeiten auszuführen (z.B. eine einfache geometrischen Form zeichnen oder einfache mathematische Berechnungen durchführen und ausgeben).
- ☐ Die SuS können einen vorgegebene Befehl mit einem Parameter auf Fehler überprüfen und diese, falls nötig, beheben.
- ☐ Die SuS können einen eigenen Befehl mit mehreren Parameter schreiben und aufrufen.

Grober Lektionsablauf

Lektion 1/2: Ein-Parameter-Befehle

Beispiel: Zwi Codes, für ein 4- und ein 6-Eck, unterscheiden sich nur im verwendeten Winkel und in der Anzahl Schleifenwiederholungen. Um Code kürzer und übersichtlicher zu schreiben, führen wir den Begriff des Parameters ein. Die SuS erhalten als erste Aufgabe mehrere Code-Beispiele, die sie selbst implementieren und ausführen sollen. Die Resultate sollen sie miteinander vergleichen, um die Unterschiede zwischen Parameter und Variablen, die sich ausserhalb des Befehls befindet, zu ergründen.

Lektion 2/2: Mehr-Parameter-Befehl

In der zweiten Lektion arbeiten SuS weiter an den Beispielen aus der ersten Lektion (Vertiefung). Zudem sollen Sie nochmals das *debugging* vertiefen, indem Sie vorgegebene Befehle auf ihre Funktionalität überprüfen und Programm-Fehler beheben. Im Anschluss wird die Möglichkeit, mehr als einen Parameter zu verwenden, angesprochen und anhand von Übungen umgesetzt.

Mögliche Schwierigkeiten & geeignete Massnahmen

1. Die SuS könnten die Parameter eines Befehls möglicherweise mit einer Variablen ausserhalb des Befehlskörpers verwechseln, bspw. indem sie den Parameter nach der Definition versuchen mit `print()` anzugeben oder indem sie zuerst eine Variable definieren und diese anschliessend als Parameter verwenden möchten. Hier könnte man gezielt Übungen einsetzen, die den Unterschied zwischen “allgemeinen” Variablen und Parametern aufzeigen: Bspw. könnten die SuS unterschiedliche Programme ausführen und die jeweiligen Outputs miteinander vergleichen. Anhand dieses Vergleiches sollen sie herleiten, worin sich Parameter von anderen Variablen unterscheiden.
2. Man könnte den Inhalt der ersten Lektion gut auf eine Doppellektion ausweiten und die Befehle mit mehreren Parametern in der darauffolgenden Doppellektion behandeln oder sogar weglassen. Denn die verschiedenen Datentypen könnten noch eine weitere Schwierigkeit für die SuS darstellen: Die SuS könnten möglicherweise beim Aufrufen eines Befehls, der arithmetische Operatoren verwendet, dem Parameter einen String übergeben. Auch hier könnte man weitere Übungen zur Verfügung stellen, um den SuS diese Problematik aufzuzeigen.

9 DL 8: Befehle / Funktionen mit Parameter, mit `return`

DL 8

Thema

In dieser Doppellektion geht es um die Definition einer Funktion. Hierbei ist lediglich das Konzept des `return`-Statements neu. Dabei soll ein Bezug zum Begriff der Funktion aus der Mathematik hergestellt werden. Eine mögliche Umsetzung, um das ganze etwas graphisch zu gestalten, ist, dass man den SuS ein GUI mit einem Taschenrechner vorgibt und die SuS die Funktionen für die Knöpfe programmieren lässt.

Operationalisierte Lernziele

- ☐ Die SuS können mathematische Funktionen, wie bspw. lineare, quadratische und Potenzfunktionen, in einem Python-Programm implementieren und diese aufrufen, um einen Funktionswerte einer Variable zu übergeben.
- ☐ Die SuS können die Mitternachtsformel als eine Funktion in Python schreiben und die Lösung als String zurückgeben.
- ☐ Die SuS können in eignen Worten beschreiben, wozu ein `return`-Statement benötigt wird.
- ☐ Die SuS können anhand einer vorgegebene Funktion herleiten, was die Funktion beim Aufrufen macht und welchen Wert die Funktion zurück gibt.

Grober Lektionsablauf

Lektion 1/2: einfacher `return`-Befehl

Einstieg: Zu Beginn, als wir Befehle eingeführt haben, haben wir von Bausteinen gesprochen. Die Bausteine sollten auch eine Verwendung/Nutzen haben: Ein Befehl sollte etwas zurückgegeben (ein `return`-Statement enthalten) können. Die LP erklärt, wie man `return`-Statements in einen Befehl integriert. Daraufhin integrieren die SuS in ihrem Programm eine einfache mathematische Funktion, die nur aus einem `return`-Statement besteht. Als Ergänzung können sie weitere Funktionen implementieren, beispielsweise um die Fläche eines Rechtecks zu berechnen usw.

Lektion 2/2: `return` in Schleifen und Verzweigungen

In der zweiten Lektion sollen die SuS sich Überlegungen zu Schleifen und Verzweigungen machen. Wo platziert man das `return`-Statement in einer Schleife oder in einer Verzweigung? Im Anschluss sollen sie die Mitternachtsformel mithilfe einer Verzweigung programmieren. Um das Thema weiter zu vertiefen, gibt es zum Schluss noch weitere Übungen, wo sie den `return`.Wert einer Funktion bestimmen sollen.

Mögliche Schwierigkeiten & geeignete Massnahmen

1. Die Platzierung des `return`-Statments stellt einige Schwierigkeit dar. Die SuS können annehmen, dass der Code, der nach dem `return`-Statement geschrieben wird, trotzdem ausgeführt wird. Es besteht auch die Möglichkeit, dass sie das `return`-Statement aussserhalb des Befehlsblock angeben (Syntax/Abstand nicht einhalten). Ihnen ist vermutlich auch nicht bewusst, dass man mehrere `return` Statements einsetzen kann. Hier könnte man wiederum gezielt Übungen einsetzen, die diese Probleme aufzeigen: Bspw. den SuS eine Funktion geben, die einen Befehl nach dem `return`-Statement enthält und sie anhand des Outputs herleiten lassen, dass das `return` einer Abbruchbedingung gleicht, bzw. die SuS einen Vergleich zwischen zwei Funktionen machen lasen, die eine Verzweigung beinhalten, wobei sich die Funktionen grundsätzlich nur darin unterscheiden, dass die Platzierung des `return` anders ist und, falls nötig, zusätzliche Variablen eingesetzt werden können.

10 DL 9: Koordinaten & Zufallszahlen

DL 9

Thema

Die SuS beschäftigen sich mit der Integration von neuen Python-Modulen, spezifisch sollen Sie sich mit dem `random`-Modul. Dabei vertiefen sie ihre Kenntnisse zu Befehlen mit Parametern. Als Resultat generieren sie Zufallsbilder mit `turtle` und können Koordinaten der `turtle` ablesen und verändern.

Operationalisierte Lernziele

- ☐ Die SuS können in eignen Worten die Bedeutung von `from ... import ...` oder `from ... import *` erklären und Module wie `random` selbständig in ihren Programmen integrieren.
- ☐ Die SuS können den Befehl `x=random.randint(start, end)`, um Zufallszahlen zu generieren, in ein Programm einbauen und sinnvoll verwenden.
- ☐ Die SuS können das Ergebnis eines vorgegebenes Programm mit `turtle`-Befehlen (z.B. `forward()`), von Hand in einem Koordinatensystem einzeichnen und dadurch die Position der `turtle` am Ende des Programms bestimmen.
- ☐ Die SuS können die Position der `turtle` mithilfe folgender Funktionen verändern und mit `print(pos())` wiedergeben: `setX(), setY(), setPos(), setheading()`.
- ☐ Die SuS können mithilfe von `turtle`- und `random`-Befehlen, bzw. -Funktionen eigene Befehle, bzw. eigene Funktionen schreiben (mit oder ohne Parameter), die beim Aufrufen Zufallsbilder generieren. In ihren Befehlen sollten mindestens eine Schleife, ein `random`-Befehl, eine `turtle`-Farbänderung und eine Verzweigung vorkommen.

Grober Lektionsablauf

Lektion 1/2: Module und Koordinaten

Zu Beginn eine kurze Erklärung zu Modulen in Python mit Bezug auf die vorherigen Lektionen (vordefinierte und neue Befehle). Anschliessend sollen die SuS sich mit dem `random`-Modul vertraut machen indem sie eigene Befehle schreiben, welche Befehle des `random`-Moduls beinhalten. Zudem sollen sie sich mit dem Prinzip der Koordinaten vertraut machen, indem sie zuerst von Hand `turtle`-Porgramme ausführen und danach in eigenen Programmen die Position der `turtle` manipulieren und ablesen.

Lektion 2/2: Zufallsbild

In der zweiten Lektion sollen Sie den Inhalt der ersten Lektion nutzen, um eine Funktion (ohne `return`), zu schreiben, die nach dem Aufrufen ein zufällig generiertes Bild erzeugt.

Mögliche Schwierigkeiten & geeignete Massnahmen

1. Die SuS könnten möglicherweise vergessen, dass Befehle sequentiell abgearbeitet werden und könnten daher versuchen, neue Module am Ende des Programms einzufügen. Daher sollte man zu Beginn der DL nochmals erwähnen, dass der Code sequentiell abgearbeitet wird.
2. Den SuS alle Funktionen eines Moduls zu präsentieren oder Sie die Online-Dokumentationen (bspw. `random`-Dokumentation) durchlesen zu lassen, wäre für sie vermutlich ein "*overkill*". Stattdessen wollte Spickzettel mit den wichtigsten Funktionen angefertigt werden, welchen sie im Unterricht verwenden sollen, sowie einer Kurzbeschreibung zu den jeweiligen Funktionen. Die Beschreibung sollte folgende Punkte beinhalten: Wozu ist die Funktion nützlich? Wie viele Parameter muss man mitgeben? Welche Datentypen müssen die mitgegebenen Parameter haben? Eventuell kann man den SuS zunächst ein paar Beispiel-Codes geben, die sie auf ihre Funktionalität untersuchen sollen. Somit können sie sich beim Schreiben eines eigenen Befehls, an diesen Beispielen orientieren.

11 DL 10: Listen

DL 10

Thema

In dieser Lektion geht es um das Erstellen, Bearbeiten und Lesen von Listen.

Operationalisierte Lernziele

- ☐ Die SuS können in eignen Worten beschreiben was eine Liste und ein Index sind und deren Bedeutung anhand eines eignen Beispiels veranschaulichen.
- ☐ Die SuS können in Python eine Liste mit mehreren Werten erstellen, die Werte verändern, einen beliebigen Werte aus der Liste einer neuen Variable übergeben und einen Bereich von Indizes mit `[*:*]` angeben, um einen Teil der Liste auszugeben.
- ☐ Die SuS können mithilfe von `.append()` oder `.insert()` neue Elemente einer Liste hinzufügen.
- ☐ Die SuS können mithilfe einer `for`- Schleife die Elemente einer Liste mit Zahlen oder Tupeln von Zahlen durchlaufen und diese Daten nutzen, um die Position der `turtle` zu verändern.

Grober Lektionsablauf

Lektion 1/2: Listen

In der ersten Lektion wird zunächst das Konzept einer Liste eingeführt und mit Beispielen veranschaulicht. Anschliessend sollen die SuS unterschiedliche Übungen zu den Listen lösen, wobei sie zunächst Beispiele analysieren und anschliessend eigene Listen schreiben und deren Elemente mit `print()` ausgeben. Danach sollen sie Zufalls-listen erstellen, deren Inhalt sie wiederum mit einer `for`-Schliefe ausgeben sollen.

Lektion 2/2: Random `turtle`

In der zweiten Lektion sollen sie den Code aus der vorherigen Lektion anpassen, indem Sie Listen einbauen. Ziel ist es, dass Sie am Ende der Lektion ein Programm haben, dass ihnen eine oder mehrere Listen mit zufällige Zahlen oder Koordinaten erstellt, die wiederum genutzt werden, um ein Zufallsbild zu generieren (bspw. ein Blumenfeld).

Mögliche Schwierigkeiten & geeignete Massnahmen

1. SuS könnten zunächst die Vorstellung haben, dass eine Liste mit dem Index “1” beginnt und somit versuchen, dass erste Element einer Liste mit `mylist[1]` abzufragen. Eine graphische Darstellung, wobei die Indizes direkt unter der Liste angegeben sind, kann hier weiterhelfen.
2. Der Unterschied zwischen `.append()` und `.insert()` könnte möglicherweise unklar sein. Da die SuS bisher nie mit Objekten arbeiten mussten, könnte es zunächst verwirrend sein, warum man den Befehl mit einem Punkt ausführt (z.B. `mylist.append("hallo")`). Sie könnten versuchen die Funktion alleinstehend (`.append("hallo")`) oder mit der Liste als Parameter `.append("hallo", mylist)` aufzurufen. Auch hier kann es nützlich sein einen Spickzettel mit den wichtigsten Informationen (Anzahl Parameter, Parametertypen, Nutzen usw.) anzufertigen und ihnen Beispiele zu geben, die sie analysieren sollen und die ihnen aufzuzeigen, wie man die Funktionen einsetzen könnte.
3. Ein weiteres Problem könnte die Verschachtelung von Befehlen darstellen: Bspw. `mylist .append(randint(1,2))` Hier könnte man den SuS als Tipp mitgeben, damit es für sie übersichtlicher ist, mit Variablen zu arbeiten, bspw. `a = randint(1,2)` und dann `mylist.append(a)` zu schreiben.